Self-stabilizing with service guarantee construction of 1-hop weight-based bounded size clusters ☆

Colette Johnen[a], Fouzi Mekhaldi[a]

[a] *Université de Bordeaux, LaBRI, CNRS, UMR 5800. F-33405 Talence Cedex, France*

Abstract

This paper makes contributions in two areas. First, we introduce an extended approach of self-stabilization, called self-stabilization with service guarantee. A self-stabilizing system tolerates transient faults: it automatically recovers to a correct behavior after a stabilization period. However, during stabilization periods, no property on the system behavior is guaranteed. A self-stabilizing protocol with service guarantee quickly provides an useful minimal service, and it maintains the minimal service during stabilization despite the occurrence of some disruptions. To illustrate our approach, we propose a clustering protocol (called `SG-BSC`), that builds 1-hop, bounded size and weight based clusters. `SG-BSC` protocol is self-stabilizing with service guarantee: it quickly reaches, in 4 rounds, a safe configuration from any arbitrary one. In a safe configuration, the following useful minimal service is provided: "each node belongs to a cluster, each cluster has a leader and a bounded size. All nodes of a cluster are at most at distance 1 of the cluster-head". The convergence to a legitimate configuration (optimum service) is done in at most $\frac{7*N}{2} + 4$ rounds, where N is the number of nodes. We prove that any self-stabilizing protocol building weight-based clusters requires $O(N)$ rounds to stabilize. Once the optimum service is reached, any cluster-head has the highest weight in its cluster, and the number of clusters is locally optimal. During the stabilization period, the minimal service is preserved; so, the hierarchical organization stays available throughout the entire network. Simulation results show the interest of self-stabilization with service guarantee compared to self-stabilization.

Keywords: Self-stabilization, Service guarantee, Safety, Minimal service, Clustering, Bounded clusters

1. Introduction

One of the most wanted properties of distributed systems is the fault tolerance and adaptivity to topological changes, which consist of the system's ability to react to faults and disruptions in a well-defined manner. Self-stabilization [14] is one approach to design distributed systems having such properties. A self-stabilizing system, regardless of the initial configuration, converges without any external intervention in finite time (called stabilization period) to a legitimate configuration where the intended behavior is exhibited. Self-stabilization is thus attractive since it does not require correct initialization (any configuration can be initial), it ensures recovering from any transient failure and adaptation to topology changes in dynamic networks. It also provides foundation for self-properties as self-healing, self-organizing and self-adaptive.

Even though self-stabilization offers some advantages, this approach has a major drawback: during all stabilization periods, self-stabilizing protocols do not guarantee any property (except the eventual convergence) even if perturbations could be handled in a safe manner. Thus, the self-stabilization property is suited for distributed systems with intermittent disruptions, where the delay between successive disruptions is large enough so that the system can recover and then provide its service for a while. Whereas in large scale dynamic networks, like mobile ad-hoc networks, where disruptions and topology changes occur very often, the paradigm of self-stabilization is not adequate. Indeed, as the delay between successive disruptions is very small, the system may be continually disrupted causing a total loss of service. In order to overcome these drawbacks, we study the self-stabilization with service guarantee.

A self-stabilizing with service guarantee protocol $\mathcal{P}$ quickly provides a minimal service (called also a safety property). Once the minimal service is offered, $\mathcal{P}$ progresses to reach a legitimate configuration providing the optimum service while maintaining the minimal service in spite of any protocol action. Furthermore, the minimal service is preserved despite occurrences of some specific disruptions, called *Highly Tolerated Disruptions*, denoted $\mathcal{HTD}$. The service guarantee is thus provided by both: a fast recovering to an useful minimal service, and the preservation of minimal service despite occurrences of $\mathcal{HTD}$ disruptions.

Related approaches. The self-stabilization with service guarantee approach is related to the fault-containment [10], time-adaptive self-stabilization [27], snap-stabilization [6], best-effort [18], safe convergence [25] and super-stabilization [15] approaches. The common goal of these approaches is to provide a desired safety property during stabilization after one or several well defined events.

☆. This work was partially supported by the ANR projects ALADDIN and Displexity.

Email addresses: `johnen@labri.fr` (Colette Johnen), `mekhaldi@labri.fr` (Fouzi Mekhaldi)

URL: `http://labri.fr/~johnen` (Colette Johnen), `http://lri.fr/~mekhaldi` (Fouzi Mekhaldi)

The *fault-containment* confines the effect of a single fault to a certain constant-distance neighborhood from the faulty node. So, once a node v undergoes a fault, neighbors of v till a limited distance may be affected by this fault, but the remaining part of the system is not affected. The *1-strong self-stabilization* [21], a particular case of fault-containment, guarantees that a single memory corruption fault cannot be propagated, and that a single computation step leads the system to a legitimate configuration.

The *time-adaptive self-stabilization* [27] is an approach that ensures recovery from faults in time which is proportional to the number of faults. The studied problem in [27] is the bit persistent (a bit value must be equal at all nodes). A time-adaptive self-stabilizing algorithm is proposed for synchronous arbitrary networks where the number of faults $F \leqslant N/2$. The algorithm reaches a legitimate configuration in $O(F)$ rounds, but reaches a terminal configurations in $O(D)$ rounds (D is the network diameter) even for a small number of faults.

A protocol is *snap-stabilizing* if it always behaves according to its specification whatever its initial configuration. The safety property in snap-stabilization is *user-centric* [12] (not *system-centric* as in safe convergence, super-stabilization and self-stabilization with service guarantee approaches). This means that the answer to a properly initiated request by the protocol is correct. However, between the request and the answer, the system can behave arbitrarily, except of giving an erroneous answer. This approach is thus suited for service-oriented protocols, but not to silent protocols like clustering protocols.

A *best-effort* protocol maintains a predicate, denoted P_C, despite occurrences of topological changes defined by the topological predicate P_T. The best-effort property does not guarantee self-stabilization; for example, a best-effort protocol cannot start from an incorrect initial configuration. Moreover, P_T includes only topological changes, whereas $\mathcal{HTD}$ can contain topological changes and several other events (like memory corruption, loss or unordered reception of messages etc.). So a self-stabilizing with service guarantee protocol also provides the best-effort property.

The *safe convergence* property ensures that (1) the system quickly converges to a safe configuration, and (2) the safety property stays satisfied during the convergence to a legitimate configuration under any computation step. However, external disruptions are not handled. Let us study the self-stabilizing with service guarantee protocol [22] computing the knowledge of 1-hop neighbor clusters. The stabilization time of this protocol is 4 rounds as much as the time to reach a safe configuration. In this case, the safe convergence approach contributes nothing compared to self-stabilization (they become equivalents). The main specificity of [22] is the maintain of safety property in spite of disruptions made by the clustering protocol (i.e., reconstruction of clusters).

A *super-stabilizing* protocol guarantees that (1) starting from a legitimate configuration, a safety property is preserved after only one topology change (of a set $\mathcal{HTD}$), and (2) the safety property is maintained during recovering to a legitimate configuration assuming that no more topology change occurs during the convergence. Self-stabilization with service guarantee provides and maintains the safety property even before stabilization, unlike super-stabilization. For example, the super-stabilizing coloring algorithm [15] stabilizes in $O(N)$ rounds (N is the number of nodes), but from an illegitimate configuration it does not quickly converge to a safe configuration. Furthermore, a self-stabilizing with service guarantee protocol preserves the safety property in spite of several $\mathcal{HTD}$ disruptions that are simultaneous or not. Whereas, a super-stabilizing protocol handles only one disruption: if disruptions occur in bursts, a super-stabilizing protocol handles them as any self-stabilizing protocol.

Studied problem and Related works. We are interested to the 1-hop weight-based bounded size clustering in the context of dynamic networks. The clustering problem consists to gather mobile nodes into non-overlapping groups called clusters.
Each cluster has a single head, called leader or cluster-head, that acts as local coordinator of cluster, and eventually a set of ordinary nodes located within a closed proximity. In 1-hop clusters, ordinary nodes are neighbor (at distance 1) of their leader.

The stability and efficiency of the clustering structure is clearly related to the stability and efficiency of cluster-heads. A cluster-head should have a fair threshold of battery power, and reliable communication links. The weight of a node is an integer value representing node's capacity to ensure the cluster-head functions. In weight-based cluster, the heads are selected in regard of their weight. So, the stability of structure is improved compared to not weight-based approaches.
Building bounded size clusters ensures that the number of nodes inside each cluster is bounded by a threshold. The value of this threshold is decided by the cluster-head. Notice that this problem is an extension of the capacitated dominating set problem [26], where each cluster has a bounded size to the capacity of the cluster-head, but the selection of cluster-heads is not weight-based.

Numerous clustering protocols are proposed in the literature. A survey on clustering protocols for sensor networks can be found in [1]. In [19], `Highest-Connectivity` and `Lowest-ID` protocols building overlapping 1-hop clusters are proposed. `Highest-Connectivity` chooses the node having the highest degree (i.e., number of neighbors) in its neighborhood as leader, whereas, `Lowest-ID` selects as leader the node having the lowest identity. Protocol proposed in [28] organizes nodes into non-overlapping 1-hop clusters, where leaders are selected according to their identity. In [2], the protocol `DMAC` builds weight-based 1-hop clusters, where each leader has the biggest weight among nodes of its cluster, and two leaders cannot be neighbor (the dominating set is independent). `GDMAC` [3], a generalized version of `DMAC`, is proposed in order to reduce the resignation and affiliation overheads by (1) authorizing a leader to have k leaders in its neighborhood, and by (2) allowing an ordinary node v to be in the cluster of the leader does not having the largest weight in v's neighborhood. The selected leaders by `GDMAC` form a k-fold dominating set. Most of protocols surveyed in [1] and those proposed in [19, 28, 2, 3] address the 1-hop weight based clustering problem, but they are not self-stabilizing.

A survey on self-stabilizing protocols building dominating sets and its variants can be found in [20]. Two self-stabilizing clustering protocols are proposed in [4, 29]. In [4], cluster-heads are selected according to their identity in order to form 2-hops clusters: each node is in at most two hops away from its cluster-head. In [29], cluster-heads are selected according to their k-density: a criteria that depends on the k-neighborhood of the node, but the built clusters are 1-hop. In [17], two self-stabilizing protocols building connected dominating sets are proposed. The first protocol computes a maximal independent set, and after that it selects some nodes so that the set becomes connected. The second protocol is based on the following principle: "a node with two neighbors that do not have the same neighborhood, it must belong to the connected dominating set". The first protocol may select all nodes as dominating. The second one avoids such situation; however, it requires that each node knows its neighborhood at distance 2. In [24], a self-stabilizing with safe convergence version of DMAC under synchronous scheduler is presented. In [25] a self-stabilizing protocol with safe convergence building a minimum connected dominating set (with a certain approximation) is proposed. In a safe configuration, the built set is a dominating set (not minimal). This protocol requires a distinguished node. Self-stabilizing k-clustering protocols, where every node is at distance at most k from its cluster-head, are also proposed, for instance in [13, 8, 11].

Few protocols in the literature build bounded size clusters although it is useful to ensure load balancing over heads. In fact, if a certain network zone becomes densely populated with nodes, the head might not be able to handle all the traffic generated by nodes of its cluster. In addition, the power consumption of a cluster-head is proportional to the number of active nodes in its cluster. Thus, the lifetime of a head is inversely proportional to its cluster's size. As consequence, controlling the number of nodes in a cluster will extend its head's lifetime, and will improve the stability of the cluster. To our knowledge, the only protocols building bounded size clusters are [30, 9, 5, 23]. In [30], the obtained clusters have a size bounded by a lower and an upper bound. This solution cannot be applied to 1-hop clusters, because the degree of nodes may be less than the lower bound. Protocols [30, 9] are not self-stabilizing. The self-stabilizing protocol [5] also builds clusters having a size bounded by two thresholds, but they are multi-hops (no bound on the deep of clusters), and each cluster forms a spanning tree.

The self-stabilizing protocol BSC (Bounded Size Clustering) [23] builds 1-hop weight-based bounded size clusters. The clustering structure built by our protocol is inspired from the one produced by BSC, i.e., final clusters of both protocols satisfy the same properties (defined in Section 3.1). However, during the convergence phase, the behavior of two protocols is not the same: the refinement of clusters in order to get final ones is different. Although BSC is self-stabilizing, it does not have any service guarantee. In BSC, during construction of final clusters, i.e., stabilization period, a node may not belong to a cluster even if it was initially in a well formed cluster. This situation is avoided in our protocol.

Motivation and Contributions. The stabilization time of weight-based clustering protocols is proportional to the network size (Theorem 4). So, in large scale networks, the convergence of these protocols is very slow. Moreover, during the convergence of a self-stabilizing clustering protocol, a node may be outside any cluster even if it was initially in a cluster. Nevertheless, a crucial challenge in ad-hoc networks is the fast establishment and maintenance of clustering structure despite occurrences of topology changes. Thus, the fast achievement of a minimal service is necessary to avoid a long absence of service. Hence, the interest of self-stabilization with service guarantee.

We propose a self-stabilizing with service guarantee protocol building 1-hop bounded size weight-based clusters, called SG-BSC. SG-BSC protocol overcomes the limitation of BSC by dealing with the absence of service guarantee in the hierarchical structure produced during stabilization.

Starting from an arbitrary configuration, SG-BSC reaches a safe configuration in at most 4 rounds. In a safe configuration, the following useful minimal service is provided: "each node belongs to a cluster having an effectual leader, and the size of each cluster is bounded". Thereafter, SG-BSC protocol converges to a legitimate configuration in at most $\frac{7*N}{2} + 4$ rounds where N is the number of network nodes. During the convergence, the minimal service is preserved, i.e., the network stays partitioned into bounded size clusters. SG-BSC requires $O(log\ N)$ bits per node, where N is the number of nodes.

To analyze the interest/cost of service guarantee, simulation experiments are conducted to evaluate the performances of SG-BSC and BSC (the self-stabilizing version of SG-BSC) protocols. The comparison between protocols is based on the number of selected leaders, availability of both minimal and optimum services and the stabilization time. Obtained results show that the self-stabilization with service guarantee is more suitable than self-stabilization to large scale dynamic networks, since it allows SG-BSC to provide a highly available hierarchical structure (unlike BSC). However, to ensure this service guarantee, the stabilization time and the average number of leaders are slightly increased (they are higher in SG-BSC protocol than in BSC protocol).

The rest of the paper is organized as follows. In section 2, communication and computation models, self-stabilization and self-stabilization with service guarantee concepts are defined. Weight-based bounded size clustering problem and SG-BSC protocol are presented in section 3. In sections 4 and 5, proofs of service guarantee and stabilization are described, accompanied by their time complexity measures. Simulation results comparing the performance of SG-BSC with BSC are presented in section 6.

2. Model and Concepts

We model a distributed system S by an undirected graph $G = (V, E)$ where the vertex set V is the set of (mobile) nodes and the edge set E is the set of communication links. Every node v in the network is assigned a unique identifier. A link

$(u, v) \in E$ exists if and only if nodes u and v can directly communicate (links are bidirectional); so, u and v are neighbors. We note N_v the set of v's neighbors, i.e., $N_v = \{u \in V \mid (u, v) \in E\}$. The mobility of nodes is modeled by the creation/failure of links, and arrival/departure of nodes. Due to mobility, the neighborhood N_v of some nodes v may change. 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.

We consider a weighted network, i.e., a weight w_v (a real number) is assigned to each node v. The weight value of a node can increase or decrease during time reflecting changes in the state of nodes. For the sake of simplicity, we assume that nodes weight are different; the tie in node's weight could be broken using nodes identifier *id*.

Communication model. We use the *local shared memory model* introduced in [14]. Each node maintains a set of local variables. A node can read its own variables and those of its neighbors, but it can modify only its own ones. The *state* of a node is defined by the values of its local variables. The union of states of all nodes determines the *configuration* of the system. The *program* of each node v is a set of *rules*; each one has the form: $Rule_i :< Guard_i > \rightarrow < Action_i >$. The *guard* of a v's rule is a Boolean expression involving v's state and the state of v's neighbors. The *action* of a v's rule updates only v's state. 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.

Computation model. Nodes are not synchronized; nevertheless several nodes may perform their actions at the same time. During a *computation step* $c_i \rightarrow c_{i+1}$, one or several enabled nodes perform an enabled action and the system reaches the configuration c_{i+1} from the configuration c_i. A *computation* e 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 weakly *fair*, if for any node v that is always enabled along this computation, it eventually performs an action. In this paper, we study only weakly fair computations.

We note by C the set of all possible configurations, and by $\mathcal{E}$ the set of all weakly fair computations of the system. The set of weakly fair computations starting from a particular configuration $c \in C$ is denoted $\mathcal{E}_c$. $\mathcal{E}_A$ denotes the set of weakly fair computations where the initial configuration belongs to the set of configurations $A \subset C$.

A node v is *neutralized* during the computation step $c_i \xrightarrow{cs} c_{i+1}$, if v is enabled in c_i and not enabled in $c_{i=1}$, but it did not execute any action during cs. The neutralization of a node v happens when at least one v's neighbor changes its state during cs, and after this change, the guard of all v's rules become unsatisfied in c_{i+1}.

2.1. Self-Stabilization

A distributed system is self-stabilizing if and only if, it converges to a legitimate configuration regardless of its initial one, and it remains in a legitimate configuration till no disturbing event occurs. A configuration is legitimate if it matches the specification of the problem. To formally define self-stabilization, we use the *attractor* notion. A set of configurations B is an *attractor* if and only if: (1) starting from any arbitrary configuration, the system reaches a configuration of B in a finite time and, (2) starting from a configuration of B, each computation step maintains the system in a configuration of B.

Definition 1 (Attractor). *Let B_1, B_2 be subsets of C. B_2 is an attractor from B_1, if and only if the following conditions hold:*
• ***Convergence:*** $\forall c \in B_1$, *If* $(\mathcal{E}_c = \emptyset)$ *then* $c \in B_2$
$\forall e \in \mathcal{E}_{B_1}(e = c_1, c_2, ...), \exists i \geqslant 1, c_i \in B_2$
• ***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 $\mathcal{L} \subseteq C$, called set of legitimate configurations, such that:*
• *$\mathcal{L}$ is an attractor from C.*
• *Each configuration of $\mathcal{L}$ matches the specification of problem.*

To measure the time complexity, we use the *round* notion [16]. The first round of a computation $e = c_1, ..., c_j, ...$ is the minimal prefix $e_1 = c_1, ..., c_j$ of e, such that every enabled node v in c_1 either executes a rule or it is neutralized during a computation step of e_1. Let e_2 be the suffix of e such that $e = e_1 e_2$. The second round of e is the first round of e_2, and so on.

Stabilization time. Stabilization time of a protocol $\mathcal{P}$ is the maximal number of rounds for all computations of $\mathcal{P}$ reaching a legitimate configuration from any initial one.

2.2. Self-stabilization with service guarantee

A protocol $\mathcal{P}$ is self-stabilizing with service guarantee if and only if:

1. $\mathcal{P}$ is self-stabilizing;
2. From any arbitrary configuration, $\mathcal{P}$ quickly reaches a safe configuration, where a minimal service (so a safety property) is provided;
3. The minimal service (safety property) is preserved during progress of $\mathcal{P}$ towards the optimum service, i.e., during stabilization, under any action of $\mathcal{P}$;
4. The minimal service (safety property) is maintained despite occurrences of some disruptions, called *Highly Tolerated Disruptions*, denoted $\mathcal{HTD}$.

In general case, the set of $\mathcal{HTD}$ disruptions may include events like: memory corruption, loss or unordered reception of messages, link creation/failure, node departure/arrival and nodes crashes. In spite of occurrences of $\mathcal{HTD}$ disruptions, the useful minimal service is preserved. However, the occurrence of other disruptions is handled by self-stabilization mechanism, i.e., after their occurrences, the system may behave arbitrarily, but it will quickly provide its minimal service. The service guarantee in our approach is thus provided through: the fast recovering to an useful minimal service, and its preservation during progress towards the optimum service despite occurrences of $\mathcal{HTD}$ disruptions.

Definition 3 (Stabilization with service guarantee). *A self-stabilizing system S has service guarantee despite occurrences of $\mathcal{HTD}$ disruptions if and only if the set of configurations satisfying the safety predicate $\mathcal{SP}$ (that stipulates the safety property) of S is:*

- *An attractor from C.*
- *Closed under any disruption of $\mathcal{HTD}$.*

$\mathcal{SP}$ and $\mathcal{HTD}$ fulfilled by our protocol are defined respectively in Definitions 6 and 7.

3. The proposed protocol SG-BSC

3.1. Specification of weight-based bounded size clustering

The studied problem consists to build non-overlapping 1-hop clusters having bounded size (called capacitated dominating set problem) where the selection criteria of leaders is weight-based. In the capacitated dominating set problem (without weight-based selection criteria), any node may be leader. However, in studied problem, leaders are selected according to their weight value: this will ensure a best choice of suitable leaders.

Note that both problems of finding the minimum number of 1-hop clusters (i.e., a minimum dominating set), and the minimum number of bounded clusters (i.e., a minimum capacitated dominating set) are NP-hard [26]. For the studied problem, our goal is not to give a distributed solution finding the minimum number of clusters, but to propose a self-stabilizing with service guarantee construction of 1-hop weight-based bounded size clusters, satisfying some desired properties like: a best choice of leaders, a limited number of ordinary nodes per cluster, and a limited number of neighbor leaders.

For the weight-based clustering, we consider that each node v has an input value, its weight named $\mathtt{w}_v$, representing its capacity to be leader. The higher the weight of a node, the more adequate this node is for the leader task. A significant node's weight can be obtained by a sum of different normalized parameters like: node mobility, memory and processing capacity, bandwidth, battery power, and so on. The computation of the weight value is out the scope of this paper. Nevertheless, we consider that the weight value of a node can increase or decrease during time, reflecting changes in the state of the node.

In order to build bounded size clusters, each node v is assigned a constant integer $\mathtt{SizeBound}_v$ that indicates the maximum number of ordinary nodes that can belong to v's cluster if v is a leader. This upper-bound is imposed by the leader of cluster, and it may be different from a leader to another one, but it still constant for every one (its value does not change during time). This limitation on the number of nodes that a leader handles, ensures the load balancing: no leader is overloaded.

Final clusters provided by our protocol SG-BSC satisfy the *well-balanced clustering properties*, defined below.

Definition 4 (well-balanced clustering). *The well-balanced clustering consists to partition V into clusters $C_1, C_2 ... C_n$ such that the following properties, called well-balanced clustering properties, are verified:*

- **1-hop clustering**: *Each cluster C_i is headed by a node named $Head_i$ and the ordinary nodes of the cluster C_i are at distance 1 from C_i's head, i.e., $\bigcap_{i=1...n} C_i = \emptyset$, $\bigcup_{i=1...n} C_i = V$, $\forall 1 \leqslant i \leqslant n, Head_i \in C_i$, and $C_i \subseteq N_{Head_i} \cup \{Head_i\}$*
- **Affiliation property**: *The head of v'scluster has a weight greater than v's weight, i.e., $\forall 1 \leqslant i \leqslant n$, $w_{Head_i} = Max\{w_v | \forall v \in C_i\}$.*
- **Size property**: *The cluster C headed by u contains at most $SizeBound_u$ ordinary nodes, i.e., $\forall 1 \leqslant i \leqslant n$, $|C_i| \leqslant SizeBound_{Head_i} + 1$.*
- **Cluster-heads neighbor property**: *If $head_j$ is in the neighborhood of $head_i$, and if the weight of $head_j$ is smaller than the weight of $head_i$ then $head_j$ cannot belonging to the C_i cluster because the cluster C_i is full, i.e. $\forall i, j \in [1, n], Head_i \notin N_{Head_j} \vee w_{Head_i} \leq w_{Head_j} \vee |C_i| = SizeBound_{Head_i} + 1$.*

Definition 5 (Legitimate configuration). *A configuration of SG-BSC protocol is legitimate if and only if* well-balanced clustering properties *are satisfied.*

The goal of *Affiliation property* is to ensure that each cluster-head is more suitable than ordinary nodes inside its cluster, and the goal of *Size property* is to provide bounded size clusters. Since clusters have bounded size, several cluster-heads may be neighbors. The *cluster-heads neighbor property* limits locally the number of cluster-heads by requiring a node v to stay cluster-head only if it cannot join any neighbor cluster: the weight of u, a cluster-head neighbor of v, is smaller than v's weight or u's cluster contains $SizeBound_u + 1$ nodes.

Notice that *cluster-head neighbor* property ensures that the set of cluster-heads S selected by SG-BSC protocol is minimal: there is not clustering architecture satisfying the *Affiliation* and *Size* properties where the cluster-heads set S' is a proper subset of S (i.e., $S' \subset S$). Notice also that when at least a node v has $\mathtt{SizeBound}_v > 0$, the trivial configuration in which each node of the network is a cluster-head, does not satisfy the specification of the problem. Because, this configuration does not satisfy the *cluster-heads neighbor property*: some cluster-heads can become ordinary (by affiliating with v) without violating the *Affiliation* and *Size* properties.

3.2. Overall presentation of SG-BSC protocol

To meet the well-balanced clustering properties, each node v maintains a set of local variables (cf Protocol 1); for instance, its hierarchical status (HS_v), the identity of its head ($Head_v$), and the weight of its head (wh_v).

An important feature of SG-BSC protocol is the service guarantee; so, during construction of well-balanced clusters, leaders must avoid generating orphan nodes (ordinary nodes without leaders) when resigning their leadership. This is why, SG-BSC protocol use three possible hierarchical status per node v: v can be cluster-head ($HS_v = CH$), ordinary ($HS_v = O$) or nearly ordinary ($HS_v = NO$). The status transition diagram is shown in Figure 1, where transitions are the Election, Resignation, Affiliation and Correction actions defined in Protocol 2.

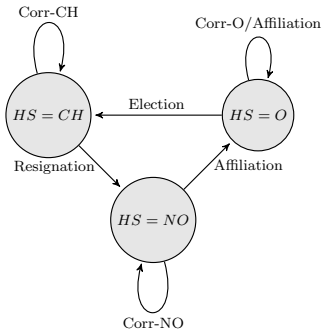


Figure 1: Status transition diagram

The nearly ordinary status is taken by a cluster-head wanting to resign its leadership. A nearly ordinary node is the head of its cluster; so, it behaves like a cluster-head but it is waiting to become ordinary. No node joins a cluster whose the head has the nearly ordinary status. Moreover, a nearly ordinary may become ordinary only once its cluster is empty.

Since, clusters have bounded size, a node u cannot freely join a cluster: u needs the permission of its future cluster-head to prevent the violation of the *Size property*. So, each leader sermine the neighbor nodes that are allowed to join its cluster. For this reason, each node v maintains CD_v, the list of nodes which may join v's cluster. For a node v that is not cluster-head, $CD_v = \emptyset$, and for a cluster-head v, CD_v is set to $CD2_v$ (for more details about the macro $CD2_v$, see Section 3.3).

Protocol 1 : Variables and macros on node v.

Parameters

$SizeBound_v \in \mathbb{N}$; /* A constant that indicates the maximum number of ordinary nodes can be in v's cluster */

Input variables

$w_v \in \mathbb{R}$; /* Weight of node v */

Local variables

$HS_v \in \{CH, O, NO\}$; /* Hierarchical status of node v */
$Head_v \in \{IDs\}$; /* Identity of v's cluster-head */
$wh_v \in \mathbb{R}$; /* Weight of the head of v */
$CD_v \subseteq \{IDs\}$; /* List of nodes that can join v's cluster */
$S_v \in \mathbb{N}$; /* Local value about the size of v's cluster */

Macros

$Cluster_v = \{z \in N_v : Head_z = v\}$; /* The v's cluster members (i.e., set of nodes having chosen v as cluster-head) */
$Size_v = |Cluster_v|$; /* Size of v's cluster */
$N_v^+ = \{z \in N_v, v \in CD_z \wedge HS_z = CH \wedge w_z > w_{Head_v} \wedge w_z > w_v\}$;
/* The v's neighbors could be cluster-head of v */
$BestHead_v = z \in N_v^+$ where $w_z = Max\{w_u \mid \forall u \in N_v^+\}$;
/* The most suitable leader of N_v^+ */;
$CD2_v$ /* Nodes authorized by v to join v's cluster */

Begin

$CD0_v := \{z \in N_v : wh_z < w_v \wedge w_z < w_v\}$;
If $|CD0_v| \leqslant SizeBound_v - Size_v$ **then** $CD1_v := CD0_v$;
Else $CD1_v$ contains the $SizeBound_v - Size_v$ smallest members of $CD0_v$;
If $CD_v \subseteq \{CD1_v \cup Cluster_v\}$ **then** $CD2_v := CD1_v$;
Else $CD2_v := \emptyset$;

End

The set N_v^+ contains v's neighbors (u) that could be v's future head: they have cluster-head status, they authorize v to join their cluster (i.e., $v \in CD_u$), and they are suitable (i.e., they have a bigger weight than both v and v's current head). For a cluster-head v, $N_v^+ \neq \emptyset$ means that v does not satisfy the cluster-heads neighbor property. Whereas, for an ordinary node v, it means that v does not satisfy the affiliation property. In both cases, v must leave its cluster to affiliate with the most suitable cluster-head among N_v^+ (i.e., node of N_v^+ having the highest weight).

`SG-BSC` protocol consists of four kind of rules (cf Protocol 2). Election rule allows a node to become cluster-head. Resignation rule allows a cluster-head to become nearly ordinary. Affiliation rule allows an ordinary or a nearly ordinary node to leave its cluster and to join another one as ordinary. Correction rules update if necessary the value of v's local variables without changing v's status.

Protocol 2 : `SG-BSC` Clustering Protocol.

Predicates

/* `Change(v)` is verified if a node has to change its cluster-head */
$\texttt{Change(v)} \equiv (Head_v \notin N_v \cup \{v\}) \vee (w_v > w_{Head_v}) \vee (HS_{Head_v} \neq CH) \vee (S_{Head_v} > SizeBound_{Head_v})$

/* The guard of Election rule */
$\texttt{MustBecomeHead}(v) \equiv (HS_v = O \wedge N_v^+ = \emptyset \wedge \texttt{Change(v)}) \vee (HS_v = NO \wedge N_v^+ = \emptyset)$

/* The guard of Affiliation rule */
$\texttt{MustAffiliate}(v) \equiv (HS_v = O \wedge N_v^+ \neq \emptyset) \vee (HS_v = NO \wedge Size_v = 0 \wedge N_v^+ \neq \emptyset)$

/* The guard of Resignation rule */
$\texttt{MustResign}(v) \equiv (HS_v = CH) \wedge (N_v^+ \neq \emptyset)$

/* The guards of Correction rules */
$\texttt{Corr-CH-g(v)} \equiv (HS_v = CH) \wedge (Head_v \neq v \vee CD_v \neq CD2_v \vee S_v \neq Size_v \vee wh_v \neq w_v)$
$\texttt{Corr-NO-g(v)} \equiv (HS_v = NO) \wedge (Head_v \neq v \vee CD_v \neq \emptyset \vee S_v \neq 0 \vee wh_v \neq w_v)$
$\texttt{Corr-O-g(v)} \equiv (HS_v = O) \wedge (CD_v \neq \emptyset \vee S_v \neq 0 \vee wh_v \neq w_{Head_v})$

Rules

/* Clustering Construction rules */
Election : $\texttt{MustBecomeHead}(v) \longrightarrow HS_v := CH;\ Head_v := v;\ CD_v := CD2_v;\ S_v := Size_v;\ wh_v := w_v;$

Affiliation : $\texttt{MustAffiliate}(v) \rightarrow HS_v := O; Head_v := BestHead_v;\ CD_v := \emptyset; S_v := 0;\ wh_v := w_{Head_v};$

Resignation : $\texttt{MustResign}(v) \longrightarrow HS_v := NO;\ Head_v := v;\ CD_v := \emptyset;\ S_v := 0;\ wh_v := w_v;$

/* Correction rules */
Corr-CH : $\neg\texttt{MustResign}(v) \wedge \texttt{Corr-CH-g(v)} \longrightarrow Head_v := v;\ CD_v := CD2_v;\ S_v := Size_v;\ wh_v := w_v;$

Corr-NO : $\neg\texttt{MustBecomeHead}(v) \wedge \neg\texttt{MustAffiliate}(v) \wedge \texttt{Corr-NO-g(v)} \longrightarrow Head_v := v;\ CD_v := \emptyset;\ S_v := 0;\ wh_v := w_v;$

Corr-O : $\neg\texttt{MustBecomeHead}(v) \wedge \neg\texttt{MustAffiliate}(v) \wedge \texttt{Corr-O-g(v)} \longrightarrow CD_v := \emptyset;\ S_v := 0;\ wh_v := w_{Head_v};$

Election and Affiliation of ordinary nodes. An ordinary node v has to change its cluster, i.e., it has to join another cluster or to become cluster-head, if it does not satisfy the *Affiliation* or *Size* properties, i.e., when it satisfies the predicate `Change`(v). In this case, one of the guards `MustBecomeHead`(v) and `MustAffiliate`(v) is satisfied. The rule executed by v depends on N_v^+ value. If $N_v^+ = \emptyset$, then no v's neighbor can be the

cluster-head of v. So, v must become cluster-head (`Election` rule). Otherwise (i.e., $N_v^+ \neq \emptyset$), v has a neighbor that could be its new cluster-head. So, v affiliates to the node of N_v^+ having the largest weight (`Affiliation` rule).

Resignation. A cluster-head v has to resign its leadership when it does not satisfy the *Cluster-heads neighbor property*, i.e. $N_v^+ \neq \emptyset$. In this case, v verifies the guard `MustResign`(v). However, if a cluster-head v by resigning its leadership takes the ordinary status, then ordinary nodes of its cluster become *orphan* (their head is now ordinary), and they are outside all clusters. In order to maintain the hierarchical structure, a cluster-head executing `Resignation` rule does not take directly the ordinary status: it takes the nearly ordinary status where it still behaves as a cluster-head.

Election and Affiliation of nearly ordinary nodes. After that a cluster-head v has performed `Resignation` rule and became nearly ordinary, we have $CD_v = \emptyset$. Thus, no node can longer join v's cluster; because $\forall u \in N_v,\ v \notin N_u^+$. Furthermore, all members of v's cluster verify the predicate `Change`; so, they will leave v's cluster. Therefore, v's cluster will eventually be empty ($Size_v = 0$). When $Size_v = 0$ and $N_v^+ \neq \emptyset$, v can become ordinary by performing `Affiliation` rule.

The mechanism employed by our protocol guarantees that during reconstruction of clusters caused by topology changes or the change of node's weight, no cluster-head abandons its leadership, and generates orphan nodes. Therefore, the hierarchical structure of the network is continuously available even during reorganization of clusters.

Correction. Due to an incorrect initialization, memory corruption or topology changes, a node v may need to change the value of its local variables (S_v, CD_v, wh_v or $Head_v$) without changing its status. This is the role of correction rules `Corr-CH`(v), `Corr-NO`(v) and `Corr-O`(v). These rules have less priority than Election, Resignation and Affiliation rules, i.e., if v has one of these last rules enabled, all correction rules are disabled on v.

Observation 1 (Memory space complexity). *Each cluster-head v in `SG-BSC` protocol (so, every node) maintains a list of nodes authorized to join v's cluster. Thus, `SG-BSC` protocol requires $O(\delta \log N)$ bits per node, where δ is the maximal degree of nodes. However, this bound can be reduced to $O(\log N)$ bits per node. In this case, each cluster-head v maintains only the identity of node having the biggest identifier among nodes of CD_v. All nodes having an identifier bigger than this identifier are not allowed to join v's cluster.*

3.3. Minimal service of `SG-BSC` protocol

During construction of clusters satisfying the *well balanced clustering* properties, our protocol provides the following useful minimal service (safety property):

- Each node v belongs to one cluster having an effectual leader (no condition on leader's weight, but it is v's neighbor and its hierarchical status is not ordinary);
- The Size property of clusters is satisfied.

In other words, once a safe configuration is reached, the entire network is partitioned into bounded size clusters.

Definition 6 (Safety predicate). *The safety predicate $\mathcal{SP}$ is defined as follow:*
- $P_s(v) \equiv |Cluster_v \cup CD_v| \leqslant SizeBound_v$
- $\mathcal{SP}(v) \equiv (Head_v \in N_v \cup \{v\}) \ \wedge (HS_{Head_v} \neq O) \ \wedge P_s(v)$
- $\mathcal{SP} \equiv \forall v,\ \mathcal{SP}(v) = True$

Preservation of the Size property. The affiliation and resignation mechanisms used by our protocol ensure that once a node is in a cluster, it always belong to a cluster whatever reorganization of clusters that can happen. However, to maintain the minimal service, *how the Size property is preserved after any computation step ?*

A cluster whose cluster-head v satisfies the predicate $P_s(v)$, verifies the Size property in the current configuration and after any computation step. On the contrary, a cluster whose cluster-head v satisfies the Size property but it does not satisfy the predicate $P_s(v)$, may no longer verify the size property after the specific computation step where all nodes of CD_v join v's cluster. This feature is illustrated in Figure 2. In the initial configuration, $Cluster_6 = \{1\}$, and $CD_6 = \{2, 3, 4\}$. Thus, the Size property is satisfied, but the $P_s(6)$ predicate is not satisfied: $|CD_6 \cup Cluster_6| = |\{1, 2, 3, 4\}| > 3$. After the computation step where all nodes of CD_6 join 6's cluster, the Size property is no more satisfied.

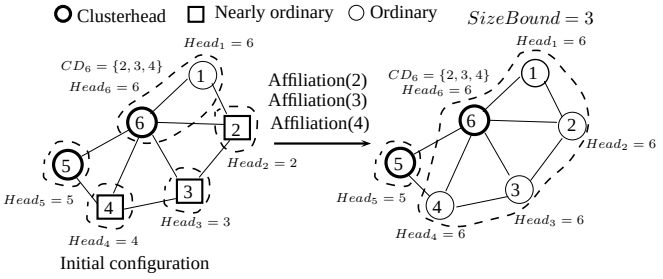


Figure 2: Violation of the Size property from a configuration not satisfying $P_s(v)$

The variable CD_v is computed and updated in such a way that the predicate $P_s(v)$ stays verified after any computation step. For each cluster-head v, the macro $CD2_v$ is used to compute CD_v value. $CD2_v$ is computed in 3 steps. $CD0_v$ is the set of v's neighbors that want to enter into v's cluster, i.e., their weight and their head's weight are smaller than v's weight. The size of $CD0_v$ can be greater than $SizeBound_v - Size_v$: $CD1_v$ is a subset of $CD0_v$, containing at most $SizeBound_v - Size_v$ nodes (having the smallest identifier). 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).

Notation 1. *We note by $CD_v(c)$ and $Cluster_v(c)$ respectively the value of CD_v variable and the value of $Cluster_v$ in the configuration c.*

Figure 3 illustrates why sometimes $CD2_v$ is set to $\emptyset$ whereas $CD1_v \neq \emptyset$. In the initial configuration c_1, there are 5 clusters

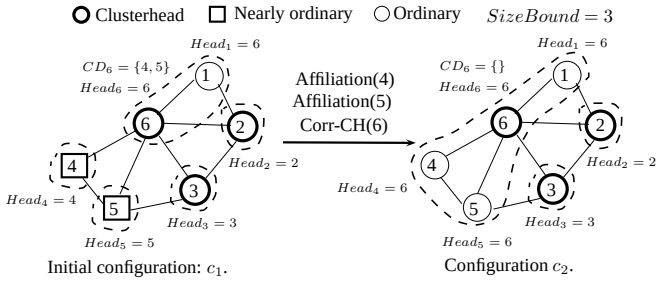


Figure 3: Example of CD value computation

satisfying the Size property, and $CD_6 = \{4, 5\}$. For simplicity, we assume the weight of a node is its identity. Thus, clusterhead 6 has the highest weight in its neighborhood. Nodes 2, 3, 4 and 5 want to belong to $Cluster_6$ (node 1 already belongs to $Cluster_6$); so, $CD0_6 = N_6 - Cluster_6 = \{2, 3, 4, 5\}$. $CD1_6$ contains only two nodes, because $SizeBound = 3$ (for every node) and $|Cluster_6| = 1$; so, $CD1_6 = \{2, 3\}$. In the reached configuration c_2, $CD_6(c_2)$ must be set to $\emptyset$ (so, $CD2_6(c_1) = \emptyset$) because $CD_6(c_1) \nsubseteq \{CD1_6(c_1) \cup Cluster_6(c_1)\}$. This will ensure that whatever the action done by nodes 4 and 5 -initially in CD_6- during the computation step $c_1 \rightarrow c_2$, $P_s(6)$ is still verified in c_2: $|CD_6(c_2) \cup Cluster_6(c_2)| \leqslant SizeBound_6$.

4. Service guarantee of `SG-BSC` protocol

In this section, we prove that `SG-BSC` protocol quickly (in at most 4 rounds) reaches a safe configuration. Moreover, the safety property is preserved under any protocol action and also despite occurrences of $\mathcal{HTD}$ disruptions.

The convergence from an unsafe to a safe configuration is straightforward: once an ordinary node v locally detects an unsafe situation (i.e., its cluster has more than $SizeBound_{Head_v}$ members, its head has ordinary status or it is not v's neighbor), v becomes cluster-head if it cannot join a neighbor cluster without violating *Affiliation* and *Size* properties.

In follows, we prove that the set of configurations A_i is an attractor from A_{i-1} for $1 \leqslant i \leqslant 4$ (Let A_0 be C), where:
$P_t(v) \equiv CD_v = \emptyset$;
$A_1 = \{c \in C \mid \forall v \in V :\ P_s(v) \ \vee P_t(v)$ is satisfied $\}$.
$A_2 = A_1 \cap \{c \in C \mid \forall v \in V :\ P_s(v)$ is satisfied $\}$.
$A_3 = A_2 \cap \{c \in C \mid \forall v \in V :\ |Cluster_v| \leqslant SizeBound_v\}$.
$A_4 = A_3 \cap \{\ c \in C \mid \forall v \in V :$
$(HS_v = O \wedge Head_v \in N_v \wedge HS_{Head_v} \neq O) \vee$
$(HS_v \neq O \wedge S_v \leqslant SizeBound_v \wedge Head_v = v)\ \}$.

Observation 2. *Let v be a node, and cs be a computation step: $c_1 \xrightarrow{cs} c_2$.*
• *According to the macro N^+ and* `Affiliation` *rule:*

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

• *According to the macro $CD2_v$:*

$$CD1_v(c_1) \cap Cluster_v(c_1) = \emptyset \quad (2)$$

$$CD2_v(c_1) = \emptyset \ \vee \ \Big(CD2_v(c_1) = CD1_v(c_1) \wedge CD_v(c_1) \subseteq \{CD2_v(c_1) \cup Cluster_v(c_1)\}\Big) \quad (3)$$

Lemma 1. *Let a computation step: $c_1 \xrightarrow{cs} c_2$ in which a node v performs an action. $P_s(v) \vee P_t(v)$ is satisfied in c_2.*

Proof: Any rule performed by v during cs updates CD_v (see rule actions). If $CD_v(c_2) = \emptyset$ then $P_s(v) \vee P_t(v)$ is satisfied in c_2. Assume that $CD_v(c_2) \neq \emptyset$. During cs, CD_v is set to $CD2_v(c_1)$ and $CD2_v(c_1) \neq \emptyset$. From Equation (3), we have: $CD_v(c_2) = CD1_v(c_1) \wedge CD1_v(c_1) \neq \emptyset$. Since $CD1_v(c_1)$ contains at most $SizeBound_v - |Cluster_v(c_1)|$ elements, then $|CD1_v(c_1) \cup Cluster_v(c_1)| \leqslant SizeBound_v$. Thus, $|CD_v(c_2) \cup Cluster_v(c_1)| \leqslant SizeBound_v$.
Now, 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 (3) and our assumptions, we have: $|CD_v(c_2) \cup CD_v(c_1) \cup Cluster_v(c_1)| \leqslant |CD_v(c_2) \cup Cluster_v(c_1)|$ (because $CD_v(c_2) = CD2_v(c_1)$). Finally, we conclude that $|CD_v(c_2) \cup Cluster_v(c_2)| \leqslant |CD_v(c_2) \cup Cluster_v(c_1)| \leqslant SizeBound_v$. Therefore, $P_s(v) \vee P_t(v)$ is satisfied in c_2 when $CD_v(c_2) \neq \emptyset$. ■

Lemma 2. *A_2 is closed under any computation step.*

Proof: Let v be a node, and c_1 be a configuration in which $P_s(v)$ is satisfied. Assume that a computation step $c_1 \xrightarrow{cs} c_2$ exists, such that $P_s(v)$ is not satisfied in c_2.
In c_1, $|CD_v(c_1) \cup Cluster_v(c_1)| \leqslant SizeBound_v$, and in c_2, $|CD_v(c_2) \cup Cluster_v(c_2)| > SizeBound_v$. So, 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), $z \notin Cluster_v(c_2)$. Thus, v has changed the value of CD_v during cs to include z: $CD_v(c_2) \neq \emptyset$. By Lemma 1, we have: $|CD_v(c_2) \cup Cluster_v(c_2)| \leqslant SizeBound_v$. There is a contradiction: cs does not exist and A_2 is closed. ■

Corollary 1. *A_1 is closed under any computation step.*

Proof: Let v be a node, c_1 be a configuration satisfying $P_s(v) \vee P_t(v)$, and cs be a computation step $c_1 \xrightarrow{cs} c_2$.
If v performs an action during cs then $P_s(v) \vee P_t(v)$ is satisfied in c_2 (Lemma 1). If $P_s(v)$ is satisfied in c_1, then $P_s(v)$ is satisfied in c_2 (A_2 is closed, Lemma 2).
Last case, v does not perform any action during cs and $P_t(v)$ is satisfied in c_1; $P_t(v)$ is still satisfied in c_2. ■

Lemma 3. *If $P_s(v) \vee P_t(v)$ is not satisfied in a configuration c, the node v is enabled in c.*

Proof: Let c be a configuration in which $P_s(v) \vee P_t(v)$ is not satisfied, i.e., $CD_v(c) \neq \emptyset \wedge |CD_v(c) \cup Cluster_v(c)| > SizeBound_v$. According to v's hierarchical status in c, two cases are possible:
• **Case 1 (v is ordinary or nearly ordinary):** v is enabled in c, because `Corr-O-g(v)` or `Corr-NO-g(v)` is satisfied in c ($CD_v(c) \neq \emptyset$).
• **Case 2 (v is cluster-head):** According to the macros $CD2_v$, we have: $CD2_v(c) = \emptyset$ or $CD2_v(c) = CD1_v(c)$.
If $CD2_v(c) = \emptyset$ then $CD_v(c) \neq CD2_v(c)$, `Corr-CH-g(v)` is satisfied in c. Otherwise $CD2_v(c) = CD1_v(c) \neq \emptyset$. We have $0 < |CD1_v(c)| \leqslant SizeBound_v - |Cluster_v(c)|$ (by Equation 2). As $|CD_v(c)| > SizeBound_v - |Cluster_v(c)|$ (by assumption), $CD_v(c) \neq CD1_v(c)$. `Corr-CH-g(v)` is satisfied in c.
Whatever the status of v, v is enabled in c. ■

Corollary 2. *A_1 is an attractor from C in one round.*

Proof: A node v that never satisfy $P_s(v) \vee P_t(v)$ during a computation e, is always enabled (Lemma 3). By fairness, v will perform an action. After v's move, $P_s(v) \vee P_t(v)$ is satisfied in c_2 (Lemma 1). ■

Lemma 4. *A_2 is an attractor from A_1.*

Proof: Let c_1 be a configuration of A_1 but not of A_2. In c_1, there exists a node v verifying $P_t(v) \wedge \neg P_s(v)$, i.e., $CD_v(c_1) = \emptyset \wedge |Cluster_v(c_1)| > SizeBound_v$. While $P_s(v)$ is not satisfied, $P_t(v)$ is verified (A_1 is closed, Lemma 1). So, in c_1, no node can join v's cluster ($CD_v(c_1) = \emptyset$).
Let u be a node of $Cluster_v(c_1)$ ($u \neq v \wedge Head_u = v$); so, $v \notin N_u^+$ is forever satisfied (by definition of N_v^+). In c_1, the node u is enabled whatever its status:
• u **is cluster-head:** The guard `Corr-CH-g`(u) is satisfied ($Head_u \neq u$). Thus, u is enabled: `Resignation(u)` or `Corr-CH(u)` rule is enabled.
• u **is nearly ordinary:** The guard `Corr-NO-g`(u) is satisfied ($Head_u \neq u$). Thus, u is enabled: `Election(u)`, `Affiliation(u)` or `Corr-NO(u)` rule is enabled.
• u **is ordinary:** If $S_{Head_u} \leqslant SizeBound_v$, then $S_v \neq Size_v$, and v is enabled (the guard `Corr-CH-g`(v) is verified). After v's action, we have $S_{Head_u} = Size_v > SizeBound_v$; so, `Change(u)` is satisfied. Therefore, `MustBecomeHead`(u) or `MustAffiliate`(u) is satisfied, and the rule `Election(u)` or `Affiliation(u)` is enabled.
The node u stays enabled unless it performs an action. By fairness, u eventually performs a rule, and gets $Head_u \in N_u^+ \cup \{u\}$ (see rules action), i.e., $Head_u \neq v$. This means that u leaves $Cluster_v$. Thus, eventually a configuration of A_2 in which $|Cluster_v| \leqslant SizeBound_v$ is reached.
A_2 is closed under any computation step (Lemma 2). Therefore, A_2 is an attractor from A_1. ■

Corollary 3. *A configuration of A_2 is reached from A_1 in at most two rounds.*

Proof: In the first round, all cluster-heads v having $S_v \leq SizeBound_v$ and $Size_v > SizeBound_v$ update their variable S_v to set $S_v = Size_v > SizeBound_v$. Thereafter, each node u of $Cluster_v$ is enabled because it verifies the predicate `Change`(u). Thus, at the end of the second round, node u has done an action (to quit the v's cluster) or it is neutralized. The only action that neutralizes this node is setting the variable S_v to a value inferior than $SizeBound_v$; this action occurs when $P_s(v)$ is satisfied. ■

Lemma 5. *A_3 is an attractor from A_2.*

Proof: In A_2, each node v satisfies: $|Cluster_v| \leqslant |CD_v \cup Cluster_v| \leqslant SizeBound_v$. So, $A_3 = A_2$. Therefore, A_3 is an attractor from A_2. ■

Lemma 6. *A_4 is closed under any computation step.*

Proof: Let c_1 be a configuration of A_4. Assume that a computation step $c_1 \xrightarrow{cs} c_2$ exists where $c_2 \notin A_4$. Let $P4a(v)$ be the predicate ($HS_v \neq O \vee Head_v \in N_v$). Let $P4b(v)$ be the predicate ($HS_v = O \vee (S_v \leqslant SizeBound_v \wedge Head_v = v)$). $P4a(v)$ and $P4b(v)$ are verified by v in c_1. The value of $P4a(v)$ and $P4b(v)$ can be modified only by a v's rule; and, after any v's action from $c1$, the predicates $P4a(v)$ and $P4b(v)$ still verified. So, $P4a(v)$ and $P4b(v)$ are also verified in c_2.
Thus, according to our assumption, there exists a node v such that in c_1, ($HS_v = O \wedge HS_{Head_v} \neq O$) but in c_2, ($HS_{Head_v} = O \wedge HS_v = O$). Let u be the v's head in c_1 ($u \neq v$). Only two actions require study: the execution of `Affiliation` rule by v or by u during cs (to change $Head_v$ or to set $HS = O$).
The node u cannot perform `Affiliation` rule during cs, because in c_1, $Size_u \neq 0$ ($v \in Cluster_u$). Assume that v executes `Affiliation` rule during cs. Let z be the cluster-head chosen by v during cs. In c_1, $HS_z = CH$ ($z \in N_v^+$). During cs, z cannot perform `Affiliation` rule (it can only do `Resignation`). In c_2, we have $HS_{Head_v} \neq O$. ■

Lemma 7. *A_4 is an attractor from A_3 in one round.*

Proof: Let v be a node, and c be a configuration of A_3. In c, $Size_v \leqslant SizeBound_v$ (Lemma 5). So, in c if $S_v > SizeBound_v$ then $S_v \neq Size_v$. Assume that c does not belong to A_4. In c, v has one of the following states:
• $HS_v = CH$ **and,** $Head_v \neq v$ **or** $S_v > SizeBound_v$**:** The guard `Corr-CH-g`(v) is satisfied. By fairness, v will perform the `Resignation` or `Corr-CH` rule, and after that $HS_v \neq O \wedge Head_v = v \wedge S_v \leqslant SizeBound_v$ holds.
• $HS_v = O$ **and,** $Head_v \notin N_v$ **or** $HS_{Head_v} = O$**:** `Change`(v) is satisfied; so, the guard `MustBecomeHead`(v) or `MustAffiliate`(v) is verified. As all computations are fair, v eventually performs `Election` or `Affiliation` rule, and after that ($HS_v = CH \wedge Head_v = v \wedge S_v \leqslant SizeBound_v$) $\vee$ ($HS_v = O \wedge Head_v \in N_v \wedge HS_{Head_v} \neq O$) is verified.
• $HS_v = NO$ **and,** $Head_v \neq v$ **or** $S_v > SizeBound_v$**:** The guard `Corr-NO-g`(v) is satisfied. By fairness, v will perform `Election`, `Affiliation` or `Corr-NO` rule. Upon performing `Election` or `Corr-NO` rule, $HS_v \neq O \wedge Head_v = v \wedge S_v \leqslant SizeBound_v$ holds. The execution of `Affiliation` rule makes $HS_v = O \wedge Head_v \in N_v \wedge HS_{Head_v} = CH$ satisfied.
A configuration of A_4 is thus reached from a configuration of A_3. As A_4 is closed under any computation step (Lemma 6), A_4 is an attractor from A_3. ■

Observation 3. *In a configuration of A_4, for every node v, we have: $Head_v \neq v$ if and only if $HS_v = O$.*

Corollary 4. *The set of configurations satisfying the predicate $\mathcal{SP}$ is an attractor.*

Proof: A_4 is attractor (Lemma 7), and once a configuration of A_4 is reached, the predicate $\mathcal{SP}$ is satisfied. ■

Definition 7 (Highly tolerated disruptions). *The set of highly tolerated disruptions $\mathcal{HTD}$ handled by our protocol is:*
• *Change of node's weight,*

• *Crash of ordinary nodes,*
• *Creation of new links between existing nodes (without emergence of new nodes),*
• *Failure of links between two ordinary nodes, or two nodes behaving as leaders (cluster-head or nearly ordinary),*
• *Emergence of sub-networks verifying the predicate $\mathcal{SP}$.*

Lemma 8. *$\mathcal{SP}_v$ is closed under the set of highly tolerated disruptions $\mathcal{HTD}$.*

Proof: Let z be the head of a node v (z is a cluster-head or nearly ordinary). The safety predicate $\mathcal{SP}_v$ ensures that the leader z is neighbor of v, z does not have the ordinary status, and the size of z's cluster is less than $SizeBound_z$.

$\mathcal{SP}_v$ is falsified only following the occurrence of: z's departure or crash, or failure of the link between z and v. Therefore, the safety predicate $\mathcal{SP}$ is preserved under any disruption of $\mathcal{HTD}$. ■

Theorem 1. *The protocol SG-BSC has a service guarantee despite the occurrence of $\mathcal{HTD}$ disruptions.*

Proof: The proof follows directly from corollary 4 and Lemma 8. ■

Time to reach a safe configuration. The convergence time to a safe configuration is the maximum number of rounds needed to reach a safe configuration from any arbitrary initial one. In A_4, the safety predicate $\mathcal{SP}$ is satisfied; so, any configuration of A_4 is safe. Therefore, according to Lemma 7, and Corollaries 2 and 3, the convergence time to a safe configuration is at most **4** rounds.

5. Stabilization of `SG-BSC` protocol

Once a safe configuration is reached, the minimal service is provided. Thereafter, our protocol progresses to reach a legitimate configuration, i.e., to provide the optimum service, preserving the minimal service. The convergence to a legitimate configuration is done in phases. During the phase i, the node of $V_i = V - Set_i$ having the highest weight, named vh_i, becomes cluster-head. The cluster of vh_i is filled out, and members of vh_i's cluster get their final state. At the end of the i^{th} phase, the set of configurations $L_i^{''}$ is reached: all nodes of Set_i have chosen their cluster-head and, they reached their final state (they have stabilized). The system is stabilized when $Set_i = V$. We define sets $L_i^{''}$ and Set_i as follows:

Notation 2. $Set_0 = \emptyset$; $L_0^{''} = A_4$;
$V_i = V - Set_i$; *Set of nodes that do not belong to* Set_i.
Let vh_i *be the node having the highest weight in* V_i;
$L_{i+1} = L_i^{''} \cap \{c \in C |\ HS_{vh_i} = CH\}$;
$Bound_i = \textbf{\textit{Min}}(SizeBound_{vh_i},\ |N_{vh_i} \cap V_i|)$;
$L_{i+1}^{'} = L_{i+1} \cap \{c \in C |\ |Cluster_{vh_i}| = Bound_i\}$;
$Set_{i+1} = Set_i \cup \{vh_i\} \cup Cluster_{vh_i}$;
$L_{i+1}^{''} = L_{i+1}^{'} \cap \{c \in C |\ \forall\ v \in Set_{i+1} :\ (CD_v = \emptyset) \wedge ((HS_v \neq O \wedge S_v = Size_v) \vee (HS_v = O \wedge S_v = 0))\}$;

5.1. Illustration of the stabilization process

Figure 4 illustrates the convergence process from a safe configuration to a legitimate one. For simplicity in this example, the weight of a node is its identifier.

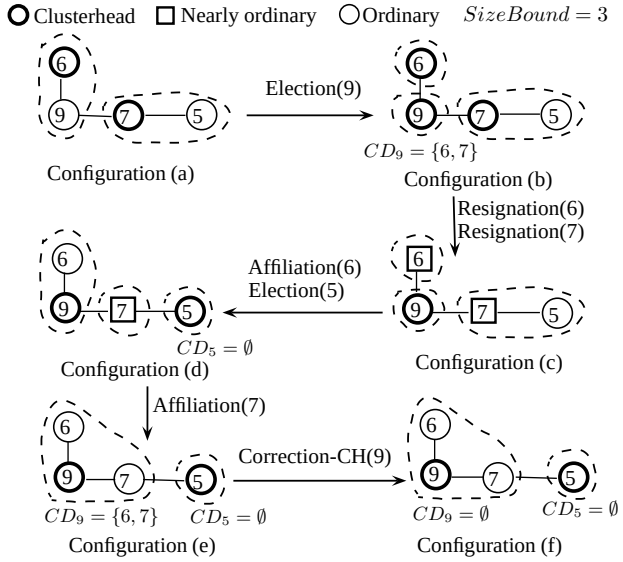


Figure 4: Illustration of convergence to a legitimate configuration.

- Set_0 is the set of nodes having initially their terminal state. Usually, $Set_0 = \emptyset$.
- We name vh_0, the node having the highest weight in $V_0 = V - Set_0$. In Figure 4, $vh_0 = 9$. In configuration (a), node 9 has to be cluster-head (it verifies `MustBecomeHead`(6)). Once node 9 performs `Election` rule, the system reaches a configuration of $L_1 = A_4 \cap \{c \in C \mid HS_{vh_0} = CH\}$ where node 9 will never change its status. Configuration (b) belongs to L_1.
- Now, cluster-heads 6 and 7 have to resign their leadership because they satisfy the guard `MustResign`. These nodes eventually perform the `Resignation` rule, and configuration (c) is reached.
- Node 5 has to leave its cluster because it satisfies the guard `Change`(5). As node 5 cannot join another cluster, then it becomes cluster-head, and configuration (d) is reached.
- Let $L_1^{'} = L_1 \cap \{c \in C \mid Min(SizeBound_{vh_0}, |N_{vh_0} \cap V_0|) = |Cluster_{vh_0}|\}$ be the set of configurations where vh_0's cluster is stable (no node will quit or join this cluster). Configuration (e) belongs to $L_1^{'}$.
- In the last step of phase 1, all nodes of vh_0's cluster reach their terminal state (a configuration of $L_1^{''}$). The configuration (f) belongs to $L_1^{''}$, where $L_1^{''} = L_1^{'} \cap \{c \in C |\ \forall\ v \in Set_0 \cup \{vh_0\} \cup \{Cluster_{vh_0}\}, CD_v = \emptyset\}$.

At the end of first phase, set of nodes having their final state is $Set_1 = Set_0 \cup \{vh_0\} \cup \{Cluster_{vh_0}\}$. Each phase i is similar to the first one: the node of $V_i = V - Set_i$ having the highest weight, named vh_i, becomes cluster-head. The cluster of vh_i is filled out, and members of vh_i's cluster get their final state. At the end of i^{th} phase, the set of nodes having their final state is $Set_i = Set_{i-1} \cup \{vh_i\} \cup \{Cluster_{vh_i}\}$.

5.2. Proof of stabilization

We will prove that configurations set $L_j^{''}$ is an attractor from C such that at the j^{th} phase $Set_j = V$.

Observation 4. *In each phase i, the set Set_i increases up to contain all nodes.*

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

$$If\ (v \in Set_i)\ then\ Head_v \in Set_i \quad (5)$$

$$If\ (v \in V_i)\ then\ Head_v \notin Set_i \quad (6)$$

For any value of i, any configuration of $L_i, L_i^{'}$ or $L_i^{''}$ belongs to A_4, because $L_i, L_i^{'}$ and $L_i^{''}$ are subsets of $L_0^{''} = A_4$.

Lemma 9. *For any value of i, L_{i+1} is an attractor from C assuming that $L_i^{''}$ is an attractor from C.*

Proof: Assume that $L_i^{''}$ is an attractor from C. Let c be a configuration of $L_i^{''}$ but not of L_{i+1}. In c we have: $\forall u \in N_{vh_i}$, if $u \in V_i$ then $w_{vh_i} > w_u$ (by definition of vh_i); else (i.e., $u \in Set_i$) $CD_u = \emptyset$. So, $N^+_{vh_i}$ is empty forever. In c, there are two possibilities:

• vh_i **is nearly ordinary**: the guard `MustBecomeHead`(vh_i) is forever satisfied. By fairness, vh_i eventually performs the `Election` rule. After that we get: $HS_{vh_i} = CH$.

• vh_i **is ordinary**: from Observation 3, $Head_{vh_i} \neq vh_i$. According to Equation 6 and to the definition of vh_i, we have: $w_{vh_i} > w_{Head_{vh_i}}$. So, `MustBecomeHead`($vh_i$) is forever satisfied. By fairness, vh_i eventually performs the `Election` rule. After that we get: $HS_{vh_i} = CH$.

In both cases L_{i+1} is reached from $L_i^{''}$. Furthermore, the value of HS_{vh_i} is never modified, because the rule `Resignation`(vh_i) is never enabled ($N^+_{vh_i} = \emptyset$ forever). Thus, L_{i+1} is an attractor from $L_i^{''}$. ■

Corollary 5. *The set of configurations L_{i+1} is reached from $L_i^{''}$ in at most one round.*

Lemma 10. *Let c_1 be a configuration of L_{i+1}. Assuming that L_{i+1} is an attractor from C. For any computation step $c_1 \xrightarrow{cs} c_2$, we have: $Cluster_{vh_i}(c_1) \subseteq Cluster_{vh_i}(c_2)$.*

Proof: Let u be a node of $Cluster_{vh_i}$ ($Head_u = vh_i$).

In c_1, $HS_u = O$ (Observations 3); and $\forall z \in N_u$, if $z \in V_i$ then $w_{Head_u} > w_z$; else (i.e., $z \in Set_i$) $CD_z = \emptyset$. We conclude that $z \notin N_u^+$, so N_u^+ is empty forever.

The predicate `MustAffiliate`(u) is never verified, so the node u cannot change its cluster-head.

The guard `Change`(u) is never verified, because in L_{i+1} we have: $S_{Head_u} \leqslant SizeBound_{Head_u}$ (Lemma 7), $vh_i \in N_u$ (Lemma 7), $w_{vh_i} > w_u$ (by definition of vh_i), and $HS_{vh_i} = CH$ (Lemma 9). Thus, the guard `MustBecomeHead`(u) is also never verified and u cannot become a cluster-head.

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

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

Proof: In a configuration c of L_{i+1}, $HS_{vh_i} = CH$. Let u be a node of $\{N_{vh_i} \cap Set_i\}$. Thus, there exists an integer j, $(0 < j < i)$, such that: $(u = vh_j)\ \vee\ (u \in Cluster_{vh_j})$.

In L_{j+1}, u can never leave $Cluster_{vh_j}$ (see Lemma 10). 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 $Cluster_{vh_i}$, i.e., $Cluster_{vh_i} \subseteq \{N_{vh_i} \cap V_i\}$.

In L_{i+1}, $|Cluster_{vh_i}| \leqslant SizeBound_{vh_i}$ (Observation 4), thus $|Cluster_{vh_i}| \leqslant Bound_i$ is forever verified.

Once $|Cluster_{vh_i}| = Bound_i$, $Cluster_{vh_i}$ is never modified (no node leaves $Cluster_{vh_i}$, see Lemma 10).

Assume that $|Cluster_{vh_i}| < Bound_i$. $CD1_{vh_i} \neq \emptyset$ because $CD1_{vh_i}$ contains $Bound_i - |Cluster_{vh_i}|$ elements of $\{N_{vh_i} \cap V_i\} - Cluster_{vh_i}$.

While $CD_{vh_i} \neq CD2_{vh_i}$, the node vh_i is enabled (`Corr-CH-g`(vh_i) is verified). By fairness, vh_i performs `Corr-CH` rule, and gets $CD_{vh_i} = CD1_{vh_i}$ or $CD_{vh_i} = \emptyset$ (now, $CD_{vh_i} \subseteq \{CD1_{vh_i} \cup Cluster_{vh_i}\}$). In the last case, `Corr-CH-g`(vh_i) is still verified till vh_i performs again the `Corr-CH` rule. After that we have: $CD_{vh_i} = CD1_{vh_i} \neq \emptyset$.

Let v be a node of CD_{vh_i}. We have $vh_i \in N_v^+$: $N_v^+ \neq \emptyset$. v is enabled whatever its status:

• v **is ordinary**: `MustAffiliate`(v) is always verified.

• v **is nearly ordinary**: All nodes z of $Cluster_v$ satisfy `Change`(z) ($HS_{Head_z} \neq CH$), and are enabled because `MustBecomeHead`(z) or `MustAffiliate`(z) is satisfied. After z's action, $Head_z \neq v$ holds; so, the size of $Cluster_v$ decreases. Eventually a configuration where $Size_v = 0$ is reached, and `MustAffiliate`(v) becomes satisfied.

• v **is cluster-head**: `MustResign`(v) is always verified. After performing the `Resignation` rule, v becomes nearly ordinary node. Eventually it will reach a configuration where `MustAffiliate`(v) is satisfied (see previous case).

The node vh_i has a higher weight than nodes of N_v^+, because $\forall z \in N_v$, if $w_z > w_{vh_i}$ then $z \in Set_i$ and $z \notin N_v^+$ ($CD_z = \emptyset$). By fairness v performs the `Affiliation` rule, and it chooses vh_i as cluster-head. Eventually, when each node v of CD_{vh_i} performs `Affiliation` rule, a configuration where $|Cluster_{vh_i}| = Bound_i$ is reached. ■

Corollary 6. *The configurations set $L_{i+1}^{'}$ is reached from L_{i+1} in at most five rounds.*

Proof: During the first and second rounds, vh_i sets CD_{vh_i} to $CD2_{vh_i}$. During the third round, every cluster-head $v \in CD_{vh_i}$ is enabled and it performs the `Resignation` rule to become nearly ordinary. At the end of this round, each node v of CD_{vh_i} is either nearly ordinary or ordinary.

During the fourth round, every ordinary node u member of $Cluster_v$, where v is nearly ordinary, leaves its cluster. So, at the end of this round, $Size_v = 0$. During the fifth (last) round, all nodes v of CD_{vh_i} perform the `Affiliation` rule to join the cluster of vh_i. ■

Lemma 12. *In $L_{i+1}^{'}$, for any node v of Set_{i+1}, $HS_v \neq NO$ and the* `Election`, `Affiliation`, `Resignation` *and* `Corr-NO` *rules are disabled.*

Proof: Let v be a node of Set_{i+1}. There exists an integer j, $0 < j \leqslant i$, such that $(v \in Set_{j+1}) \wedge (v \notin Set_j)$. So, $(v = vh_j) \vee (v \in Cluster_{vh_j})$.
In a configuration of L'_{j+1}, if $(v = vh_j)$ then v is cluster-head (by definition of L'_{j+1} and Lemma 9); else (i.e. $v \in Cluster_{vh_j}$) v is ordinary node (Observation 3).
Thus, in L'_{j+1} the node v is either cluster-head or ordinary ($HS_v \neq NO$), and `Corr-NO` rule is disabled for v.
• v **is cluster-head** ($v = vh_j$): According to definition of L'_{j+1} and Lemma 9, v can never execute the `Resignation` rule. `Election` and `Affiliation` rules are also disabled for v (see protocol rules).
• v **is ordinary** ($v \in Cluster_{vh_j}$): According to definition of L'_{j+1} and Lemma 10, v can never execute the `Election` or `Affiliation` rule to leave the cluster. `Resignation` rule is also disabled for v (see protocol rules).

According to that, in L'_{j+1} a node of Set_{j+1} can never perform the `Election`, `Affiliation`, `Resignation` and `Corr-NO` rules. This is also verified in L'_{i+1}, because $L'_{i+1} \subseteq L'_{j+1}$ $(j \leqslant i)$. ■

Lemma 13. *In L'_{i+1}, $CD2_{vh_i} = \emptyset$.*

Proof: In L'_{i+1}, we have $|Cluster_{vh_i}| = Bound_i$ (Lemma 11). If $|Cluster_{vh_i}| = SizeBound_{vh_i}$, then $CD1_{vh_i} = \emptyset$. Else (i.e., $|Cluster_{vh_i}| = |N_{vh_i} \cap V_i|$), no node u of N_{vh_i} can join $Cluster_{vh_i}$, because $w_{Head_u} \geqslant w_{vh_i}$. So, $CD0_{vh_i} = \emptyset$. Therefore, in L'_{i+1}, $CD2_{vh_i} = \emptyset$. ■

Lemma 14. *For any value of i, L''_{i+1} is an attractor from C assuming that L'_{i+1} is an attractor from C.*

Proof: Let v be a node of Set_{i+1} but not of Set_i. From Lemma 12, $HS_v \neq NO$; thus two cases are possible:
• v **is ordinary:** If $CD_v \neq \emptyset$ or $S_v \neq 0$, then v is enabled (`Corr-O-g`(v) is verified). By fairness, the node v eventually performs `Corr-O` rule. After that, $CD_v = \emptyset \wedge S_v = 0$ holds forever.
• v **is cluster-head:** By definition, v is vh_i. In L'_{i+1}, we have $CD2_{vh_i} = \emptyset$ (Lemma 13). If $CD_{vh_i} \neq \emptyset$ (i.e., $CD_{vh_i} \neq CD2_{vh_i}$) or $S_v \neq Size_v$, then `Corr-CH-g`(vh_i) is verified. By fairness, vh_i will perform `Corr-CH` rule. After that, $CD_{vh_i} = \emptyset \wedge S_v = Size_v$ holds forever. ■

Corollary 7. *The configurations set L''_{i+1} is reached from L'_{i+1} in at most one round.*

Theorem 2. *The system eventually reaches a configuration of L''_j, where $Set_j = V$, and any configuration of L''_j is a terminal configuration.*

Proof: According to Observation 4, there exists j such that $Set_j = V$. Since $L''_0 = A_4$ is an attractor from C (Lemma 7), then for any value of i, the sets L_i, L'_i, L''_i are attractors from C (Observation 4, and Lemmas 9, 11, 14). Therefore, we conclude that L''_j is an attractor from C.
In a configuration of L''_j, we have:
• A node v cannot perform `Election`, `Resignation`, `Affiliation` and `Corr-NO` rules (Lemma 12).
• $CD_v = \emptyset$ for any node v (By definition of L''_j).
• If v is a cluster-head then $Head_v = v$ (Observation 3), $CD2_v = \emptyset$ (see lemma 13), and $S_v = Size_v$. Thus, v cannot perform the rule `Corr-CH`.
• If v is an ordinary node then $S_v = 0$. Thus, v cannot perform the rule `Corr-O`.
In a configuration of L''_j, v can never perform an action. Thus, a terminal configuration is reached. ■

5.3. Proof of correctness

In this section, we will prove that any terminal configuration is legitimate, i.e. the well balanced clustering properties are satisfied in this configuration.

Theorem 3. *In a terminal configuration, the well-balanced clustering properties are satisfied.*

Proof: Let j be an integer where $Set_j = V$. According to Theorem 2, L''_j is an attractor from C, and any terminal configuration belongs to L''_j. Let c be a terminal configuration of L''_j. In c, $CD_v = \emptyset$ for any node v; and v is not a nearly ordinary node ($L''_j \subseteq L'_j$ and Lemma 12). In c, two cases are possible:
• v **is ordinary**: The guard `MustAffiliate`(v) is unsatisfied in c. Thus, $Head_v \in N_v \wedge HS_{Head_v} = CH \wedge w_{Head_v} > w_v$. Therefore, the *Affiliation property* is fulfilled in c.
• v **is cluster-head**: In A_3, $|Cluster_v| \leqslant SizeBound_v$. As A_3 is an attractor from C (Lemma 5). Thus, in c the *Size property* is satisfied.
Assume now that there exists a node $z \in N_v$ such that: $HS_z = CH \wedge w_z > w_v \wedge Size_z < SizeBound_z$, i.e., the *Cluster-heads neighbor property* is not satisfied. The node v belongs to $CD0_z$ because $w_{Head_v} = w_v < w_z$. Moreover, $CD1_z \neq \emptyset$ because $Size_z < SizeBound_z$. Since, in the configuration c $CD_z = \emptyset$; then according to the macro $CD2$, $CD2_z \neq \emptyset$ in c. Thus, `Corr-CH-g`(z) is satisfied ($CD_z \neq CD2_z$). The configuration c is not terminal; there is a contradiction. We conclude that, in c, for any cluster-head $z \in N_v$: $(w_v > w_z) \vee (Size_z = SizeBound_z)$. Thus, the *Cluster-heads neighbor property* is verified in c. ■

5.4. Stabilization time

The set L_{i+1} is reached from L''_i in at most one round (Corollary 5). Five more rounds are needed to reach L'_{i+1} from L_{i+1} (Corollary 6). The set L''_{i+1} is reached from L'_{i+1} in at most one round (Corollary 7).

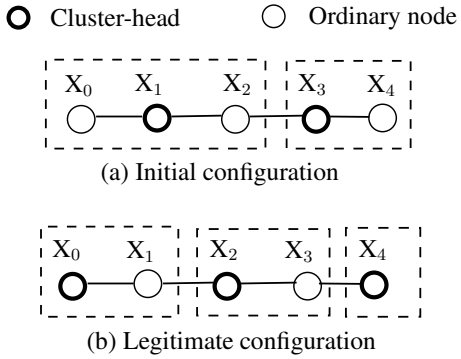
Note that when the vh_i's cluster is empty, i.e., no ordinary node belongs to vh_i's cluster, configuration sets L'_{i+1} and L''_{i+1} are reached upon L_{i+1} is reached (no additional rounds are required). Thus, L''_{i+1} is reached from L''_i in at most one round. So, in the worst case: (1) no cluster is empty: seven rounds are necessary to reach L''_{i+1} from L''_i, for $0 < i \leqslant j$, and (2) each cluster contains exactly one ordinary node ($\frac{|V|}{2}$ clusters).
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 is the number of built clusters $j = \frac{|V|}{2}$.

The time to reach $L_0^{''}$ (A_4) is 4 rounds (see Section 4). Therefore, along any computation and from any initial configuration, the legitimate configuration is reached in at most $\frac{7*|V|}{2} + 4$ rounds.

Note that any self-stabilizing weight-based clustering protocol requires $O(N)$ rounds to stabilize (Theorem 4).

Theorem 4. *The convergence time of a self-stabilizing weight-based clustering protocol is intrinsically proportional to the network size.*

Proof: Let us study the example of network presented in Figure 5. In initial configuration, there are $(N-1)/2$ clusters where N is the number of nodes (network size). For any value of i that is old, X_i is a cluster-head, X_{i+1} is affiliated with X_i, and X_0 is member of X_1's cluster. Weights of nodes are ordered as follows: $w_{X_i} > w_{X_{i+1}}$.
The legitimate configuration is defined as: for any value of i that is even, X_i is a cluster-head, and X_{i+1} affiliates with X_i. To reach such legitimate configuration, each node has to change its role. X_{i+1} detects that it has to change its role only after a change on X_i's role. Clearly, X_i can change its role only after i rounds. Therefore, the convergence time is $O(N)$ rounds. ■



(a) Initial configuration

(b) Legitimate configuration

Figure 5: Convergence time

6. Simulation results

In this section, we present an experimental study based on a comparison between `BSC` and `SG-BSC` protocols. Simulation experiments are carried out thanks to the `NS2.34` simulator. Protocols are evaluated in the context of mobile ad-hoc networks. So to achieve this study, protocols were adapted to the message passing model. Each node v broadcasts periodically (once per 0.4 second) to its direct neighbors a message containing its state. Based on this message, v's neighbors decide to update their states or not. After a change in its state, a node broadcasts its new state to its neighbors. If during a certain period (in our simulations, 0.85 second), no message is received from a neighbor node, it is assumed to be out of the neighborhood.

The default parameters value used during simulation are presented in Table 1. Our network is composed of mobile nodes, with a propagation radio range of $250m$, randomly placed within a $1200m * 1200m$ area.

Parameter	Value
Simulation time	100s
Number of nodes	70
Transmission range	250m
Network area	1200m*1200m
Speed	0m/s - 12m/s
Pause time	0.5s
Wmin	50
Wmax	80
Δ	2
SizeBound	10
freq	2c/s

Table 1: Default value of simulation parameters.

Mobility model. Each node moves randomly according to the *Random Waypoint model* [7]. Initially, network nodes are randomly placed in the network area. Each node selects a random destination and moves to it with a randomly chosen speed (between 0 m/s and 12 m/s). Upon reaching this destination, another random speed and destination are targeted after a pause time. The process is repeated until the end of simulation.

Weight variation model. Each node randomly chooses its initial weight between two values `Wmin` and `Wmax`. The node's weight changes according to a frequency `freq`, which is the number of changes per second. Based on the frequency value, the time when a node undergoes the weight change is chosen randomly. We limit the variation of weight to a parameter Δ. Thus, the new weight of a node is randomly chosen between $w - \Delta$ and $w + \Delta$, where w is the current weight of node.

Observed metrics. To evaluate the interest of the service guarantee, and to analyze its cost, four metrics are studied:
• *The average number of leaders*. It is the number of nodes having either cluster-head or nearly ordinary status.
• *The availability of minimal service*. It represents the percentage of time where the minimal service is available.
• *The availability of optimum service*. It represents the percentage of time where the optimum service is available.
• *The average stabilization time*. It is the time (in seconds) required to reach a legitimate configuration for the first time (i.e., from the initial configuration) whatever the occurrence of perturbations during this period.
Simulation scenarios. Three types of simulation scenarios are conducted:
• *Network size variation*: nodes are not mobile, their number varies between 10 and 70, and `freq` = 2 c/s.
• *Weight variation*: the considered frequency values are: 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 1, 2, 3, 4 and 5 c/s. The network contains 70 static nodes.
• *Mobility variation*: the network contains 70 nodes where `freq` = $2c/s$, and speed of nodes is varied from $0m/s$ to $12m/s$.

For both protocols, identical mobility and weight variation scenarios are used in order to gather fair results. During each simulation, measurements are collected every 0.02 second to obtain the average values. Furthermore, to get accurate results,

each simulation is driven with thirty different runs. The observed metrics are then averaged on these different runs. In order to show how these average values are confident, a confidence interval is computed using the confidence level 95%.

6.1. Availability of optimum service

The Availability of optimum service as a function of the network size, weight change frequency and nodes speed are presented in Figures 6(a), 6(b) and 6(c).

In both BSC and SG-BSC protocols, reconstruction of clusters by chain reaction may occur after a single event (election or motion of a cluster-head). In fact, when two cluster-heads become neighbor, one of them must defer to the other (to satisfy *Cluster-heads neighbor property*). This situation may trigger Election/Resignation of leaders that propagates throughout the entire network, generating a complete reconstruction of clusters. Such effect is called *chain reaction* (an example is given in Figure 5). Due to chain reaction caused by weight changes or motion of nodes, the hierarchical structure is continuously reconstructed in order to achieve the *well-balanced clustering properties*. As a result, the optimum service is often broken, so it is not highly available.

We observe also that the optimum service is slightly less available in SG-BSC protocol compared to BSC protocol. The largest margin observed is 6% in large and dynamic networks. This is caused by the convergence time towards the optimum service (i.e., the stabilization time). In fact, Figure 7 shows that the time required by SG-BSC protocol to reach the optimum service is larger than the one required by BSC protocol. This feature is due to the resignation process used by SG-BSC protocol that consists to slowdown the stabilization process in order to maintain the minimal service.

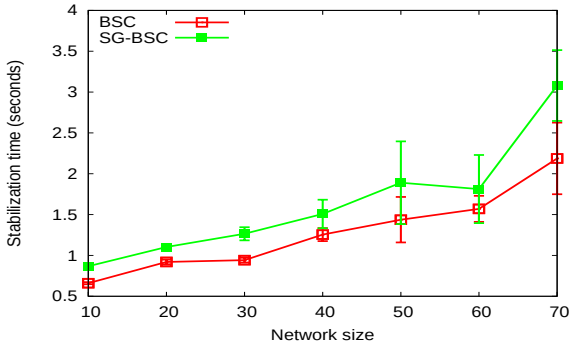


Figure 7: Stabilization time in function of the network size

6.2. Average number of leaders

The number of selected leaders is proportional to the network size (shown in Figure 8).

In a large scale network, SG-BSC protocol generates a slightly higher number of leaders than BSC (about 1 additional leader in a network of 70 nodes). Recall that in SG-BSC protocol, a node cannot freely resign its leadership: it becomes a nearly ordinary node and so it still behaves as leader. A nearly ordinary node may become ordinary only once its cluster is empty; and during all this period it rank among the leaders. The slowdown of during resignation process explains why the average number of leaders is sightly higher in SG-BSC protocol than BSC protocol.

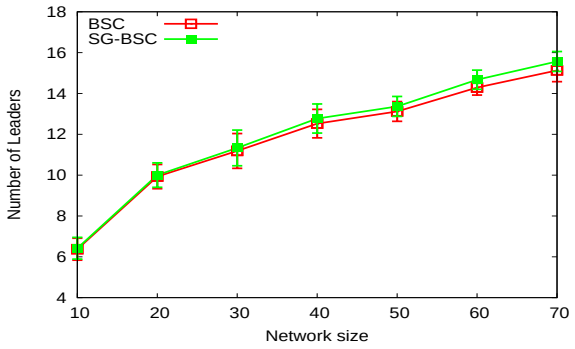


Figure 8: Average number of Leaders

6.3. Availability of minimal service

The availability of minimal service according to the network size and the frequency of weight variation are illustrated respectively in Figures 9(a) and 9(b).

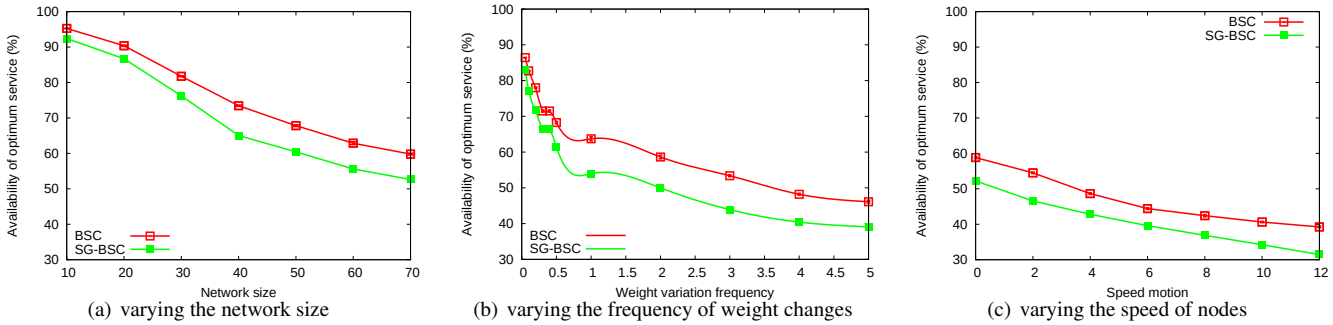
In SG-BSC protocol, the minimal service is highly available (about 99% of time) whatever the network size and the weight variation frequency. Whereas, in BSC, the minimal service is often broken: unavailable during almost 20% of time when $freq = 5\ c/s$. This difference on protocol's performance reflects the interest of self-stabilization with service guarantee in large scale networks compared to classical self-stabilization.

The rupture of minimal service in BSC protocol happens during reconstruction of clusters (due to weight variation or leaders motion). In particular, when reconstruction of clusters by chain reaction occurs, it generates a continuous disruption of minimal service (by creating orphan nodes).

The performance of SG-BSC shows that using the service guarantee property improves the availability of minimal service despite the occurrence of chain reaction.

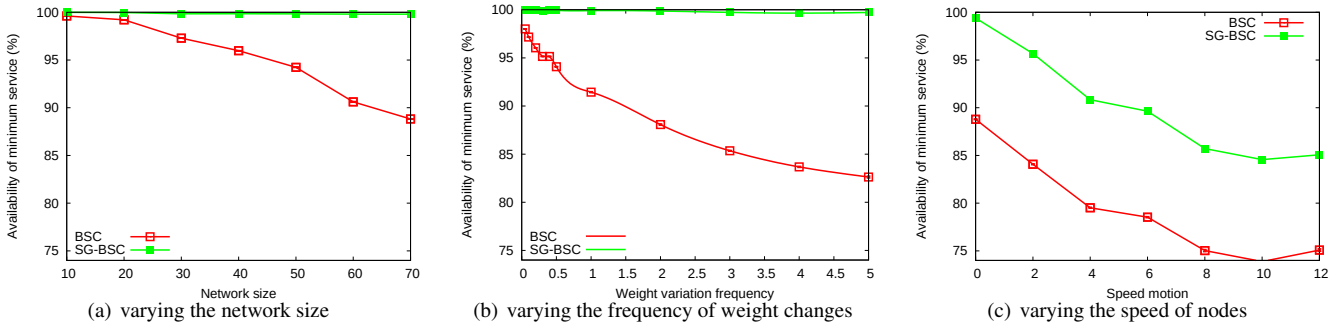
It is proved in section 4 that SG-BSC protocol maintains the minimal service whatever the network size and frequency of weight variation. The rupture observed (less than 1%), in large scale network or when the frequency is very high, is due to the occurrence of disruptions other than $\mathcal{HTD}$ (loss and unordered message reception). In fact, in SG-BSC protocol (resp. in BSC protocol), when a node undergoes a weight change, it broadcasts a message to its neighbors indicating its new state. When frequency of weight variation is very high, the number of exchanged messages is important, and events like message loss and unordered message reception happen more frequently. Owing to these disruptions, an ordinary node can affiliate with a node (by considering it as cluster-head) which is no more cluster-head (it already resigned its leadership). This situation falsifies the minimal service.

Increasing the speed of nodes has a negative impact on the availability of minimal service (see Figure 9(c)). In a dynamic network, due to nodes motion, an ordinary node and its cluster-head may be outside the transmission range one of the other, i.e., they are no longer neighbors. This situation breaks the minimal service. However, even in dynamic networks, the minimal service is more preserved by the SG-BSC protocol than by the BSC protocol.



(a) varying the network size (b) varying the frequency of weight changes (c) varying the speed of nodes

Figure 6: Availability of optimum service



(a) varying the network size (b) varying the frequency of weight changes (c) varying the speed of nodes

Figure 9: Availability of minimal service

7. Conclusion

We introduced the self-stabilization with service guarantee approach, it extends self-stabilization, safe convergence and super-stabilization approaches, by improving their service guarantee. As an application, we propose SG-BSC protocol building 1-hop bounded size weight based clusters. Proofs of stabilization, service guarantee, correctness, time complexity as well as simulation results are provided.
SG-BSC protocol requires $O(log\ N)$ bits per node.

Compared to BSC, SG-BSC is scalable, and it is more suitable for large-scale modern distributed systems such as mobile ad-hoc networks. This is due firstly to the constant time (4 rounds at most) required to provide an useful minimal service, and secondly to the maintain of minimal service during progress of the protocol towards the optimum service.

Providing the service guarantee have a price in terms of number of leaders and stabilization time. In fact, theoretically, the service guarantee slowdown the stabilization process by a multiplicative factor of $\frac{7}{2}$ rounds in the case of our protocol (stabilization time of BSC is N rounds, whereas the one of SG-BSC is $\frac{7*N}{2} + 5$ rounds). Simulation results pointed out that the service guarantee induces also an increase in the average number of leaders, however really negligible (about 1 additional leader in a network of 70 nodes).

The minimal service offered by SG-BSC protocol guarantees that the entire network is partitioned into bounded clusters. By simulation, we show that when the minimal service is deteriorated in BSC, it stays highly available in SG-BSC whatever the network size and reconstruction of clusters frequency. Thus, using SG-BSC (or any self-stabilizing with service guarantee) protocol as a clustering protocol ensures the continuity of operation of upper-layer hierarchical protocols, like hierarchical routing protocols.

References

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

[2] 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), 1999, pp. 889–893.

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

[4] D. Bein, A. K. Datta, C. R. Jagganagari, 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), IEEE Computer Society, 2005, pp. 436–441.

[5] A. Bui, S. Clavière, A. K. Datta, L. L. Larmore, D. Sohier, Self-stabilizing hierarchical construction of bounded size clusters, in: the 18th International Colloquium on Structural Information and Communication Complexity, SIROCCO'11, springer LNCS 6796, 2011, pp. 54–65.

[6] A. Bui, A. K. Datta, F. Petit, V. Villain, Snap-stabilization and PIF in tree networks, Distributed Computing 20 (1) (2007) 3–19.

[7] T. Camp, J. Boleng, V. Davies, A survey of mobility models for ad hoc network research, Wireless communications and Mobile computing 2 (5) (2002) 483–502.

[8] E. Caron, A. K. Datta, B. Depardon, L. L. Larmore, self-stabilizing k-clustering algorithm for weighted graphs, Journal of Parallel and Distributed Computing 70 (11) (2010) 1159–1173.

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

[10] A. Dasgupta, S. Ghosh, X. Xiao, Fault-containment in weakly-stabilizing systems, in: the 11th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS'09), Springer LNCS 5873, 2009, pp. 209–223.

[11] A. K. Datta, L. L. Larmore, P. Vemula, A self-stabilizing o(k)-time k-clustering algorithm, Computer Journal 53 (3) (2010) 342–350.

[12] S. Delaët, S. Devismes, M. Nesterenko, S. Tixeuil, Snap-stabilization in message-passing systems, in: the 10th International Conference on Distributed Computing and Networking (ICDCN'09) Springer LNCS 5408, vol. 5408, 2009, pp. 281–286.
[13] M. Demirbas, A. Arora, V. Mittal, V. Kulathumani, A fault-local self-stabilizing clustering service for wireless ad hoc networks, IEEE Transactions on Parallel and Distributed Systems 17 (9) (2006) 912–922.
[14] E. W. Dijkstra, Self-stabilizing systems in spite of distributed control, Communications of the ACM 17 (11) (1974) 643–644.
[15] S. Dolev, T. Herman, Superstabilizing protocols for dynamic distributed systems, Chicago J. Theor. Comput. Sci. 1997.
[16] S. Dolev, A. Israeli, S. Moran, Uniform dynamic self-stabilizing leader election, IEEE Transactions on Parallel and Distributed Systems 8 (4) (1997) 424–440.
[17] V. Drabkin, R. Friedman, M. Gradinariu, Self-stabilizing wireless connected overlays, in: the 10th International Conference On Principles Of Distributed Systems (OPODIS'06), Springer LNCS 4305, 2006, pp. 425–439.
[18] B. Ducourthial, S. Khalfallah, F. Petit, Best-effort group service in dynamic networks, in: the 22nd Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'10), 2010, pp. 233–242.
[19] M. Gerla, J. T. Tsai, Multicluster, mobile, multimedia radio network, Journal of Wireless Networks 1 (3) (1995) 255–265.
[20] N. Guellati, M. Kheddouci, A survey on self-stabilizing algorithms for independence, domination, coloring, and matching in graphs, Journal of Parallel and Distributed Computing 70 (4) (2010) 406–415.
[21] S. D. Joffroy Beauquier, S. Haddad, A 1-strong self-stabilizing transformer, in: the 8th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS'06), Springer LNCS 4280, 2006, pp. 95–109.
[22] C. Johnen, F. Mekhaldi, Self-stabilizing computation and preservation of knowledge of neighbor clusters, in: IEEE International Conferences on Self-Adaptive and Self-Organizing Systems (SASO'11), 2011, pp. 41–50.
[23] C. Johnen, 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), 2008, pp. 43–50.
[24] H. Kakugawa, 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.
[25] S. Kamei, 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, 2008, pp. 496–511.
[26] F. Kuhn, T. Moscibroda, Distributed approximation of capacitated dominating sets, in: the 19th annual ACM symposium on Parallel algorithms and architectures, SPAA'07, 2007, pp. 161–170.
[27] S. Kutten, B. Patt-Shamir, Stabilizing time-adaptive protocols, Theoretical Computer Science 220 (1) (1999) 93–111.
[28] C. R. Lin, M. Gerla, Adaptive clustering for mobile wireless networks, IEEE Journal on Selected Areas in Communications 15 (7) (1997) 1265–1275.
[29] N. Mitton, E. Fleury, I. Guérin-Lassous, S. Tixeuil, Self-stabilization in self-organized multihop wireless networks, in: the 25th IEEE International Conference on Distributed Computing Systems Workshops (WWAN'05), 2005, pp. 909–915.
[30] O. Tomoyuki, I. Shinji, K. Yoshiaki, I. Kenji, M. Kaori, An adaptive maintenance of hierarchical structure in ad hoc networks and its evaluation, in: the 22nd International Conference on Distributed Computing Systems (ICDCS'02), 2002, pp. 7–13.