Self-Stabilizing construction of Bounded Size Clusters

Colette Johnen
LRI, Univ. Paris-Sud, CNRS
F-91405 Orsay Cedex, France
colette@lri.fr

Le Huy Nguyen
LRI, Univ. Paris-Sud, CNRS
F-91405 Orsay Cedex, France
lehuy@lri.fr

Abstract

Clustering means partitioning nodes into groups called clusters, providing the network with a hierarchical organization.
A self-stabilizing protocol, 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 protocols are adapted to highly dynamic networks as ad hoc or sensors networks. In this paper, we propose a self-stabilizing clustering protocol. Our protocol guarantees a threshold ($SizeBound$) on the number of nodes that a clusterhead handle. Therefore, none of the clusterheads are overloaded at any time. The criterion of the clusterheads election is based on their $weight$ value, a general parameter that can be computed according to several node parameters as transmission power, battery power, ...

1. Introduction

An *ad hoc* network is a self-organized network, especially those with wireless or temporary plug-in connections. 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. In these networks, mobile nodes may move arbitrary often; thus, the network's topology may change rapidly and unpredictably. Self-configuring, self-healing and self-organizing management is needed in such networks.

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. It is experimentally shown that these protocols, when coupled with an ad hoc routing protocol, produce throughput improvements of up to 80% over the ad hoc routing protocol alone [6]. Members of a cluster can share resources such as software, memory space, printer, etc. Also, clustering facilitates the reuse of resources, which improves the system capacity. 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 protocols have been proposed during the last ten years [2, 3, 4, 7, 9, 17]. A detailed survey on clustering algorithms can be found in [17]. The clustering protocols in [2, 9] construct a spanning tree. Then the clusters are constructed on top of the spanning tree. The clusterheads set does not necessarily form a dominating set (i.e., a node can be at distance greater than 1 from its clusterhead). In [17], the performance of the node-degree based clustering algorithms are evaluated. In [7], the weight-based distributed clustering protocol 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 [4], a Distributed and Mobility-Adaptive Clustering protocol, 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 protocol, called Generalized DMAC (GDMAC), is proposed

in [3].

A system is self-stabilizing [8] 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 protocols does not depend on initialization of variables, and a self-stabilizing protocol converges to some predefined stable configuration starting from an arbitrary initial one. Self-stabilizing protocols are thus, inherently tolerant to transient faults in the system. Many self-stabilizing protocols 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 converge to a configuration consistent with the new topology. Several self-stabilizing protocols for cluster formation and clusterhead selection have been proposed [5, 11, 12, 14, 16]. [14] presents a robust self-stabilizing version of DMAC under the synchronous schedule. A self-stabilizing version of DMAC and GDMAC is presented in [12]. A robust and self-stabilizing version of GDMAC is presented in [11]. In [5], a self-stabilizing link-cluster protocol 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 protocol requires three types of messages, our protocol adapted to message passing model requires one type of messages. A self-stabilizing protocol for cluster formation is presented in [16]. A density criteria (defined in [15]) is used to select clusterhead: a node v chooses in its neighborhood the node having the biggest 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 protocol builds clusters on local information; thus, it requires only communication between nodes at distance 1 of each others.

Contribution: In this paper, we propose a self-stabilizing clustering protocol which takes into consideration the number of nodes a clusterhead can handle, and transmission power, mobility, and battery power of the nodes. Our protocol is based on local properties. Our protocol guarantees that network nodes are partitioned into clusters: each cluster has at most $SizeBound$ nodes. Thus, our protocol 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 to 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). A trivial solution guaranteeing the first requirements is that every node is the head of cluster having no other member. This solution does not provide an useful hierarchical organization. Therefore, we minimize the number of clusterheads in a neighborhood. Several clusterheads are neighbor only if it is necessary: no clusterhead can join another cluster and gives up its responsibility.

Our protocol is designed for the state model. Nevertheless, it can be easily transformed into a protocol 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 protocol is described in section 4. The protocol proof is presented in section 5. We conclude by a discussion about the convergence time 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* [13] 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 problem specification 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 stabilizing to $\mathcal{SP}$ - a predicate on configurations - 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. *$\forall c \in \mathcal{L}$, c satisfies the predicate $\mathcal{SP}$.*

3. Well-balanced Clustering for network

We consider weight-based 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 loss of 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).

Protocol 1 : definition of Constants, Variables and Macros 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 where v is an ordinary node, this list should be empty.

$S_v : \mathbb{N}$; - is the size of v's cluster. If v is an ordinary node then S_v should be 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 $CD0_v$; **fi**
  if $CD_v \subseteq CD1_v \cup \{w \in N_v : Head_w = v\}$ **then** $CD2_v := CD1_v$;
  else $CD2_v := \emptyset$; **fi**
end

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 guaran-

tees 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 protocol building bounded size clusters

The code of the "self-stabilizing construction of bounded size clusters" is presented in Protocol 2; the constants, the variables, and macros are defined in Protocol 1.

Notation 1
$Cluster_v$ is the set of nodes belonging to the v's cluster (having chosen v as their clusterhead) : $Cluster_v := \{z \in N_v : Head_z = v\}$. The safety predicate is defined as : $P_s(v) \equiv |CD_v \cup Cluster_v| \leq SizeBound$.

Our algorithm is a distributed algorithm requiring only communication between neighbors. Thus, a node gets partial information about the network state. To obtain a complete knowledge of the network state requires communication, memory space, and time that are proportional to the network size. Moreover, this knowledge will be not accurate in highly dynamic network, or large scale network where topology changes happen very often. Therefore, centralized algorithms (i.e., algorithms requiring a complete knowledge of the network) are not suitable for dynamic and large scale networks, even if they build very efficient clustering structures, as the algorithms proposed in [1].
The main idea of the algorithm is that a node becomes and stays clusterhead only when it cannot join one of its neighbor cluster without violating the well-balanced clustering properties.
To prevent that the size condition is violated, a node u cannot freely join a cluster : u needs to have the permission of the clusterhead. More precisely, only the nodes belonging to the set CD_v may join v's cluster. The goal of this mechanism is to enforce the size condition. To achieve that goal, any clusterhead v will verify a property stronger than the size condition, the safety predicate (defined above). A cluster whose the head satisfying the safety predicate verifies the size condition after any computation step. A contrario, a cluster whose the head v does not satisfy the safety predicate, may not verify the size condition, after a specific computation step (i.e., all nodes belonging to CD_v join v's cluster). Therefore, we have designed our protocol is such a way that the safety predicate is eventually always satisfied by all clusterheads.
For each clusterhead v, the macro $CD2_v$ is used to set CD_v value. $CD0_v$ is the set of v's neighbors wanting to join

Protocol 2 : Self-Stabilizing Clustering protocol on v

Predicates

$\mathbf{G_0}(v) \equiv [(S_{Head_v} > SizeBound) \vee (Head_v \notin N_v) \vee (Ch_{Head_v} = F) \vee (w_{Head_v} < w_v)]$
$\mathbf{G_{11}}(v) \equiv (Ch_v = F) \wedge (N_v^+ = \emptyset) \wedge \mathbf{G_0}(v)$
$\mathbf{G_{12}}(v) \equiv (Ch_v = T) \wedge (N_v^+ = \emptyset) \wedge (Head_v \neq v)$

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

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

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

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

Rules

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

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

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

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

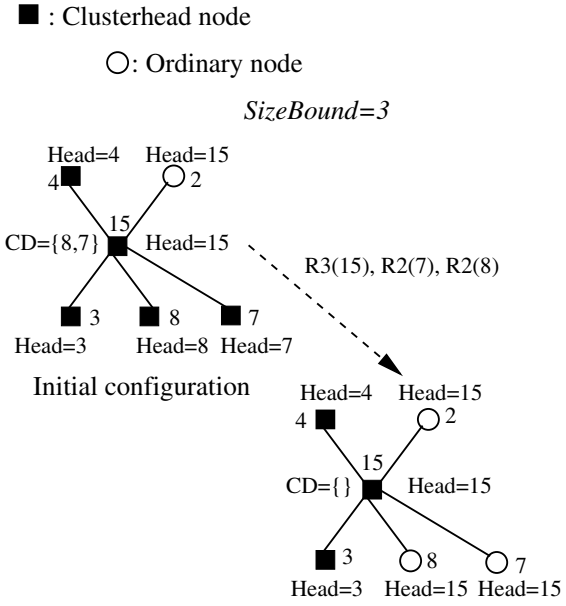


Figure 1. Illustration of CD value computation

the v's cluster (i.e., their clusterhead has a smaller weight than v' weight). Only a subset of $CD0_v$ is selectionned to be in $CD2_v$. This subset of $CD0_v$ is computed to ensure that the safety predicate would be satisfied by v after any computation step from the current configuration. 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 satisfies the predicate $P_s(v)$). In this configuration, the node having the biggest weight is node 15. All its neighbors want to join its cluster $CD0_{15} = N_{15} - Cluster_{15} = \{3, 4, 7, 8\}$. Node 15 can allow only two nodes to join its cluster (because $SizeBound$ = 3). The two selected nodes are the nodes of $CD0_{15}$ having the two-smallest weight. Therefore $CD1_{15} = \{3, 4\}$. In the Figure 1, during the first computation step, all nodes of $CD0_{15}$ join $Cluster_{15}$. At the end of this computation step, $Cluster_{15} = \{2, 7, 8\}$. If CD_{15} would have the value $CD1_{15} = \{3, 4\}$. After this step, $|CD_{15} \cup Cluster_{15}|$ could have the value $5 = |\{2, 3, 4, 7, 8\}|$, the safety predicate would not be satisfied: after another computation step, the 15's cluster could have 5 members. Therefore, CD_{15} is set to $\emptyset$, and not to $CD1_{15}$.

The macro N_v^+ helps us to guarantee the *clusterhead neighboring* condition. The set N_v^+ is not empty, if the clusterhead v does not verify the clusterhead neighboring condition. In this case, v is enabled ; it has to join an existing cluster in its neighborhood. 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 biggest weight as its clusterhead (rule R_2). The N_v^+ set contains 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 joins their cluster (i.e., v belongs to their CD set).

We split the possible 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 (S or CD value) 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.

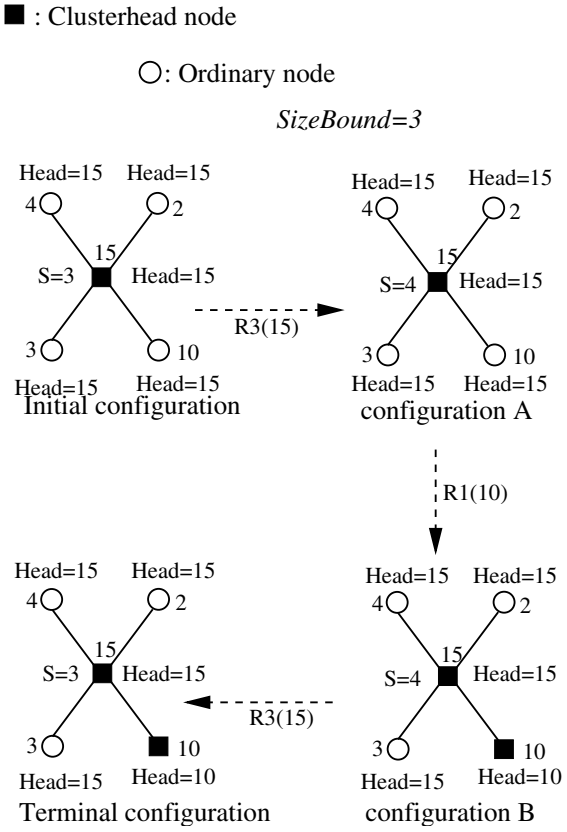


Figure 2. Illustration of Self-stabilizing construction of bounded clusters

Protocol 1 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 the safety predicate holds continuously till the protocol converges to a legitimate configuration, once a configuration reaching this predicate is reached. 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.

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 in a legitimate configuration, the well-balanced clustering properties are verfied.

5.1. Safety

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

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 v's cluster in c.*

Observation 2 *Assume that we have a computation step $c_1 \xrightarrow{cs} c_2$. 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*
  $|CD_v(c_2) \cup Cluster_v(c_1)| \leq SizeBound$
- $CD1_v(c_1) \cap Cluster_v(c_1) = \emptyset$.

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 $c_1 \xrightarrow{cs} c_2$. We will prove that $P_s(v)$ holds in c_2.
In any case ($CD_v(c_2) = CD_v(c_1)$, $CD_v(c_2) = \emptyset$, or $CD_v(c_2) = CD2_v(c_1)$), we have $|CD_v(c_1) \cup Cluster_v(c_1)| \leq SizeBound$ and $|CD_v(c_2) \cup Cluster_v(c_1)| \leq SizeBound$.
Case 1: $Cluster_v(c_2) \subseteq Cluster_v(c_1)$. $P_s(v)$ is satisfied in c_2.
Case 2: $Cluster_v(c_2) \subseteq \{u_1, .. u_m\} \cup Cluster_v(c_1)$.
According to the observation 2, we have $\{u_1, .. u_m\} \subset CD_v(c_1)$.
If $CD_v(c_2) = CD_v(c_1)$ or $CD_v(c_2) = \emptyset$ then $|CD_v(c_2) \cup Cluster_v(c_2)| \leq |CD_v(c_1) \cup \{u_1, .. u_m\} \cup Cluster_v(c_1)| \leq |CD_v(c_1) \cup Cluster_v(c_1)| \leq SizeBound$. $P_s(v)$ is satisfied in c_2.
If $CD_v(c_2) = CD2_v(c_1)$ then (according to the $CD2_v$ definition) $CD_v(c_1) \subset (CD_v(c_2) \cup Cluster_v(c_1))$.
Thus $|CD_v(c_2) \cup Cluster_v(c_2)| = |CD_v(c_2) \cup \{u_1, .. u_m\} \cup Cluster_v(c_1)| \leq |CD_v(c_2) \cup Cluster_v(c_1)| \leq SizeBound$.
$P_s(v)$ is satisfied in c_2. □

5.2. Convergence

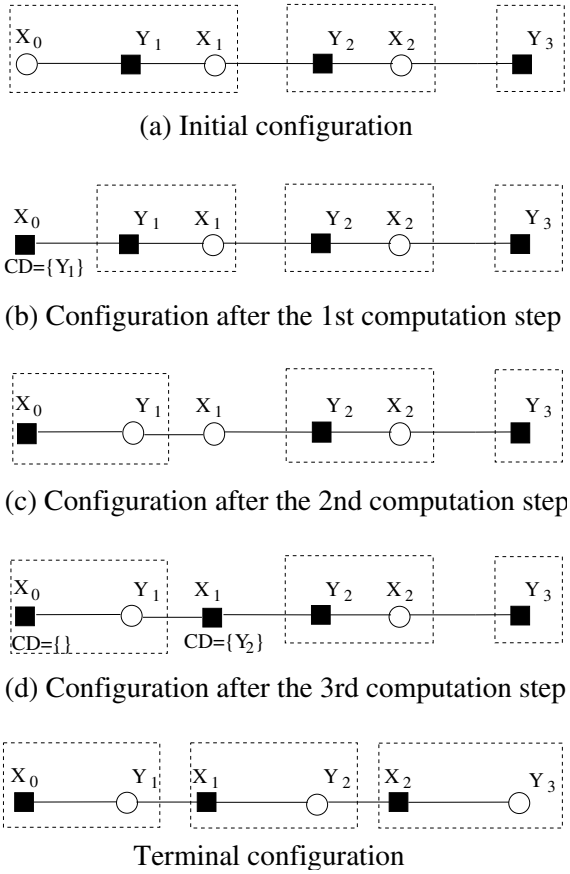
In this subsection, some of the proofs are omitted dee to lack of space, they can be found in [10].

Lemma 2 *$P_g \equiv (G_{12}(v) = F) \wedge (G_4(v) = F)$.*
$A_4 = A_2 \cap \{c \in \mathcal{C} \mid \forall v : Head_v \in N_v \cup \{v\}$ and $|Cluster_v| \leq SizeBound$ and $P_g(v)$ is satisfied$\}$ is an attractor from $\mathcal{C}$.

■ : Clusterhead node

○ : Ordinary node

Sizebound=3



(a) Initial configuration

(b) Configuration after the 1st computation step

(c) Configuration after the 2nd computation step

(d) Configuration after the 3rd computation step

Terminal configuration

Figure 3. Stabilization time.

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 3
$L"_0 = A_4$ *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 3 *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 4 *Let c_1 be a configuration of L_{i+1}. Let cs be a computation step from c_1: $c_1 \xrightarrow{cs} c_2$. We have $Cluster_{vh_i}(c_1) \subseteq Cluster_{vh_i}(c_2)$.*

Proof:
Let u be a node of $Cluster_{vh_i}$ (i.e., $Head_u = vh_i$). In L_{i+1}, a 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 verified. In L_{i+1}, $G_1(u)$ is not verified. We conclude that the node u stays in the Cluster of vh_i forever. □

Lemma 5 *Let c_1 be a configuration of L_{i+1}. Let cs be a computation step from c_1: $c_1 \xrightarrow{cs} c_2$. We have $CD1_{vh_i}(c_2) \neq CD1_{vh_i}(c_1)$ if only if $Cluster_{vh_i}(c_1) \neq Cluster_{vh_i}(c_2)$.*

Lemma 6 *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, only the nodes of V_i may be in the Cluster of vh_i, therefore, the size of $Cluster_{vh_i}$ is bounded by $SizeBound_i$.
As no node can quit $Cluster_{vh_i}$ (lemma 4), $Cluster_{vh_i}$ will eventually stay identical forever. According lemma 5, $CD1_{vh_i}$ will eventually stay identical forever.
Once $CD1_{vh_i}$ is set up, if, $CD_{vh_i} = CD1_{vh_i}$, $R_3(vh_i)$ is never enabled (i.e., CD_{vh_i} stays always equal to $CD1_{vh_i}$). Once, $CD1_{vh_i}$ is set up, if $CD_{vh_i} \neq CD1_{vh_i}$ then $R_3(vh_i)$ is enabled forever. By fairness, $R_3(vh_i)$ action will be eventually performed. After at most two $R_3(vh_i)$ actions, we have $CD_{vh_i} = CD1_{vh_i}$. We conclude that any computation has a suffix where CD_{vh_i} stay equal to $CD1_{vh_i}$. In this suffix, the size of CD_{vh_i} is equal to $SizeBound_i - |Cluster_{vh_i}|$

Assume that size of $Cluster_{vh_i}$ is forever smaller than $SizeBound_i$. In that case, the computation has a suffix where a node u will stay forever in the set 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. 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: $Cluster_{vh_i}$ is modified. There is a contradiction. □

Lemma 7 *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 attractors for any value of $1 \geq i \leq j$. 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 Lemma 2, in a terminal configuration $S_z \leq SizeBound$, thus, the *size* condition is satisfied.
Let v be a clusterhead, neighbor of z such that $w_v > w_z$. Notice that $CD0_v$ is not empty (it contains z). $G_3(v)$ is not verified. Thus, $CD2_v$ is equal to CD_v. We have $CD_v = \emptyset$, thus, $CD2_v = CD1_v = \emptyset$. We conclude that $Size_v = SizeBound$. Thus, every clusterhead v in z' neighborhood verifies the following predicate: $(w_v \leq w_z)$ or $(S_v = SizeBound)$. Therefore, the *clusterhead neighboring* condition is satisfied. □

6. Convergence times

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 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 join 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] B. Awerbuch and D. Peleg. Sparse partitions (extended abstract). In *the 31st Annual IEEE Symposium on Foundations of Computer Science (FOCS90)*, pages 503–513, 1990.

[2] 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.

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

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

[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] E. M. Belding-Royer. Multi-level hierarchies for scalable ad hoc routing. *Wireless Networks*, 9(5):461–478, 2003.

[7] 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.

[8] E. Dijkstra. Self stabilizing systems in spite of distributed control. *Communications of the Association of the Computing Machinery*, 17:643–644, 1974.

[9] 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.

[10] C. Johnen and L. Nguyen. Self-stabilizing bounded size clustering algorithm. Technical Report 1464, L.R.I, 2006.

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

[12] C. Johnen and L. H. 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] F. G. Nocetti, J. S. Gonzalez, and I. Stojmenovic. Connectivity based k-hop clustering in wireless networks. *Telecommunication Systems*, 22(1–4):205–220, 2003.