Robust Self-Stabilizing construction of bounded size weight-based clusters

Rapport de Recherche LRI N^o 1518

Colette Johnen
LaBRI, Univ. Bordeaux, CNRS
F-33405 Talence Cedex, France
johnen@labri.fr

Fouzi Mekhaldi
LRI, Univ. Paris-Sud, CNRS
F-91405 Orsay Cedex, France
mekhaldi@lri.fr

Abstract

An Ad hoc network consists 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 partitionning network nodes into non-overlapping groups called clusters. Clusters give a hierarchical organisation to the network that facilitates network management and that increases its scalability.
In this paper, we propose a clustering protocol for Ad hoc network that ensures fault-tolerance, load-balancing, best choice of clusterheads, and reliability.
The clusterheads are selected according to their weight (node's parameter). The higher the weight of a node, the more suitable this node is for the role of clusterhead. In Ad hoc network, the amount of bandwidth, memory and processing capacity, or battery power of a node can be used to determine the weight values.
The load balancing is achieved by building bounded size clusters. Our protocol guarantees a threshold ($SizeBound$) on the number of nodes that a clusterhead handle. So, none of the clusterheads are overloaded. The fault-tolerance and the reliability, in our protocol, are achieved by the robust self-stabilization property.
The robustness property guarantees that starting from an arbitrary configuration, after four rounds, a safe configuration is reached. In a safe configuration, a minimum service is achieved: the network is partitioned into bounded size clusters having a leader. During the convergence to a legitimate configuration, the safety property stays verified. Thus, the minimal service is continuously provided. In a legitimate configuration, the optimal service is achieved: the clusters satisfy the well-balanced clustering properties (i.e. best choice of clusterheads, and their number is locally minimized).

Keywords: Ad hoc network, Clustering, Load-Balancing, Fault-tolerance, Self-Stabilization, Robustness, Reliability.

Résumé

Un réseau Ad hoc est un ensemble d'hôtes (noeuds) communiquant entre eux via un réseau sans fil sans infrastructure fixe. Dans un tel réseau, ce sont les noeuds qui prennent en charge la gestion du réseau (routage, bande passante, ... etc.), et ils doivent prendre des décisions collectives d'une manière répartie. Le problème d'agrégation consiste à partitionner les noeuds d'un réseau en grappes disjointes. Ces grappes donnent au réseau une organisation hiérarchique, ce qui facilite la gestion du réseau, et renforce son extensibilité.
Dans cet article, nous proposons un protocole d'agrégation, pour les réseaux Ad hoc, assurant la tolérance aux pannes, la répartition de la charge, un meilleur choix des responsables de grappes, et de la régularité.
Les responsables de grappes sont choisis selon leur poids (paramètre mesurant l'aptitude d'un noeud à devenir responsable). Plus le poids d'un noeud est grand, plus ce noeud est apte à être responsable. Dans un réseau Ad hoc, la bande passante, la capacité de stockage et de calcul, et l'énergie d'un noeud peuvent être utilisées pour déterminer le poids de ce noeud.
La construction de grappes ayant une taille bornée assure la répartition de la charge. Dans notre protocole, chaque grappe contiendra au maximum $SizeBound$ membres. Donc, aucun responsable ne sera surchargé. La tolérance aux pannes et la régularité sont assurées par la propriété de Stabilisation Robuste.
La propriété de Robustesse garantie qu'à partir de n'importe quelle configuration, une configuration sûre sera atteinte au bout de 4 cycles maximum. Dans une configuration sûre, un service minimum est offert : le réseau est divisé en grappes de taille bornée avec un responsable élu au sein de chaque grappe. Pendant la convergence vers une configuration légitime, la propriété de sûreté reste vérifiée. Donc, le service minimum est continuellement offert. Dans une configuration légitime, un service optimale sera offert : les grappes vérifient les propriétés de grappes équilibrées (c-a-d un choix des responsables localement optimale, et leur nombre est aussi localement optimale).

Mots-clés: Réseau Ad hoc, Agrégation, Répartition de charge, Tolérance aux pannes, Auto-Stabilisation, Robustesse, Régularité.

1 Introduction

A Mobile Ad hoc NETwork (MANET) is an infrastructureless wireless network. It consists of mobile hosts which can move arbitrarily, and communicate between them via wireless links without any pre-existing fixed infrastructure. The absence of infrastructure in MANET may arise in emergency situations, remote regions, hostile areas, or due to the latency period and financial costs that are involved in the deployment of a fixed infrastructure. However, owing to the absence of infrastructure, nodes in MANET need to behave as routers in order to ensure distant communications.
In the flat architecture of MANET, all nodes are considered equal and take the same part in the routing and forwarding task. This kind of organisation is not scalable on large scale network due to resource consumption (energy, and bandwidth), and communication overhead. To provide scalable solutions in many large scale networks, a hierarchical structure has been used (for instance, in [18]).

The clustering is a hierarchical organization which consists of partitioning network nodes into groups called clusters. Each cluster has a single clusterhead that acts as local coordinator of cluster. Several tasks can be achieved by a clusterhead like: resolving channel scheduling, performing power measurement/control, maintaining time division frame synchronization, and enhancing the spatial reuse of time slots and codes.
Clustering has other advantages. It limits the amount of topology information stored and maintained at nodes, since nodes outside of a cluster usually do not need to know the detailed state of this cluster. When topology changes occur in a cluster, only nodes in this cluster has to change their topology information. Moreover, clustering reduces the amount of routing information stored in nodes; and decrease the transmission overhead.

Numerous clustering protocols are proposed [2, 4, 10, 19, 22], since the clustered architecture is more effective than the flat architecture. In [10], Lowest-ID and Highest-Connectivity protocols are presented. In the Lowest-ID protocol, the node having the lowest identity in its neighborhood is selected as clusterhead. Whereas, Highest-Connectivity chooses as clusterheads the nodes having the highest degree in their neighborhood. In [19], a network architecture for MANET is proposed, where nodes are organised into nonoverlapping clusters, and the clusterheads are selected according to their identity. In [2], the weight notion is introduced to generalize the selection criteria of clusterheads; the suitability of a node to become clusterhead is based on the node's weight value. A node is chosen to be clusterhead if its weight is higher than any of its neighbors weight. In [4], a clustering protocol based on combined weight metric is proposed. The combined metric takes into account several system parameters like: node's degree, battery power, mobility and some one. In [22], a weight-based clustering protocol is presented, the weight of a node dependS on its degree and the degree of its neighbors. A survey on clustering protocols can be found on [1].

A technique for designing solutions that tolerate transient faults is the self-stabilization. Starting from any configuration, a self-stabilizing system reaches in a finite time a legitimate configuration. The system behaves properly from any legitimate configuration. Conversely, a non self-stabilizing system driven to an illegitimate configuration by some perturbations, may not recover to a correct behavior. Self-stabilizing protocols are attractive because they need not be initialized: they converge from any state to a legitimate configuration. They have also the ability to adapt to network topology changes. If the current configuration is inconsistent with the network topology, the self-stabilizing protocol eventually converges from it to a legitimate configuration.
In [24], a self-stabilizing protocol that constructs a minimal dominating set is presented. In [9, 12], self-stabilizing protocols building a connected dominating set are presented. A set is dominating if each node of the network is either member of this set, or it has a neighbor

that is member of the set. A dominating set is minimal if any node leaves the set then the set is no more a dominating set. The clusterheads set may be a dominating set in order to to build 1-hop clusters (i.e., nodes are at distance 1 of their clusterhead). In [11], a self-stabilizing protocol to construct a maximal independent set (MIS) is presented. In [8], a probabilistic self-stabilizing MIS protocol is presented; in the average the MIS is built in $O(lg(|V|))$ rounds, where $|V|$ is the network size. (the nodes in this set are not neighbors, and the set is maximal to this property). In [3], a self-stabilizing link-cluster protocol is proposed under the asynchronous message-passing system model. In the obtained clusters, each node is in at most two hops of its clusterhead. A self-stabilizing clustering protocol is presented in [21]; the density criteria (defined in [20]) is used to select the clusterheads.

The period of time where a self-stabilizing system converges to a legitimate configuration is called the convergence period. Usually, during the convergence period, a self-stabilizing system does not guarantee any property. In addition, the duration of convergence period may be proportional to the size of the network; particularly, in weight-based clustering protocols. Thus, in large scale networks, the convergence towards a legitimate configuration can require a long time. In order to overcome this drawback, we are interested to the robust stabilization. The robust stabilization guarantees that from an illegitimate configuration and without occurrence of faults, the system reaches quickly a safe configuration. From a safe configuration, all reached configurations are safe. Thus, the safety property stays always verified.
In [17] a robust self-stabilizing protocol building a minimum connected dominating sets is proposed. In a safe configuration, the built set is a dominating set. In [16], a robust self-stabilizing version of DMAC (presented in [2]) under synchronous scheduler is presented. A robust self-stabilizing weight-based clustering protocol for ad hoc networks is proposed in [14]. It is a robust self-stabilizing version of GDMAC. In [16] and [14], a configuration is safe iff the network is partioned into clusters.

In some solutions described previously, once a clusterhead is selected, all its neighbors must be ordinary nodes. So, two clusterheads are never neighbors. A problem can arise where nodes are non-uniform distributed in the whole area. If a certain zone becomes densely populated with nodes, the clusterhead might not be able to handle all the traffic generated by the nodes of its cluster. In addition, the power consumption of a clusterhead depends proportionally on the number of nodes of its cluster. Thus, controlling the number of nodes in a cluster will extend its clusterhead's lifetime, and will improve the stability of the cluster. For this reason, our protocol keeps the number of nodes in a cluster inferior to a pre-defined threshold, called *SizeBound*. Thus, the clusters have bounded size, and the clusterheads are not overloaded. In [5], the Weighted Clustering Algorithm (WCA) is presented. It builds bounded clusters. In [23], the obtained clusters have a size bounded by a lower and an upper bound.

In this paper, we propose the first robust self-stabilizing protocol that constructs bounded size weight-based clusters. The obtained clusters satisfy the *well-balanced clustering properties*, defined in [13]:
(i). Each node belongs to a cluster.
(ii). The size of each cluster is less than *SizeBound*.
(iii). Two clusterheads are neighbors only if the cluster of the one having the highest weight contains *SizeBound* members.

The protocol presented in [13] is self-stabilizing, but it is not a robust self-stabilizing protocol. During the convergence period, a node may not belong to a cluster even if it belongs initially to a well-formed cluster (i.e., satisfies the well-balanced clustering properties). In this paper, the presented protocol is a robust self-stabilizing one. From an illegitimate configuration, our protocol reaches a safe configuration in a constant number of rounds, and from this configuration, the protocol provides a minimum service: each node belongs to a cluster having a leader,

and the size of each cluster is less than $SizeBound$. Notice that, in a safe configuration, the well-balanced clustering properties may not be verified.
If no fault occurs for enough period of time, our protocol converges to a legitimate configuration in $O(|V|)$ rounds. In a legitimate configuration, the system provides the optimal service, i.e., the clusters satisfy the well-balanced clustering properties. During the convergence to a legitimate configuration, the safety property is preserved, i.e., the network stays partitioned into clusters having less than $SizeBound$ members.

2 Model and Concepts

In this section, we give some definitions that are used throughout the paper.

2.1 Model

A distributed system S is modelled by an undirected graph $G = (V, E)$ in which, V is the set of (mobile) nodes and E is the set of edges. There is an edge $\{u, v\} \in E$, if and only if u and v can communicate between them (links are bidirectional). In this case, we say that u and v are neighbors, and we note by N_v the set of neighbors of the node v.
Due to mobility of nodes, the neighborhood N_v of a node v can change in the time. In this paper, we assume that at any moment N_v contains the current neighbors of v, i.e., it is always consistent with the current graph G.

Every node v in the network is assigned a unique identifier id. For simplicity, we identify each node with its identity id, and we denote both with v.
We consider a weighted network, i.e., a weight w_v (a real number) is assigned to each node v. For the sake of simplicity, in this paper we assume that the nodes weight are different (the tie in node's weight could be broken by the id).

We use the *state model* of computation introduced in [6], in which each node v maintains some local variables. The node v can read its own variables and those of its neighbors, but can modify only its own one.
The *state* of a node is defined by the values of its local variables. A *configuration* of the system S is an instance of the node states. The *program* of each node is a set of *rules*. Each rule has the following form: $Rule_i : Guard_i \longrightarrow Action_i$. The *guard* of a rule of a node v is a Boolean expression involving the local variables of v, and those of its neighbors. The *action* of a rule of v updates one or more variables of v. A rule can be executed only if it is *enabled*, i.e., its guard evaluates to true. A node is said to be enabled if at least one of its rules is enabled. In a *terminal configuration*, no node is enabled.
A *computation step* $c_i \rightarrow c_{i+1}$ consists of one or more enabled nodes executing a rule. A *computation* is a sequence of configurations $e = c_0, c_1, ..., c_i, ...$, where c_{i+1} is reached from c_i by one computation step: $\forall i \geqslant 0, c_i \rightarrow c_{i+1}$. We say that a computation e is maximal if it is infinite, or if it reaches a terminal configuration.
A computation is *fair*, if for any node v that is continuously enabled along this computation, eventually performs an action.
We note by $\mathcal{C}$ the set of all possible configurations, and by $\mathcal{E}$ the set of all possible computations of the system S. The set of computations starting from a particular configuration $c \in \mathcal{C}$ is denoted $\mathcal{E}_c$. $\mathcal{E}_A$ is the set of computations where the initial configuration belongs to the configurations set $A \subset \mathcal{C}$.

We say that a node v is neutralized in the computation step $c_i \rightarrow c_{i+1}$, if v is enabled in c_i and not enabled in c_{i+1}, but does not execute any action during this computation step. The neutralization of a node represents the following situation: at least one neighbor of v changes its state between c_i and c_{i+1}, and after this change, the guard of all actions of v are false.

We use the notion of *round* [7] to measure the time complexity. We say that the prefix $e' = c_i, c_{i+1}, ..., c_j$ of a computation $e = c_1, ..., c_j, ...$ is a round, if the following conditions hold:

1. Every node v that is enabled in c_i, either executes or becomes neutralized during some step of e'.

2. The prefix $c_i, ..., c_{j-1}$ does not satisfy the condition 1.

The round complexity of a computation is the number of disjoint rounds in this computation.

2.2 Self-Stabilization

A distributed system is called self-stabilizing if and only if without occurrence of faults, it converges to a legitimate configuration regardless of its initial one, and it remains in a legitimate configuration till no fault occurs.

In Ad hoc network, the events like node arrival, node departure, communication link failure, network merging and so one, are possible at any time. These events cause changes in the network topology, and possibly in the legitimacy of the system configuration. One example in which topology changes falsify the legitimacy of the system configuration is the clustering task. A clustering protocol ensures that each node affiliates with a clusterhead which is in a closed proximity. So, the legitimacy of a configuration depends on the current topology configuration, and topology changes could lead the system to an illegitimate configuration. Therefore, self-stabilizing clustering protocols designed for such networks should tolerate such events.

We use the *attractor* notion [15] to define self-stabilization. Informally, a set of configurations B satisfying a property $\mathcal{P}$ is said *attractor*, if and only if it is guaranteed that the system converges to a configuration c of B in a finite time from any arbitrary configuration. Furthermore, once a configuration c of B is reached and without occurrence of faults, all reached configurations from c belong to B (the property $\mathcal{P}$ stays verified).

Definition 1 ***(Attractor)***. *Let B_1 and B_2 be subsets of configurations of $\mathcal{C}$. B_2 is an attractor from B_1, if and only if the following conditions hold:*

- ***Convergence:***
  $\forall e \in \mathcal{E}_{B_1}(e = c_1, c_2, ...), \exists i \geqslant 1 : c_i \in B_2$.
  $\forall c \in B_1$, *If* $(\mathcal{E}_c = \emptyset)$ *then* $c \in B_2$.
- ***Closure:*** $\forall e \in \mathcal{E}_{B_2}(e = c_1, ...), \forall i \geqslant 1 : c_i \in B_2$.

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

- *$\mathcal{L}$ is an attractor from $\mathcal{C}$.*
- *All configurations of $\mathcal{L}$ satisfy the specification problem.*

2.3 Robustness

A self-stabilizing protocol guarantees to reach a legitimate configuration. However, self-stabilizing protocols do not guarantee any property during the convergence period. Moreover, the convergence time of a self-stabilizing weight-based clustering protocol is intrinsically proportional to the network diameter. So, in large scale networks, the convergence may require a long time.

Our protocol is self-stabilizing. And, it also ensures a safety property during the convergence period. The safety property is defined in such a way that the system still performs

correctly its task once the safety property is reached. From an illegitimate configuration, our protocol reaches a safe configuration after four rounds. In a safe configuration, our protocol provides a minimum service: each node belongs to a cluster, and the size of each cluster is less than *SizeBound*. Along the convergence to a legitimate configuration, the safety property is preserved. Once a legitimate configuration is reached, the optimal service is provided: the clusters satisfy the well-balanced clustering properties. Moreover, the safety predicate stays verified after some changes in the network topologies.

Definition 3 ***(Robustness under Input Change [14])***. *Let $\mathcal{SP}$ be the safety predicate, that stipulate the safe configurations. Let $\mathcal{IC}$ be a set of input changes that can occur in the system. A self-stabilizing system is robust under any input changes of $\mathcal{IC}$ if and only if the set of configurations satisfying $\mathcal{SP}$ is:*

- *closed under any computation step.*
- *closed under any input changes of $\mathcal{IC}$.*

3 Robust Self-Stabilizing protocol for Bounded Size Weight-based Clusters

3.1 The protocol goal

Our clustering protocol partitions the nodes of the network into clusters (i.e. each node belongs to one and only one cluster). Each cluster has a single leader (named clusterhead) that acts as the local coordinator of its cluster. To reduce the intra-clusters communication cost, each clusterhead is at distance 1 of the ordinary nodes in its cluster. Thus, our protocol builds 1-hop clusters; the set of clusterheads is a dominating set.

A node decides its own role knowing solely the state of its neighbors at one hop. Therefore, the decision of the node's role is taken quickly, and with a little communication overhead.

Our protocol is weight based: the clusterhead selection criteria is based on the weight of nodes. Each node has a weight w representing its capacity to be clusterhead. The higher the weight of a node, the more suitable this node is for the role of clusterhead.
A significant node's weight can be obtained by a sum of different normalised parameters like: node mobility, memory and processing capacity, bandwidth, battery and transmission power, and so one. The computation of the weight value is out the scope of this paper. Therefore, the weight of a node is an input value that can increase or decrease, reflecting the changes in the node's status.

The proposed clustering protocol provides bounded size clusters; at most *SizeBound* ordinary nodes can be in a cluster. This limitation on the number of nodes that a clusterhead handle, ensures the load balancing over the network: no clusterhead is overloaded at any time.
As clusters have a bounded size and the network topology is arbitrary, it is necessary to allow clusterheads to be neighbors: the set of clusterheads may not be an independent set.

Our protocol provides clusters satisfying the following *well-balanced clustering properties*:

- **Affiliation condition:** each ordinary node affiliates with a neighboring clusterhead, such that the weight of its clusterhead is greater than its weight.
- **Size condition:** each cluster contains at most *SizeBound* ordinary nodes.
- **Clusterhead neighboring condition:** if a clusterhead v has a neighboring clusterhead u such that $w_u > w_v$, then the size of u's cluster is *SizeBound*.

The first condition ensures that each node belongs to a cluster, and each node can communicate directly with its clusterhead (they are neighbors). Moreover, it ensures that the weight of

an ordinary node is smaller than the weight of its clusterhead (i.e. the clusterhead is the node having the highest weight of the cluster). The second condition ensures that a clusterhead will be not overburden by the management workload of its cluster: each cluster has at most $SizeBound$ members (the cluster management workload is proportional to the cluster size). The third condition limits the number of clusterheads. A node v stays clusterhead only if it cannot join another cluster in its neighborhood. If it changes of cluster then its new cluster would violate the size condition, or the affiliation condition.

3.2 The proposed protocol

The protocol's constants, variables, and macros are presented in the Protocol 1. The predicates and rules are illustrated in Protocol 2. We note by N_v the set of v's neighbors, and we note by $Cluster_v$ the v's cluster, i.e., the set of nodes having chosen v as their clusterhead: $Cluster_v = \{z \in N_v : Head_z = v\}$.

Protocol 1 : Variables and macros on node v.

Constants

$w_v \in \mathbb{R}$; The weight of node v.
$SizeBound \in \mathbb{N}$; The upper bound on a cluster size.

Local variables

$Ch_v \in \{T, F, NF\}$; Indicates the role of v.
$Head_v \in \{IDs\}$; The clusterhead's identity of v.
$CD_v \subseteq \{IDs\}$; The list of nodes that can choose v as their clusterhead.
$S_v \in \mathbb{N}$; Is the size of v's cluster.

Macros

The size of v's cluster:
$Size_v := |\{z \in N_v : Head_z = v\}|$;
The v's neighbors could be clusterheads of v:
$N_v^+ := \{z \in N_v : (v \in CD_z) \wedge (Ch_z = T) \ \wedge (w_z > w_{Head_v}) \wedge (w_z > w_v)\}$;
Computation of $CD2_v$:
Begin
$CD0_v := \{z \in N_v : w_{Head_z} < w_v \wedge w_z < w_v\}$;
If $|CD0_v| \leqslant SizeBound - Size_v$ **then** $CD1_v := CD0_v$;
Else $CD1_v$ contains the $SizeBound - Size_v$ smallest members of $CD0_v$;
If $CD_v \subseteq CD1_v \cup \{z \in N_v : Head_z = v\}$ **then** $CD2_v := CD1_v$;
Else $CD2_v := \emptyset$;
End

In our protocol, a node v has three possible states. It can be a clusterhead ($Ch_v = T$), nearly ordinary node ($Ch_v = NF$), or an ordinary node ($Ch_v = F$).
A node v that is clusterhead or nearly ordinary, is the leader of the cluster, and it is responsible to managing it.
A clusterhead having to resign its role, takes the nearly ordinary state; and it stays in this state till its cluster is not empty. On the other hand, an ordinary node v belonging to a cluster whose clusterhead has the nearly ordinary state ($Ch_{Head_v} = NF$), has to quit its cluster (to become clusterhead or to change the cluster). These conditions guarantee that during the construction/maintenance of clusters, no clusterhead abandons its leadership. Thus, the hierarchical structure of the network is continuously provides even during its reorganization.

Notation 1 *We note by $CD_v(c)$ the value of v's CD variable in a configuration c, and we note by $Cluster_v(c)$ the value of $Cluster_v$ in a configuration c.*
Let $P_s(v)$ be the following predicate, $P_s(v) \equiv |CD_v \cup Cluster_v| \leqslant SizeBound$.

Protocol 2 : Robust Self-Stabilizing Clustering Protocol on node v.

Predicates

$\mathbf{G_0}(v) \equiv (Head_v \notin N_v \cup \{v\}) \vee (w_v > w_{Head_v}) \vee (Ch_{Head_v} \neq T) \vee (S_{Head_v} > SizeBound)$

$\mathbf{G_1}(v) = \mathbf{G_{11}}(v) \vee \mathbf{G_{12}}(v)$
$\quad \mathbf{G_{11}}(v) \equiv (Ch_v = F) \wedge (N_v^+ = \emptyset) \wedge \mathbf{G_0}(v)$
$\quad \mathbf{G_{12}}(v) \equiv (Ch_v = NF) \wedge (N_v^+ = \emptyset)$
$\mathbf{G_2}(v) = \mathbf{G_{21}}(v) \vee \mathbf{G_{22}}(v)$
$\quad \mathbf{G_{21}}(v) \equiv (Ch_v = F) \wedge (N_v^+ \neq \emptyset)$
$\quad \mathbf{G_{22}}(v) \equiv (Ch_v = NF) \wedge (Size_v = 0) \wedge (N_v^+ \neq \emptyset)$
$\mathbf{G_3}(v) \equiv (Ch_v = T) \wedge (N_v^+ \neq \emptyset)$

$\mathbf{G_4}(v) \equiv (Ch_v = T) \wedge [(S_v \neq Size_v) \vee (Head_v \neq v) \ \vee (CD_v \neq CD2_v)]$
$\mathbf{G_5}(v) \equiv (Ch_v = NF) \wedge [(S_v \neq 0) \vee (Head_v \neq v) \ \vee (CD_v \neq \emptyset)]$
$\mathbf{G_6}(v) \equiv (Ch_v = F) \wedge [(S_v \neq 0) \vee (CD_v \neq \emptyset)]$

Rules

$\mathbf{R_1}(v) : \mathbf{G_1}(v) \longrightarrow Ch_v := T;\ Head_v := v;\ S_v := Size_v;\ CD_v := CD2_v;$
$\mathbf{R_2}(v) : \mathbf{G_2}(v) \longrightarrow Ch_v := F;\ Head_v := max_{w_z}\{z \in N_v^+\};\ CD_v := \emptyset;\ S_v := 0;$
$\mathbf{R_3}(v) : \mathbf{G_3}(v) \longrightarrow Ch_v := NF;\ Head_v := v;\ S_v := 0;\ CD_v := \emptyset;$

$\mathbf{R_4}(v) : \neg\mathbf{G_3}(v) \wedge \mathbf{G_4}(v) \longrightarrow Head_v := v;\ S_v := Size_v;\ CD_v := CD2_v;$
$\mathbf{R_5}(v) : \neg\mathbf{G_1}(v) \wedge \neg\mathbf{G_2}(v) \wedge \mathbf{G_5}(v) \longrightarrow Head_v := v;\ S_v := 0;\ CD_v := \emptyset;$
$\mathbf{R_6}(v) : \neg\mathbf{G_1}(v) \wedge \neg\mathbf{G_2}(v) \wedge \mathbf{G_6}(v) \longrightarrow S_v := 0; CD_v := \emptyset;$

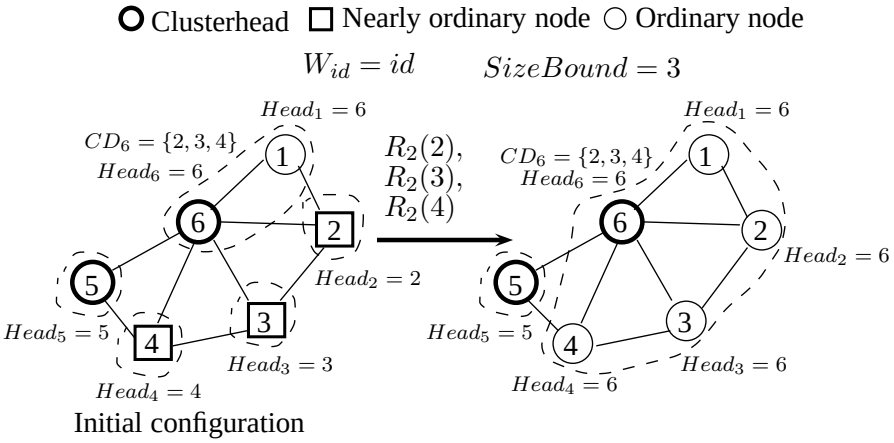

Figure 1: Violation of the size condition from a configuration not satisfying $P_s(v)$.

To prevent the violation of the size condition, a node u cannot freely join a cluster: u needs the permission of its potential new clusterhead. More precisely, only the nodes belonging to the set CD_v may join v's cluster. The goal of this mechanism is to preserve the size condition after any computation step. To achieve that goal, each clusterhead v will satisfy the predicate $P_s(v)$ (the P_s predicate is stronger than the size condition).
A cluster whose the clusterhead v satisfies the predicate $P_s(v)$, verifies the size condition in the current configuration and after any computation step. On the contrary, a cluster whose the clusterhead v not satisfying the predicate $P_s(v)$, may not verify the size condition after a specific computation step (in which all nodes of CD_v join v's cluster). This feature is illustrated in Figure 1. In the initial configuration, $Cluster_6 = \{1\}$, and $CD_6 = \{2,3,4\}$. Thus, the size condition is satisfied, but the $P_s(6)$ predicate is not satisfied:

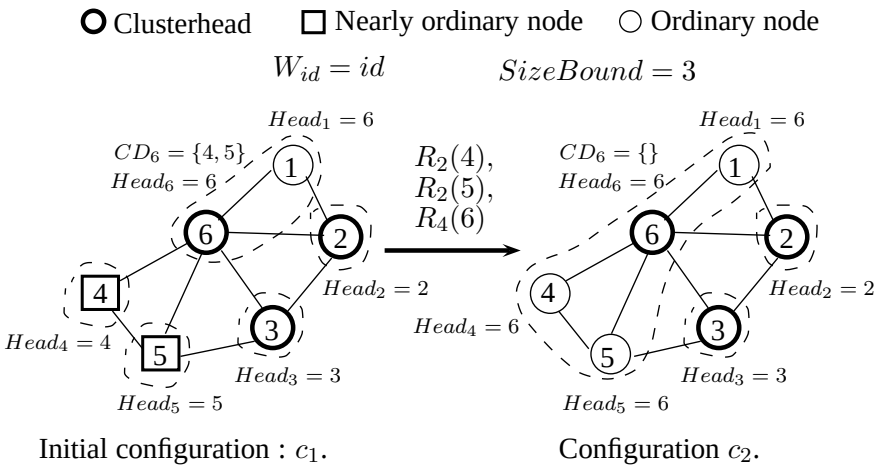

Figure 2: Illustration of CD value computation.

$|CD_6 \cup Cluster_6| = |\{1, 2, 3, 4\}| > 3$. After the computation step where all nodes of CD_6 join 6's cluster, the size condition is no more satisfied. Therefore, the variable CD_v is computed in such a way that the predicate $P_s(v)$ stays verified after any computation step.
For each clusterhead v, the macro $CD2_v$ is used to compute CD_v value. The value of $CD2_v$ is computed in 3 steps. $CD0_v$ is the set of v's neighbors that want to enter in the v's cluster, i.e., their weight and their clusterhead weight are smaller than v's weight. The size of $CD0_v$ can be greater than $SizeBound - Size_v$: $CD1_v$ is a subset of $CD0_v$, containing at most $SizeBound - Size_v$ elements. The set $CD2_v$ is a subset of $CD1_v$ ensuring that the predicate $P_s(v)$ stays verified by v after any computation step from the current configuration (assuming that $P_s(v)$ is verified in the current configuration).

The Figure 2 illustrates the computation of CD_v value. In the initial configuration, there are 5 clusters satisfying the size condition, and $CD_6 = \{4, 5\}$. For simplicity, let the weight of a node is equal to its identity. Thus, the clusterhead 6 has the highest weight in its neighborhood. The nodes $2, 3, 4$ and 5 want to belong to $Cluster_6$ (the node 1 is already in $Cluster_6$); so, $CD0_6 = N_6 - Cluster_6 = \{2, 3, 4, 5\}$.
The set $CD1_6$ contains only two nodes, because $SizeBound = 3$ and $|Cluster_6| = 1$; so, $CD1_6 = \{2, 3\}$.
In the reached configuration, c_2, $CD_6 = \emptyset$ because $CD_6(c_1) \nsubseteq \{CD1_6(c_1) \cup Cluster_6(c_1)\}$. Notice that in c_2, $P_s(6)$ is still verified: $|CD_6(c_2) \cup Cluster_6(c_2)| \leqslant SizeBound$.

The set N_v^+ is the neighbors of v that are better clusterhead than the current v's clusterhead. If the set N_v^+ is not empty, then v must change of clusterhead. For an ordinary node, the rule R_2 is enabled. A clusterhead v having $N_v^+ \neq \emptyset$ does not satisfy the neighboring clusterhead condition (it must join another cluster). In order to maintain the hierarchical structure over the network, the clusterhead v does not take directly the ordinary state; so, v becomes a nearly ordinary node (rule R_3). In this state, v performs correctly its task of clusterhead, but $Ch_v = NF$ and $CD_v = \emptyset$, i.e. no node can join the v's cluster.

An ordinary node v has to change of cluster if $G_0(v)$ is verified, i.e., its cluster does not verify the size condition ($S_{Head_v} > SizeBound$), or its clusterhead has a nearly ordinary state. The rule executed by v depends on N_v^+ value. If $N_v^+ = \emptyset$, then no node can be the clusterhead of v. So, v must become a clusterhead (rule R_1). If $N_v^+ \neq \emptyset$, then v has a neighbor that could be its new clusterhead. So, v changes of cluster (rule R_2).
The members of a cluster whose the leader has the nearly ordinary state, have to quit their cluster (they verify the predicate G_0). Thus, a nearly ordinary node v, will eventually be the

manager of an empty cluster ($Size_v = 0$). Then, either it joins an existing cluster if $N_v^+ \neq \emptyset$ (rule R_2), or it becomes a clusterhead (rule R_1).

Due to an incorrect initial configuration, a node v may have to correct the value of its local variables: $Head_v, CD_v,$ and S_v. In this case, it verifies one of the following predicates: $G_4(v), G_5(v),$ or $G_6(v)$.

3.3 Safety predicate

A configuration satisfying the safety predicate is safe. The safety predicate SP is defined as follow:
$SP \equiv \forall v,\ \mathcal{SP}_v = True$
$\mathcal{SP}_v \equiv (Head_v \in N_v \cup \{v\}) \wedge (Ch_{Head_v} \neq F)\ \wedge (|Cluster_v \cup CD_v| \leqslant SizeBound)$;

The safety predicate ensures that the following properties are satisfied:
- each node belongs to one cluster;
- each cluster has an effectual leader (no condition on a leader's weight, but its status is not ordinary node);
- the size of each cluster is less than $SizeBound$.

The safety predicate SP is preserved after the following input changes:
- the change of node's weight,
- the crash of an ordinary node,
- the failure of a link between (1) a clusterhead and a nearly ordinary node, (2) two clusterhead nodes, (3) two nearly ordinary nodes, or (4) two ordinary nodes,
- the joining of a subnetwork that verifies the predicate SP.

The convergence process from a safe configuration to a legitimate configuration is illustrated in Figure 3. The weight of a node is its identity. In the Configuration a, each node belongs to a cluster, and the size of each cluster is less than SizeBound: it is a safe configuration. The ordinary node 9 has the higher weight in the network, thus it has to be clusterhead. In initial configuration (Configuration **a**), node 9 is enabled because it is in ordinary state. During the first computation step, node 9 performs the rule R_1 to become clusterhead and it sets CD_9 to $\{6, 7\}$. In Configuration **b**, the nodes 6 and 7 must change of cluster, because $N_6^+ = N_7^+ = \{9\}$. So, node 6 and 7 resign to nearly ordinary state (Configuration **c**). As no node belongs to the cluster of 6 in configuration **c**, node 6 can perform the rule R_2 in order to integrate the cluster of 9. In the same configuration, the clusterhead of node 5 has the nearly ordinary state, thus node 5 is enabled. As node 5 has not a suitable clusterhead in its neigborhood ($N_5^+ = \emptyset$), it can perform the rule R_1. After a move of nodes 5 and 6, the configuration **d** is reached. In the configuration d, the cluster of 7 is empty, thus 7 can join the cluster of 9. In Configuration **e**, the construction of clusters is terminated. The last computation step allows to reach a terminal configuration where $CD_v = \emptyset$ for every node v.

The convergence proof is described in two steps. In the section 4, we prove that from any arbitrary configuration, a safe configuration is reached in four rounds; and the safety predicate holds continuously during the stabilization to a legitimate configuration (that is also safe).
A legitimate configuration is terminal and verifies the well-balanced clustering properties. In the section 5, we prove that from a safe configuration, the system reaches a legitimate configuration in a bounded number of rounds.
In section 6, we prove that any terminal configuration is legitimate.

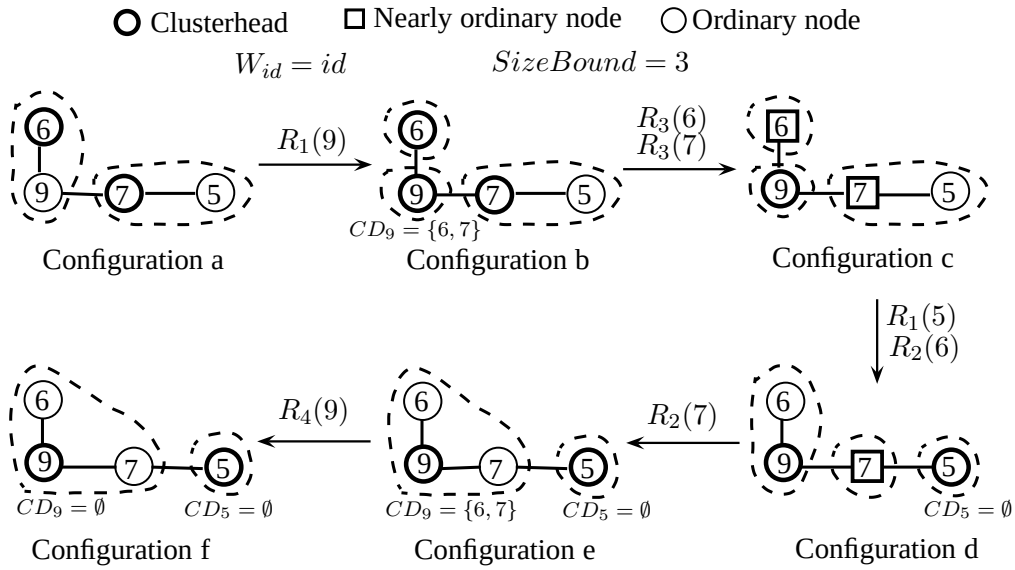

Figure 3: Illustration of convergence to a legitimate configuration.

4 Convergence to a safe configuration

In this section, we prove that A_i is an attractor from A_{i-1} for $0 \leqslant i \leqslant 3$ (Let A_{-1} is $\mathcal{C}$).

Notation 2
$P_t(v) \equiv CD_v = \emptyset$;

$A_0 = \{c \in C \mid \forall v \ : \ P_s(v) \ \vee P_t(v)$ *is satisfied* $\}$.
$A_1 = \{c \in C \mid \forall v \ : \ P_s(v)$ *is satisfied* $\}$.
$A_2 = A_1 \cap \{c \in \mathcal{C} | \ \forall v : |Cluster_v| \leqslant SizeBound\}$.
$A_3 = A_2 \cap \{c \in \mathcal{C} | \ \forall v : [(Ch_v \neq F) \wedge (S_v \leqslant SizeBound) \ \wedge (Head_v = v)] \vee$
$[(Ch_v = F) \wedge (Head_v \in N_v) \ \wedge (Ch_{Head_v} \neq F)]\}$.

Observation 1 *Let v be a node of V. Let cs be a computation step: $c_1 \xrightarrow{cs} c_2$. According to the macro N^+ and the rule R_2, we have:*

$$Cluster_v(c_2) \subseteq \{Cluster_v(c_1) \cup CD_v(c_1)\} \tag{1}$$

For any rule performed by v during cs, v updates its variable CD_v (see the rules action). According to the macro $CD2_v$, (see $CD2_v$ computation method):

$$CD_v(c_2) = \emptyset, \textbf{\textit{or}} \ CD_v(c_1) \subseteq \{CD_v(c_2) \cup Cluster_v(c_1)\} \tag{2}$$

$$CD1_v(c_1) \cap Cluster_v(c_1) = \emptyset \tag{3}$$

Lemma 1 *Let cs be a computation step: $c_1 \xrightarrow{cs} c_2$, in which a node v performs an action. If $CD_v(c_2) \neq \emptyset$ then $|CD_v(c_2) \cup Cluster_v(c_2)| \leqslant SizeBound$.*

Proof: The node v updates its CD_v variable during cs (see Observation 1). Assume that $CD_v(c_2) \neq \emptyset$. By definition of $CD1_v(c_1)$, it contains at most $SizeBound - |Cluster_v(c_1)|$ elements. Thus, $CD1_v(c_1) = \emptyset \ \vee \ |CD1_v(c_1) \cup Cluster_v(c_1)| \leqslant SizeBound$.
From $CD2_v$'s computation method, we have $(CD_v(c_2) = CD1_v(c_1)) \wedge (CD1_v(c_1) \neq \emptyset)$. Thus, $|CD_v(c_2) \ \cup \ Cluster_v(c_1)| \leqslant SizeBound$.

According to Equation (1), $|CD_v(c_2) \cup Cluster_v(c_2)| \leqslant |CD_v(c_2) \cup CD_v(c_1) \cup Cluster_v(c_1)|$.
From Equation (2), we obtain: $|CD_v(c_2) \cup CD_v(c_1) \cup Cluster_v(c_1)| \leqslant |CD_v(c_2) \cup Cluster_v(c_1)|$.
We conclude that $|CD_v(c_2) \cup Cluster_v(c_2)| \leqslant |CD_v(c_2) \cup Cluster_v(c_1)| \leqslant SizeBound$. □

Lemma 2 *A_1 is closed under any computation step.*

Proof: Let c_1 be a configuration in which $P_s(v)$ is satisfied. Assume that a computation step *cs* exists: $c_1 \xrightarrow{cs} c_2$, such that $P_s(v)$ is not satisfied in c_2. We will prove that *cs* does not exist. In c_1, $|CD_v(c_1) \cup Cluster_v(c_1)| \leqslant SizeBound$, and in c_2, $|CD_v(c_2) \cup Cluster_v(c_2)| > SizeBound$.
Thus, there exists a node z such that, $z \notin \{CD_v(c_1) \cup Cluster_v(c_1)\}$, but
$z \in \{CD_v(c_2) \cup Cluster_v(c_2)\}$.
According to Equation (1), v has changed the variable CD_v during *cs* to include the node z: $CD_v(c_2) \neq \emptyset$. According to Lemma 1, we have: $|CD_v(c_2) \cup Cluster_v(c_2)| \leqslant SizeBound$.
There is a contradiction; so, *cs* does not exist. □

Lemma 3 *A_0 is closed under any computation step.*

Proof: Assume that a computation step *cs* exists: $c_1 \xrightarrow{cs} c_2$, such that $P_s(v) \vee P_t(v)$ is satisfied in c_1. We will prove that $P_s(v) \vee P_t(v)$ stays satisfied in c_2.
If $P_s(v)$ is satisfied in c_1, then $P_s(v) \vee P_t(v)$ stays satisfied in c_2 (A_1 is closed, Lemma 2).
If $P_s(v)$ is not satisfied in c_1, then according to our assumption, $P_t(v)$ is satisfied in c_1: $CD_v(c_1) = \emptyset$. In the configuration c_2, there are two possibilities:

- **Case 1:** $CD_v(c_2) = \emptyset$, implies that $P_t(v)$ is verified in c_2.
- **Case 2:** $CD_v(c_2) \neq \emptyset$, according to Lemma 1, $P_s(v)$ is satisfied in c_2.

In both cases, $P_s(v) \vee P_t(v)$ is satisfied in c_2. □

Lemma 4 *If $P_s(v) \vee P_t(v)$ is not satisfied in a configuration c, then the node v is enabled in this configuration.*

Proof: Let c be a configuration in which $P_s(v) \vee P_t(v)$ is not satisfied.
In the configuration c, we have: $(P_s(v) = F) \wedge (P_t(v) = F)$.
According to the v's state in c, there are two cases.

- **Case 1:** the node v is an ordinary node or a nearly ordinary node. v is enabled in c, because $G_6(v)$ or $G_5(v)$ is satisfied in c ($CD_v(c) \neq \emptyset$).
- **Case 2:** the node v is a clusterhead. If $CD_v(c) \neq CD2_v(c)$, then v is enabled in c, because $G_4(v)$ is satisfied. We will prove that $CD_v(c) \neq CD2_v(c)$.
  According to $CD2_v$ computation method, we have: $CD2_v(c) = \emptyset$ or $CD2_v(c) = CD1_v(c)$.
  In the first case, $CD_v(c) \neq CD2_v(c)$, because $CD_v(c) \neq \emptyset$.
  In the second case, we will prove that $CD_v(c) \neq CD1_v(c)$.
  $CD1_v$ contains at most $SizeBound - |Cluster_v|$ elements of $CD0_v$. Two cases are possible:
  .$|Cluster_v(c)| > SizeBound$: we have, $CD1_v(c) = \emptyset$. Thus, $CD_v(c) \neq CD1_v(c)$.
  .$|Cluster_v(c)| \leqslant SizeBound$: we have, $|CD1_v(c)| \leqslant SizeBound - |Cluster_v(c)|$.
  As $|CD_v(c)| > SizeBound - |Cluster_v(c)|$ (by assumption), then $CD_v(c) \neq CD1_v(c)$. □

Lemma 5 *A_0 is an attractor from $\mathcal{C}$.*

Proof: Assume that a computation e not reaching a configuration c of A_0 exists. Thus, there exists a node v that never satisfies $P_s(v) \vee P_t(v)$ during the computation e. The node v is always enabled along the computation e (Lemma 4). By fairness, v will perform an action.
Let *cs* be a computation step: $c_1 \xrightarrow{cs} c_2$, in which v performs a rule. After the v's action, two cases are possible:

- **Case 1:** $CD_v(c_2) = \emptyset$, thus $P_t(v)$ is satisfied in c_2.
- **Case 2:** $CD_v(c_2) \neq \emptyset$. According to Lemma 1, $P_s(v)$ is satisfied in c_2.

Any computation of $\mathcal{E}$ reach a configuration of A_0 (convergence). As A_0 is closed under any computation step (Lemma 3), then A_0 is an attractor from $\mathcal{C}$. □

The following corollary is a consequence of Lemma 5.

Corollary 1 *A configuration of the set A_0 is reached from any initial configuration in at most one round.*

Lemma 6 *A_1 is an attractor from $\mathcal{C}$.*

Proof: A_0 is an attractor from $\mathcal{C}$ (Lemma 5) and A_1 is closed (Lemma 2). To prove that A_1 is an attractor from C, we need to prove that A_1 is reached from any configuration of A_0. Let c_1 be a configuration of A_0 but not of A_1.
In c_1, there exists a node v that satisfies $P_t(v) \wedge \neg P_s(v)$:

$$(CD_v(c_1) = \emptyset) \ \wedge \ (|Cluster_v(c_1)| > SizeBound).$$

Till $P_s(v)$ is not verified, $P_t(v)$ is verified (A_0 is closed, see Lemma 3). Thus, in the configuration c_1, no node can join the cluster of v ($CD_v(c_1) = \emptyset$).
Let u be a node of $Cluster_v$ ($Head_u = v$), then $v \notin N_u^+$ is forever satisfied (by definition of N_v^+). In the configuration c_1, the node u is enabled whatever its state:
- u **is clusterhead:** $G_4(u)$ is satisfied ($Head_u \neq u$); and $R_3(u)$ or $R_4(u)$ is enabled.
- u **is nearly ordinary node:** $G_5(u)$ is satisfied ($Head_u \neq u$); $R_1(u), R_2(u)$ or $R_5(u)$ is enabled.
- u **is ordinary node:** If $S_{Head_u} \leqslant SizeBound$, then $S_v \neq Size_v$, and v is enabled ($G_4(v)$ is verified). After v's action, we have $S_{Head_u} = Size_v > SizeBound$ ($G_0(u)$ is satisfied). Therefore, $G_{11}(u)$ or $G_{21}(u)$ is satisfied, and $R_1(u)$ or $R_2(u)$ is enabled.

The node u stays enabled up to the time where it performs an action. By fairness, u eventually performs a rule. After that, we get $Head_u \in N_u^+ \cup \{u\}$ (see rules action); thus $Head_u \neq v$. This means that u leaves $Cluster_v$. Eventually we reach a configuration in which $|Cluster_v| \leqslant SizeBound$; so, a configuration of A_1 is reached.
A_1 is closed under any computation step (Lemma 2). Thus, A_1 is an attractor from $\mathcal{C}$. □

The following corollary is a consequence of Lemma 6.

Corollary 2 *A configuration of the set A_1 is reached from A_0 in at most two rounds.*

Proof: In the first round, all clusterheads v having $S_v \leq SizeBound$ and $Size_v > SizeBound$ update their variable S_v to get $S_v = Size_v > SizeBound$. Thereafter, all nodes of $Cluster_v$ are enabled because they verify G_0. Thus, at the end of the second round, these nodes have done an action (to quit the v's cluster) or they are neutralized. The only action that neutralizes these nodes is the updating of the variable S_u to a value inferior to $SizeBound$; this action occurs when $P_s(u)$ is satisfied. □

Lemma 7 *$A_2 = A_1 \cap \{c \in \mathcal{C} |\ \forall v : |Cluster_v| \leqslant SizeBound\}$ is an attractor from $\mathcal{C}$.*

Proof: In A_1, each node v satisfies: $|Cluster_v| \leqslant |CD_v \cup Cluster_v| \leqslant SizeBound$.
So, $A_2 = A_1$. Therefore, A_2 is an attractor from $\mathcal{C}$ (Lemma 6). □

Lemma 8 *Let be the predicate $P_h \equiv \forall v : Ch_{Head_v} \neq F$.*
$A_2' = \{c \in C | P_h$ is satisfied $\}$ is closed under any computation step.

Proof: Let c_1 be a configuration in which P_h is satisfied. Assume that a computation step cs exists: $c_1 \xrightarrow{cs} c_2$, such that P_h is not satisfied in c_2. We prove that cs does not exist.
Let v be a node. Let u be the clusterhead of v in the configuration c_1. According to our assumption, $Ch_u \neq F$ in the configuration c_1.
During cs, there are two possibilities: either u has changed its status to become an ordinary node, or v has chosen a new clusterhead z, such that $Ch_z = F$ in c_2.
If the node u or v performs the rule R_1, R_3, R_4 or R_5, then $Ch_{Head_v} \neq F$ stays verified.

During cs, the nodes u and v have not performed the rule R_6; because this rule action does not change u's status, or v's clusterhead. Thus, during cs, one of the nodes u and v has performed R_2. There are two cases:
• **Case 1:** The node v has performed R_2.
Let z be the clusterhead chosen by v during c_s. In c_1, we have $Ch_z = T$ ($z \in N_v^+$), and in c_2, we have $Ch_z = F$. The rule R_2 is the only rule that changes the value of the variable Ch to F. During cs, the node z cannot perform R_2 ($G_2(z)$ is not verified in c_1). There is a contradiction.

• **Case 2:** The node u has performed R_2.
In c_1, we have $Ch_u \neq F$; and in c_2, we have $Ch_u = F$. The node u cannot perform the rule R_2 during cs, because $G_2(u)$ is not verified in c_1 ($Size_u \neq 0$). There is a contradiction.

After any computation step, P_h stays verified. □

Lemma 9 *$A_3 = A_2 \cap \{c \in \mathcal{C} | \forall v : [(Ch_v \neq F) \wedge (Head_v = v) \wedge (S_v \leqslant SizeBound)] \vee [(Ch_v = F) \wedge (Head_v \in N_v) \wedge (Ch_{head_v} \neq F)]\}$ is an attractor from A_2.*

Proof: Let v be a node. In a configuration of A_2, $Size_v \leqslant SizeBound$ (Lemma 7).
So, if $S_v > SizeBound$ then $S_v \neq Size_v$. In a configuration of A_2 but not of A_3, v has one of these states:
• $Ch_v = T$ **and,** $Head_v \neq v$ **or** $S_v > SizeBound$: $G_4(v)$ is satisfied. By fairness, v will perform R_3 or R_4. After v's action, we have: $(Ch_v \neq F) \wedge (Head_v = v) \wedge (S_v \leqslant SizeBound)$.
• $Ch_v = NF$ **and,** $Head_v \neq v$ **or** $S_v > SizeBound$: $G_5(v)$ is satisfied. By fairness, v will perform R_1, R_2 or R_5. After v's action, we have:

. After R_1 or R_5: $(Ch_v \neq F) \wedge (Head_v = v) \wedge (S_v \leqslant SizeBound)$ hold.

. After R_2: $(Ch_v = F) \wedge (Head_v \in N_v) \wedge (Ch_{Head_v} = T)$ hold.

• $Ch_v = F$ **and,** $Head_v \notin N_v$ **or** $Ch_{Head_v} = F$: we have $Ch_v = F$, then $Head_v \neq v$ or $Ch_{Head_v} = F$ ($Head_v = v$). So, $G_0(v)$ is satisfied; because $Head_v \notin N_v \cup \{v\}$ or $Ch_{Head_v} = F$. Therefore, $G_{11}(v)$ or $G_{21}(v)$ is satisfied. As all computations are fair, v will perform R_1 or R_2. After the v's action, v verifies:
$(Ch_v = T \wedge Head_v = v \wedge S_v \leqslant SizeBound) \vee (Ch_v = F \wedge Head_v \in N_v \wedge Ch_{Head_v} = T)$.

Thus, a configuration of A_3 is reached from a configuration of A_2.
The values of $Head_v$ are never falsified (see the rule actions). In A_2, the S_v value is inferior to $SizeBound$ after any updating (because $|Cluster_v| \leqslant SizeBound$). Thus, in A_2, once $S_v \leqslant SizeBound$ is verified, it stays verified forever. $Ch_{Head_v} \neq F$ is forever verified, because it is closed (Lemma 8). So, A_3 is closed under any computation step. □

The following corollary is a consequence of proofs of Lemma 7 and Lemma 9.

Corollary 3 *A configuration of the set A_3 is reached from a configuration of A_1 in at most one round. A_3 is an attractor from $\mathcal{C}$.*

Observation 2 *In a configuration of A_3, for each node v we have:*
$$Head_v \neq v \text{ if and only if } Ch_v = F.$$

4.1 Convergence time to a safe configuration

The Convergence time to a safe configuration is the maximum rounds needed to reach a safe configuration from any arbitrary initial configuration.
As the configurations of A_3 are safe. Thus, according to Corollaries 1, 2 and 3, the Convergence time to a safe configuration is at most four rounds.

5 Convergence to a legitimate configuration

Once a safe configuration is reached, the system progresses to reach a legitimate configuration. The convergence to a legitimate configuration is done in steps. At the end of the i^{eme} step, the configurations set $L_i^{''}$ is reached: All nodes of Set_i have chosen their clusterhead. We define the sets $L_i^{''}$ and Set_i as follows:

Notation 3 $Set_0 = \emptyset$; $L_0^{''} = A_3$;
$V_i = V - Set_i$; *V_i is the set of nodes that do not belong to Set_i.*
Let v_{hi} be the node having the highest weight in V_i;
$L_{i+1} = L_i^{''} \cap \{c \in \mathcal{C} |\ (Ch_{v_{hi}} = T)\ \}$;
$SizeBound_i = \textbf{\textit{Min}}(SizeBound,\ |N_{v_{hi}} \cap V_i|)$;
$L_{i+1}^{'} = L_{i+1} \cap \{c \in \mathcal{C} |\ |Cluster_{v_{hi}}| = SizeBound_i\ \}$;
$Set_{i+1} = Set_i \cup \{v_{hi}\} \cup Cluster_{v_{hi}}$;
$L_{i+1}^{''} = L_{i+1}^{'} \cap \{c \in \mathcal{C} |\ \forall\ v \in Set_{i+1} :\ CD_v = \emptyset\ \}$;

In Figure 3, all configurations belong to $L_0^{''}$, The set L_1 is reached after the first computation step. As L_1 is closed, the configurations from **b** to **f** belongs to L_1. The set $L_1^{'}$ is reached, after the fourth computation step: now, the cluster of 9 is stabilized. The configuration **f** is terminal and belongs to $L_1^{''}$. The configuration **f** is also legitimate because the well-balanced clustering properties are verified by the both clusters.

Observation 3 *In each step, the set Set_i increases up to contain all nodes.*

$$If\ (Set_i \neq V)\ then\ Set_i \subset Set_{i+1}\ \wedge\ Set_i \neq Set_{i+1} \tag{4}$$

$$If\ (v \in Set_i)\ then\ Head_v \in Set_i \tag{5}$$

$$If\ (v \in V_i)\ then\ Head_v \notin Set_i \tag{6}$$

For any value of i, each configuration of $L_i, L_i^{'}$, or $L_i^{''}$ belongs to A_3, because $L_i, L_i^{'}$, and $L_i^{''}$ are subsets of A_3.

We will prove that $L_j^{''}$, where $Set_j = V$, is an attractor from $\mathcal{C}$.

Lemma 10 *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: Assuming that $L_i^{''}$ is an attractor from $\mathcal{C}$, and by definition of v_{hi} we have:
$\forall u \in N_{v_{hi}}, w_{v_{hi}} > w_u\ (u \in V_i) \vee CD_u = \emptyset\ (u \in Set_i)$. So, $N^+_{v_{hi}}$ is empty forever.
In $L_i^{''}$, but not in L_{i+1} there are two possibilities :
• v_{hi} is a nearly ordinary node: $G_{12}(v_{hi})$ is forever satisfied. By fairness, v_{hi} eventually performs R_1. After that we get: $Ch_{v_{hi}} = T$.
• v_{hi} is an ordinary node: from Observations 2 and 3, we have $Head_{v_{hi}} \neq v_{hi}$. According to Equation 6 and to the definition of v_{hi}, we have: $w_{v_{hi}} > w_{Head_{v_{hi}}}$. So, $G_{11}(v_{hi})$ is forever satisfied. By fairness, v_{hi} eventually performs R_1. After that we get: $Ch_{v_{hi}} = T$.

In both cases L_{i+1} is reached from $L_i^{''}$. Furthermore, the value of $Ch_{v_{hi}}$ is never modified, because the rule $R_3(v_{hi})$ is never enabled ($N^+_{v_{hi}}$ is empty forever). Thus L_{i+1} is closed under any computation step. □

Corollary 4 *The configurations set L_{i+1} is reached from $L_i^{''}$ in at most one round.*

Lemma 11 *Let c_1 be a configuration of L_{i+1}. Assuming that L_{i+1} is an attractor from $\mathcal{C}$, for any computation step $c_1 \xrightarrow{cs} c_2$, we have: $Cluster_{v_{hi}}(c_1) \subseteq Cluster_{v_{hi}}(c_2)$.*

Proof: Let c_1 be a configuration of L_{i+1}. Let u be a node of $Cluster_{v_{hi}}$. In c_1, $Ch_u = F$ (Observations 2); and $\forall z \in N_u, w_{Head_u} > w_z$ $(z \in V_i) \vee CD_z = \emptyset$ $(z \in Set_i)$. Thus, $z \notin N_u^+$, i.e., N_u^+ is empty forever.
$G_2(u)$ is never verified, so the node u cannot change its clusterhead (the rule R_2 is disable).
$G_0(u)$ is never verified, because in L_{i+1} we have : $S_{Head_u} \leqslant SizeBound$ (Lemma 9), $v_{hi} \in N_u$ (Lemma 9), $w_{v_{hi}} > w_u$ (by definition of v_{hi}), and $Ch_{v_{hi}} = T$ (Lemma 10). Thus, $G_1(u)$ is never verified and the node u cannot become a clusterhead (the rule R_1 is disable).

According to that, once L_{i+1} is reached, no node can leave $Cluster_{v_{hi}}$. □

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: In a configuration c of L_{i+1}, we have: $Ch_{v_{hi}} = T$. Let u be a node of $\{N_{v_{hi}} \cap Set_i\}$. Thus, there exists j $(0 < j < i)$, such that: $(u = v_{hj}) \vee (u \in Cluster_{v_{hj}})$.
In L_{j+1}, the node u can never leave $Cluster_{v_{hj}}$ (see Lemma 11). Every configuration of L_{i+1} belongs to L_{j+1} $(j < i)$. So, once a configuration of L_{i+1} is reached, u can never join the $Cluster_{v_{hi}}$, i.e., $Cluster_{v_{hi}} \subseteq \{N_{v_{hi}} \cap V_i\}$.
In L_{i+1}, $|Cluster_{v_{hi}}| \leqslant SizeBound$ (Observation 3), thus $|Cluster_{v_{hi}}| \leqslant SizeBound_i$ is forever verified.

Once $|Cluster_{v_{hi}}| = SizeBound_i$, the set $Cluster_{v_{hi}}$ is never modified (see Lemma 11).
Assume that $|Cluster_{v_{hi}}| < SizeBound_i$ is verified. Thus, $CD1_{v_{hi}} \neq \emptyset$, because $CD1_{v_{hi}}$ contains $SizeBound_i - |Cluster_{v_{hi}}|$ elements of $(N_{v_{hi}} \cap V_i) - Cluster_{v_{hi}}$.

Till $CD_{v_{hi}} \neq CD2_{v_{hi}}$ the node v_{hi} is enabled ($G_4(v_{hi})$ is verified). By fairness, v_{hi} performs the rule R_4, and we get: $CD_{v_{hi}} = CD1_{v_{hi}}$or $CD_{v_{hi}} = \emptyset$ (now, $CD_{v_{hi}} \subseteq \{CD1_{v_{hi}} \cup Cluster_{v_{hi}}\}$).
In the last case, $G_4(v_{hi})$ still verified up to the time where v_{hi} performs again the rule R_4. After that we have: $CD_{v_{hi}} = CD1_{v_{hi}} \neq \emptyset$.

Let v be a node of $CD_{v_{hi}}$. We have $v_{hi} \in N_v^+$: $N_v^+ \neq \emptyset$. v is enabled whatever its state:
• v **is ordinary node**: $G_{21}(v)$ is always verified.

• v **is nearly ordinary node**: All nodes z of $Cluster_v$ satisfy $G_0(z)$ $(Ch_{Head_z} \neq T)$, thus $G_1(z)$ or $G_2(z)$ is enabled. After z's action, $Head_z \neq v$ holds. Thus, the size of $Cluster_v$ decreases. Eventually we reach a configuration in which $Size_v = 0$, and $G_{22}(v)$ is now satisfied.

• v **is clusterhead**: $G_3(v)$ is always verified. After performing $R_3(v)$, the node v becomes nearly ordinary node. Eventually it will reach a configuration in which $G_{22}(v)$ is satisfied (see the previous case).

By fairness v will perform the rule R_2, and it chooses the node v_{hi} as clusterhead, because v_{hi} is the node having the highest weight in N_v^+: $\forall z \in N_v$, if $w_z > w_{v_{hi}}$ then $z \in Set_i$ and $z \notin N_v^+$ $(CD_z = \emptyset)$. Eventually, when each node v of $CD_{v_{hi}}$ performs $R_2(v)$, a configuration where $|Cluster_{v_{hi}}| = SizeBound_i$ is reached. □

Corollary 5 *The configurations set $L_{i+1}^{'}$ is reached from L_{i+1} in at most five rounds.*

Proof: During the first and the second rounds, v_{hi} updates its variable $CD_{v_{hi}}$ to get $CD_{v_{hi}} = CD2_{v_{hi}}$. During the third round, every clusterhead $v \in CD_{v_{hi}}$ is enabled and will perform the rule $R_3(v)$. After that v becomes nearly ordinary node. At the end of this round, each node v of $CD_{v_{hi}}$ is either nearly ordinary node, or ordinary node.
During the fourth round, every node u member of $Cluster_v$ is enabled (v is a nearly ordinary node), because $G_0(u)$ is verified ($Ch_{Head_u} \neq T$). At the end of this round, the members u have quit their cluster, i.e., $Size_v = 0$.
During the last round, all nodes z of $CD_{v_{hi}}$ satisfy $G_2(z)$, and they perform the rule R_2. After that a configuration of L'_{i+1} is reached. □

Lemma 13 *In L'_{i+1}, for any node v of Set_{i+1}, $Ch_v \neq NF$ and $G_k(v) = F$, $\forall\ k = \{1, 2, 3, 5\}$.*

Proof: Let v be a node of Set_{i+1}. There exists j $(0 < j \leqslant i)$ such that:
$(v \in Set_{j+1}) \wedge (v \notin Set_j)$. So, $(v = v_{hj}) \vee (v \in Cluster_{v_{hj}})$.
In a configuration of L'_{j+1}, if $(v = v_{hj})$ then v is clusterhead (by definition of L'_{j+1} and Lemma 10); else (i.e. $v \in Cluster_{v_{hj}}$) v is ordinary node (see Observations 2, 3).
Thus, in L'_{j+1} the node v is either clusterhead or ordinary node.
• If v is a clusterhead $(v = v_{hj})$; from the definition of L'_{j+1} and Lemma 10, v can never execute the rule R_3.
• If v is an ordinary node $(v \in Cluster_{v_{hj}})$; according to the definition of L'_{j+1} and Lemma 11, v can never execute the rules R_1 and R_2.
According to that, in L'_{j+1} a node of Set_{i+1} $(j \leqslant i)$ can never perform the rules R_1, R_2, R_3, and R_5. $L'_{i+1} \subseteq L'_{j+1}$. □

Lemma 14 *In L'_{i+1}, $CD2_{vh_i} = \emptyset$.*

Proof: In L'_{i+1}, we have $|Cluster_{v_{hi}}| = SizeBound_i$ (Lemma 12).
If $|Cluster_{v_{hi}}| = SizeBound$, then $CD1_{v_{hi}} = \emptyset$.
If $|Cluster_{v_{hi}}| = |N_{v_{hi}} \cap V_i|$ $(< SizeBound)$, then a node u of $N_{v_{hi}}$ cannot join $Cluster_{v_{hi}}$, because $w_{Head_u} \geqslant w_{v_{hi}}$. So, $CD0_{v_{hi}} = \emptyset$.
Thus, in L'_{i+1}, $CD2_{v_{hi}} = \emptyset$. □

Lemma 15 *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} but not of Set_i. According to Lemma 13, $Ch_v \neq NF$; thus two cases are possible :
• v **is an ordinary node :** If $CD_v \neq \emptyset$, then v is enabled ($G_6(v)$ is verified). By fairness, the node v eventually performs the rule R_6. After that, $CD_v = \emptyset$ holds forever.
• v **is a clusterhead :** By definition, v is v_{hi}. In L'_{i+1}, we have $CD2_{v_{hi}} = \emptyset$ (Lemma 14). If $CD_{v_{hi}} \neq \emptyset$, then $G_4(v_{hi})$ is verified ($CD_{v_{hi}} \neq CD2_{v_{hi}}$). By fairness $R_4(v_{hi})$ will be performed. After that, $CD_{v_{hi}} = \emptyset$ holds forever. □

Corollary 6 *The configurations set L''_{i+1} is reached from L'_{i+1} in at most one round.*

Theorem 1 *The system eventually reaches a terminal configuration of L''_j, where $Set_j = V$, from any configuration.*

Proof: According to Observation 3, there exists j such that $Set_j = V$.
As $L''_0 = A_3$ is an attractor from $\mathcal{C}$ (Lemma 9); for any value of i, the sets L_i, L'_i, L''_i, are attractors from $\mathcal{C}$ (Observation 3, and Lemmas 10, 12, 15). Therefore, we conclude that L''_j is an attractor from $\mathcal{C}$.

In a configuration of L''_j, we have:

. A node v can perform only the rules R_4 and R_6 (Lemma 13).
. If v is a clusterhead then $Head_v = v$ (Observations 2, 3).
. $CD_v = \emptyset$ for any node v (By definition of $L_j^{''}$).
. If v is a clusterhead then $CD2_v = \emptyset$ (see lemma 14).

Thus, in a configuration of $L_j^{''}$, v can perform R_4 or R_6 only if $S_v \neq Size_v$ (v is clusterhead) or $S_v \neq 0$ (v is ordinary node).
In $L_j^{''}$, the value of $Size_v$ stays identical forever, because no node changes of cluster (Lemma 11). So, once $S_v = Size_v$, the rules $R_6(v)$ and $R_4(v)$ are disabled forever. A node v performs the rule R_4 or R_6 only once. After this action, v is disable forever; a terminal configuration is reached. □

Corollary 7 *The terminal configuration is reached from $L_j^{''}$ in at most one round.*

5.1 Stabilization time

The stabilization time is the maximum rounds needed to reach a legitimate configuration from any arbitrary initial configuration.
The set L_{i+1} is reached from $L_i^{''}$ in at most one round (Corollary 4). Five more rounds are needed to reach $L_{i+1}^{'}$ from L_{i+1} (Corollary 5). The set $L_{i+1}^{''}$ is reached from $L_{i+1}^{'}$ in at most one round (Corollary 6).

We note that when the v_{hi}'s cluster is empty, i.e., no ordinary node belongs to v_{hi}'s cluster, the sets $L_{i+1}^{'}$ and $L_{i+1}^{''}$ are reached once L_{i+1} is reached (no round is necessary), i.e., $L_{i+1}^{''}$ is reached from $L_i^{''}$ in at most one round. So, in the worst case :
(1) no cluster is empty: seven rounds are necessary to reach $L_{i+1}^{''}$ from $L_i^{''}$, for $0 < i \leqslant j$.
(2) each cluster contains exactly one ordinary node.

We conclude that, in the worst case, the configurations set $L_{i+1}^{''}$ is reached from $L_i^{''}$ after seven rounds, and $L_j^{''}$ is reached from $L_0^{''}$ after $7 * j$ rounds, where $j = \frac{|V|}{2}$.
The time to reach $L_0^{''}$ (A_3) is 4 rounds (see the Section 4.1), and a terminal configuration is reached from $L_j^{''}$ in at most one round (see Corollary 7). Therefore, along any computation, and from any initial configuration, the legitimate configuration is reached in at most $\frac{7*|V|}{2} + 5$ rounds.

6 Proof of correctness

Theorem 2 *In a terminal configuration, the well-balanced clustering properties are satisfied.*

Proof: Let j be an integer, such that $Set_j = V$. $L_j^{''}$ is an attractor from $\mathcal{C}$ (Theorem 1). According to Definition 1, any terminal configuration belongs to $L_j^{''}$. Thus, in a terminal configuration c, $CD_v = \emptyset$ for any node v; and v is not a nearly ordinary node ($L_j^{''} \subseteq L_j^{'}$ and Lemma 13).

• **v is an ordinary node:**
$G_{11}(v) = F$ implies that: $(Head_v \in N_v) \wedge (Ch_{Head_v} = T) \wedge (w_{Head_v} > w_v)$. So, the affiliation property is satisfied in the configuration c.

• **v is a clusterhead:**
$|Cluster_v| \leqslant SizeBound$ is an attractor from $\mathcal{C}$ (Lemma 7). According to Definition 1, in any terminal configuration c, the size property is satisfied.

Assume that there exists a node $z \in N_v$, such that:
$(Ch_z = T) \wedge (w_z > w_v) \wedge (Size_z < SizeBound)$. The node v belongs to $CD0_z$ because

$w_{Head_v} = w_v < w_z$, and $CD1_z \neq \emptyset$ because $Size_z < SizeBound$. Since, in the configuration c, $CD_z = \emptyset$; then according to the $CD2$ computation method, $CD2_z \neq \emptyset$ in the configuration c. Thus, $G_4(z)$ is satisfied ($CD_z \neq CD2_z$). There is a contradiction.
We conclude that, in c, for any clusterhead $z \in N_v$: $(w_v > w_z) \vee (Size_z = SizeBound)$. Thus, the neighboring clusterhead property is verified in the configuration c. □

7 Proof of robustness

Lemma 16 *The predicate $\mathcal{SP}_v$ is closed under any computation step.*

Proof: Once $Head_v \in N_v \cup \{v\}$ is verified, then the value of $Head_v$ is never falsified (see the rule actions). So, for any node v: $Head_v \in N_v \cup \{v\}$ is closed.
According to Lemma 8, $Ch_{Head_v} \neq F$ is closed.
$P_s(v) \equiv (|Cluster_v \cup CD_v| \leqslant SizeBound)$.
$A_1 = \{c \in C | \forall v : P_s(v)$ is satisfied$\}$ is closed (Lemma 2). We conclude that $\mathcal{SP}_v$ is closed under any computation step. □

Definition 4 *Let $\mathcal{IC}$ be the following set of input changes that can occur in the network:*
- *the change of node's weight,*
- *the crash of an ordinary node,*
- *the failure of a link between (1) a clusterhead and a nearly ordinary node, (2) two clusterheads, (3) two nearly ordinary nodes, or (4) two ordinary nodes,*
- *the joining of a subnetwork that verifies the predicate SP.*

Lemma 17 *$\mathcal{SP}_v$, is closed under the input changes of $\mathcal{IC}$.*

Proof: Let z be the clusterhead of the node v (z is a clusterhead or nearly ordinary node). The safety predicate $\mathcal{SP}_v$ ensure that the clusterhead z is a neighbor of the node v, z is not an ordinary node, and the size of z's cluster is less than $SizeBound$.
Once the safety predicate $\mathcal{SP}_v$ is verified and without occurrence of faults, it stays verified forever (see Lemma 16).
$\mathcal{SP}_v$ will be not verified only if one of the following events occurs: z's removal (or crash of the clusterhead z), or failure of the link between z and v.
Therefore, the safety predicate $\mathcal{SP}$ is preserved under any input changes of $\mathcal{IC}$. □

Corollary 8 *The protocol is robust under any input changes of $\mathcal{IC}$.*

Proof: The proof follows directly from Lemma 16 and Lemma 17. □

8 Conclusion

A robust self-stabilizing clustering protocol building bounded size weight-based clusters has been proposed.
Starting from an arbitrary configuration, in four rounds, the protocol reaches a safe configuration. Once a safe configuration is reached, each node belongs to a cluster having a leader, and each cluster contains at most $SizeBound$ members, but clusters may not satisfy the well-balanced clustering properties. During the construction of the final clusters, that satisfy the well-balanced clustering properties, the safety property is preserved.

Our protocol is designed for the state model. Nevertheless, it can be easily transformed into a protocol for the message-passing model.

Each node v broadcasts periodically to its neighbors a message containing its state. Based on this message, v's neighbors decide whether to update their states or not. After a change in the value its state, a node broadcasts to its neighbors its new state.

As future work, we plan to study how the proposed work could be used to conceive a robust and self-stabilizing hierarchical routing protocol.

References

[1] A. A. Abbasi and M. Younis. A survey on clustering algorithms for wireless sensor networks. *Computer Communications*, 30:2826–2841, 2007.

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

[3] 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. IEEE Computer Society, 2005.

[4] M. Chatterjee, S. K. Das, and D. Turgut. A weight-based distributed clustering algorithm for mobile ad hoc networks. In *Proceedings of the 7th International Conference on High Performance Computing (HiPC'00), LNCS 1970*, pages 511–524, 2000.

[5] M. Chatterjee, S. K. Das, and D. Turgut. Wca: A weighted clustering algorithm for mobile ad hoc networks. *Journal of Cluster Computing*, 5(2):193–204, 2002.

[6] E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11):643–644, 1974.

[7] S. Dolev, A. Israeli, and S. Moran. Uniform dynamic self-stabilizing leader election. *IEEE Transactions on Parallel and Distributed Systems*, 8:167–180, 1997.

[8] S. Dolev and N. Tzachar. Empire of colonies: Self-stabilizing and self-organizing distributed algorithm. *Theoretical Computer Science*, 410:514–532, 2009.

[9] 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.

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

[11] 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 *Proceedings of the 17th International Symposium on Parallel and Distributed Processing (IPDPS'03)*, page 162.2. IEEE Computer Society, 2003.

[12] A. Jain and A. Gupta. A distributed self-stabilizing algorithm for finding a connected dominating set in a graph. In *Proceedings of the 6th International Conference on Parallel and Distributed Computing Applications and Technologies (PDCAT'05)*, pages 615–619, 2005.

[13] C. Johnen and L. H. Nguyen. Self-stabilizing construction of bounded size clusters. In *the 2008 IEEE International Symposium on Parallel and Distributed Processing with applications (ISPA'08)*, pages 43–50, 2008.

[14] C. Johnen and L. H. Nguyen. Robust self-stabilizing weight-based clustering algorithm. *Theoretical Computer Science*, 410(6-7):581–594, 2009.

[15] 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.

[16] 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.

[17] S. Kamei and H. Kakugawa. A self-stabilizing approximation for the minimum connected dominating set with safe convergence. In *the 12th International Conference On Principles Of Distributed Systems (OPODIS'08), Springer LNCS 5401*, pages 496–511. Springer, 2008.

[18] L. Kleinrock and K. Faroukh. Hierarchical routing for large networks; performance evaluation and optimization. *Computer Networks*, 1:155–174, 1977.

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

[20] 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.

[21] 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.

[22] P. Sheu and C. Wang. Minimizing both the number of clusters and the variation of cluster sizes for mobile ad hoc networks. In *International Conference on Information Networking: Networking Technologies for Enhanced Internet Services (ICOIN'03), Springer LNCS 2662*, pages 682–691, 2003.

[23] O. Tomoyuki, I. Shinji, K. Yoshiaki, I. Kenji, and M. Kaori. An adaptive maintenance of hierarchical structure in ad hoc networks and its evaluation. In *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS'02)*, pages 7–13. IEEE Computer Society, 2002.

[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 *Proceedings of the 5th International Workshop on Distributed Computing (IWDC'03), Springer LNCS 2918*, pages 26–32, 2003.