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

Colette Johnen★, Fouzi Mekhaldi★

★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 $2N + 4$ 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 $P$ quickly provides a minimal service (called also a safety property). Once the minimal service is offered, $P$ 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 $HTD$. 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 $HTD$ 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.
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 $F \leq N/2$. 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 $HTD$ 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., reconstitution 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 $HTD$), 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 $HTD$ 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 [3], 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 protocol, 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, 19, 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 3). 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 \( \frac{7N}{2} + 4 \) 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) ∈ 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 = {w ∈ V | (u, v) ∈ 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: << Guard >> = < Action >>. 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 → 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: i.e., c_i → 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 E the set of all weakly fair computations of the system. The set of weakly fair computations starting from a particular configuration c ∈ C is denoted E_c. E_A denotes the set of weakly fair computations where the initial configuration belongs to the set of configurations A ⊆ C.

A node v is neutralized during the computation step c_i → 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: ∀e ∈ B_1, If (E_c = 0) then e ∈ B_2
- Closure: ∀e ∈ E_{B_1}(e = c_1, c_2, ...), ∀i ⩾ 1, c_i ∈ B_2

Definition 2 (Self-stabilization). A distributed system S is self-stabilizing if and only if there exists a non-empty set L ⊆ C, called set of legitimate configurations, such that:
- L is an attractor from C.
- Each configuration of 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 P is the maximal number of rounds for all computations of P reaching a legitimate configuration from any initial one.

2.2. Self-stabilization with service guarantee

A protocol P is self-stabilizing with service guarantee if and only if:
1. P is self-stabilizing;
2. From any arbitrary configuration, 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 P towards the optimum service, i.e., during stabilization, under any action of P;
4. The minimal service (safety property) is maintained despite occurrences of some disruptions, called Highly Tolerated Disruptions, denoted HTD.

In general case, the set of 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 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 HTD disruptions.
Definition 3 (Stabilization with service guarantee). A self-stabilizing system $S$ has service guarantee despite occurrences of 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 with $\mathcal{HTD}$.
- $\mathcal{SP}$ and $\mathcal{HTD}$ fulfilled by our protocol are defined respectively in Definitions 5 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 $w_v$, 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 $\text{SizeBound}$, 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 $C_1, C_2, \ldots, C_n$ 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 $\text{Head}_i$, and the ordinary nodes of the cluster $C_i$ are at distance 1 from $C_i$’s head, i.e., $\bigcap_{i=1}^n C_i = \emptyset$, $\bigcup_{i=1}^n C_i = V$, $\forall 1 \leq i \leq n$, $\text{Head}_i \in C_i$, and $C_i \subseteq N_{\text{Head}_i} \cup \{\text{Head}_i\}$.
- Affiliation property: The head of $v$’s cluster has a weight greater than $v$’s weight, i.e., $\forall 1 \leq i \leq n$, $w_{\text{Head}_i} = \max\{w_v \forall v \in C_i\}$.
- Size property: The cluster $C$ headed by $u$ contains at most $\text{SizeBound}$ ordinary nodes, i.e., $\forall 1 \leq i \leq n$, $|C_i| \leq \text{SizeBound}_{\text{Head}_i} + 1$.
- Cluster-heads neighbor property: If $\text{Head}_j$ is in the neighborhood of $\text{Head}_i$, and if the weight of $\text{Head}_j$ is smaller than the weight of head, then $\text{Head}_j$ cannot belong to the $C_i$ cluster because the cluster $C_i$ is full, i.e., $\forall i, j \in [1, n], \text{Head}_j \notin N_{\text{Head}_i}, \forall w_{\text{Head}_j} \leq w_{\text{Head}_i}$, $\forall |C_i| = \text{SizeBound}_{\text{Head}_i} + 1$.

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 $\text{SizeBound}_u + 1$ nodes.

Notice that cluster-head neighbor property ensures that the set of cluster-heads $S$ selected by SG-BSC protocol is minimal: there is no 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 $\text{SizeBound}_v > 0$, 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 I); for instance, its hierarchical status ($\text{HS}_v$), the identity of its head ($\text{Head}_i$), and the weight of its head ($w_{\text{Head}_i}$).

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 ($\text{HS}_v = \text{CH}$), ordinary ($\text{HS}_v = \text{O}$) or nearly ordinary ($\text{HS}_v = \text{NO}$). The status transition diagram is shown in Figure 1 where transitions are the Election, Resignation, Affiliation and Correction actions defined in Protocol [2].
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 serminate 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 $CD_2.$ (for more details about the macro $CD_2$, see Section 3.3).

**Protocol 1 : Variables and macros on node $v$.**

**Parameters**

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

**Input variables**

$w_i, \in \mathbb{R}$ /* Weight of node $v$ */

**Local variables**

$HS_v, \in \{CH, O, NO\}$/* Hierarchical status of node $v$ */

$Head_v, \in \{v\}$/* Identity of $v$’s cluster-head */

$wh_v, \in \mathbb{R}$/* Weight of the head of $v$ */

$CD_v, \subseteq \{ID\}$/* List of nodes that can join $v$’s cluster */

$S_v, \in \mathbb{N}$/* Local value about the size of $v$’s cluster */

**Macros**

$Cluster_v = \{z \in N : Head_v = v\}$ /* 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 */

$N^*_v = \{z \in N, v \in CD_v \& HS_v = CH \& w_z > w_{Head_v} \land w_v = w_z\}$/* The $v$’s neighbors could be cluster-head of $v$ */

$BestHead_v = z \in N^*_v$ where $w_v = \max\{w_u \mid u \in N^*_v\}$/* The most suitable leader of $N^*_v$ */

$CD_2_v$/* Nodes authorized by $v$ to join $v$’s cluster */

**Begin**

$CD_0 := \{z \in N : wh_z < w_v \land w_z \neq w_v\}$

If $CD_0 \subseteq SizeBound, - Size_v$ then $CD_1 := CD_0$;

Else $CD_1$ contains the $SizeBound, - Size_v$ smallest members of $CD_0$;

If $CD_n \subseteq \{CD_1, \cup Cluster_v\}$ then $CD_2 := CD_1$;

Else $CD_2 := \emptyset$;

**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., $v \in CD_n$), 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 a node to become 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**

/* $v$ can be changed by its head */

$Change(v) \equiv (Head_v \notin N_v \cup \{v\}) \land (w_v > w_{Head_v}) \lor (HS_{Head_v} \neq CH \lor (HS_{Head_v} \neq CH \lor SizeBound_{Head_v}))$

/* The guard of Election rule */

$MustBecomeHead(v) \equiv (HS_v = O \\& \\& N^*_v = \emptyset) \lor (HS_v = NO \\& \\& N^*_v = \emptyset)$

/* The guard of Affiliation rule */

$MustAffiliate(v) \equiv (HS_v = O \\& \\& N^*_v \neq \emptyset) \lor (HS_v = NO \\& \\& Size_v = \emptyset \\& \\& N^*_v \neq \emptyset)$

/* The guard of Resignation rule */

$MustResign(v) \equiv (HS_v = CH) \lor (N^*_v \neq \emptyset)$

/* The guards of Correction rules */

$Corr-CH-g(v) \equiv (HS_v = CH) \land (Head_v \neq v \lor CD_v \neq CD_2 \lor S_v \neq Size_v \lor wh_v \neq w_v)$

$Corr-NO-g(v) \equiv (HS_v = NO) \land (Head_v \neq v \lor CD_v \neq CD_2 \lor S_v \neq Size_v \lor wh_v \neq w_v)$

$Corr-O-g(v) \equiv (HS_v = O) \land (CD_v \neq \emptyset \lor S_v \neq 0 \lor w_v \neq w_{Head_v})$

**Rules**

/* Clustering Construction rules */

$Election : MustBecomeHead(v) \rightarrow HS_v := CH; Head_v := v; CD_v := CD_2; S_v := Size_v; wh_v := w_v$

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

$Resignation : MustResign(v) \rightarrow HS_v := NO; Head_v := v; CD_v := \emptyset; S_v := 0; wh_v := w_v$

/* Correction rules */

$Corr-CH : MustResign(v) \land Corr-CH-g(v) \rightarrow Head_v := v; CD_v := CD_2; S_v := Size_v; wh_v := w_v$

$Corr-NO : MustBecomeHead(v) \land MustAffiliate(v) \land Corr-NO-g(v) \rightarrow Head_v := v; CD_v := \emptyset; S_v := 0; wh_v := w_v$

$Corr-O : MustBecomeHead(v) \land MustAffiliate(v) \land Corr-O-g(v) \rightarrow CD_v := \emptyset; S_v := 0; wh_v := w_{Head_v}$

**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., \( N^* \neq \emptyset \)), \( v \) has a neighbor that could be its new cluster-head. So, \( v \) affiliates to the node of \( N^* \) 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. \( N^* \neq \emptyset \). 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 \( \forall u \in N^* \), \( v \notin N^*_u \). 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 \( N^* \neq \emptyset \), \( v \) 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 \( \delta \) 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 \( SP \) is defined as follow:

- \( P_s(v) \equiv |Cluster_v \cup \bar{CD}_v| \leq SizeBound_v \)
- \( SP(v) \equiv (Head_v \in N_v \cup \{v\}) \land (|HS\ Head_v| \neq 0) \land P_s(v) \)
- \( SP \equiv \forall v, SP(v) = True \)

**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 belongs 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, \( Cluster_{\emptyset} = \{1\} \), and \( CD_{\emptyset} = \{2,3,4\} \). Thus, the Size property is satisfied, but the \( P_s(6) \) predicate is not satisfied: \( |CD_{\emptyset} \cup Cluster_{\emptyset}| = |\{1,2,3,4\} > 3 \). 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 \( P_s(v) \)](image_url)

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 \( CD_v \) is used to compute \( CD_v \) value. \( CD_v \) is computed in 3 steps. \( CD_0 \), 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 \( CD_0 \) can be greater than \( SizeBound_v - Size_v \). \( CD_1 \) is a subset of \( CD_0 \), containing at most \( SizeBound_v - Size_v \) nodes (having the smallest identifier). The set \( CD_2 \) is a subset of \( CD_1 \) 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 \( CD_2 \) is set to \( \emptyset \) whereas \( CD_1 \neq \emptyset \). In the initial configuration \( c_1 \), there are 5 clusters
satisfying the Size property, and $CD_6 = \{4, 5\}$. For simplicity, we assume the weight of a node is its identity. Thus, cluster-head 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 $Clusters$); so, $CD\{6\} = N - Cluster\{6\} = \{2, 3, 4, 5\}$. $CD_6$ contains only two nodes, because $Size\{Bound\} = 3$ (for every node) and $|Cluster\{6\}| = 1$; so, $CD_6 = \{2, 3\}$. In the reached configuration $c_2$, $CD_6(\{c\})$ must be set to 0 (so, $CD_6(\{c\}) = \emptyset$) because $CD_6(\{c\}) \not\subseteq |CD_6(\{c\}) \cup Cluster\{6\}|$. This will ensure that whatever the action done by nodes 4 and 5 -initially in $CD_6$- during the computation step $c_1 \rightarrow c_2$, $P_i(6)$ (6) is still verified in $c_2$: $|CD_6(\{c\}) \cup Cluster\{6\}| \leq Size\{Bound\}_6$.

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 $H\&T\&D$ 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 $Size\{Bound\}_6$, 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 $A_1$ is an attractor from $A_{-1}$ for $1 \leq i \leq 4$ (Let $A_{0}$ be $C$), where: $P_i(\{v\}) \equiv CD_6 = \emptyset$.

$A_1 = \{c \in C \mid \forall v \in V : P_i(\{v\}) \lor P_i(\{v\}) \text{ and } P_i(\{v\}) \text{ is satisfied}\}$.

$A_2 = \{c \in C \mid \forall v \in V : P_i(\{v\}) \text{ is satisfied}\}$.

$A_3 = \{c \in C \mid \forall v \in V : \text{Corr-CH}(v) \text{ is satisfied}\}$.

$A_4 = \{c \in C \mid \forall v \in V : (HS_v = O \land Head_v \in N_v \land HS_{Head_v} = O) \lor (HS_v \neq O \land S_v \leq Size\{Bound\}_6 \land Head_v = v)\}.

Observation 2. Let $v$ be a node, and $cs$ be a computation step: $c_1 \rightarrow c_2$.

$\bullet$ According to the macro $N^+$ and Affiliation rule:

$$Cluster\{c\} \subseteq \{Cluster\{c\} \cup CD\{c\}\}$$

(1)

$\bullet$ According to the macro $CD^2$:

$$CD\{c\} \subseteq \{Cluster\{c\} \cup CD\{c\}\}$$

(2)

where $CD\{c\} = \emptyset$ is satisfied in $c_2$.

Lemma 1. Let a computation step: $c_1 \rightarrow c_2$ in which a node $v$ performs an action. $P_i(\{v\}) \lor P_i(\{v\})$ is satisfied in $c_2$.

Proof: Any rule performed by $v$ during $cs$ updates $CD_6$ (see rule actions). If $CD_6(\{c\}) = \emptyset$, then $P_i(\{v\}) \lor P_i(\{v\})$ is satisfied in $c_2$. Assume that $CD_6(\{c\}) \neq \emptyset$. During $cs$, $CD_6$ is set to $CD_2(\{c\})$ and $CD_2(\{c\}) \neq \emptyset$. From Equation (1), we have:

$$CD\{c\} = CD\{c\} \land CD\{c\} \neq \emptyset$$

Since $CD\{c\}$ contains at most $Size\{Bound\}_6$, $|Cluster\{c\}|$ elements, then $|CD\{c\} \cup Cluster\{c\}| \leq Size\{Bound\}_6$. Thus, $|CD\{c\} \cup Cluster\{c\}| \leq Size\{Bound\}_6$.

Now, according to Equation (1), $|CD\{c\} \cup Cluster\{c\}| \leq |CD\{c\} \cup Cluster\{c\}| \cup Cluster\{c\}|$. From Equation (3) and our assumptions, we have: $|CD\{c\} \cup Cluster\{c\}| \leq |CD\{c\} \cup Cluster\{c\}|$ (because $CD\{c\} = CD\{c\}$). Finally, we conclude that $|CD\{c\} \cup Cluster\{c\}| \leq |CD\{c\} \cup Cluster\{c\}| \subseteq Size\{Bound\}_6$. Therefore, $P_i(\{v\}) \lor P_i(\{v\})$ is satisfied in $c_2$ when $CD\{c\} \neq \emptyset$.

Lemma 2. A2 is closed under any computation step.

Proof: Let $v$ be a node, and $c_1$ be a configuration in which $P_i(\{v\})$ is satisfied. Assume that a computation step $c_1 \rightarrow c_2$ exists, such that $P_i(\{v\})$ is not satisfied in $c_2$.

If $c_1$, $|CD\{c\} \cup Cluster\{c\}| \leq Size\{Bound\}_6$, and in $c_2$, $|CD\{c\} \cup Cluster\{c\}| > Size\{Bound\}_6$. So, there exists a node $z$ such that $z \notin |CD\{c\} \cup Cluster\{c\}|$ but $z \in |CD\{c\} \cup Cluster\{c\}|$. According to Equation (1), $z \notin Cluster\{c\}$. Thus, $v$ has changed the value of $CD\{c\}$ during $cs$ to include $z$: $CD\{c\} \neq \emptyset$. By Lemma (1) we have: $|CD\{c\} \cup Cluster\{c\}| \leq Size\{Bound\}_6$. There is a contradiction: $c_2$ does not exist and A2 is closed.

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

Proof: Let $v$ be a node, $c_1$ be a configuration satisfying $P_i(\{v\}) \lor P_i(\{v\})$, and $cs$ be a computation step $c_1 \rightarrow c_2$.

If $v$ performs an action during $cs$ then $P_i(\{v\}) \lor P_i(\{v\})$ is satisfied in $c_2$ (Lemma (1)). If $P_i(\{v\})$ is satisfied in $c_1$, then $P_i(\{v\})$ is satisfied in $c_2$ (A2 is closed, Lemma (3)).

Last case, $v$ does not perform any action during $cs$ and $P_i(\{v\})$ is satisfied in $c_1$; $P_i(\{v\})$ is still satisfied in $c_2$.

Lemma 3. If $P_i(\{v\}) \lor P_i(\{v\})$ is not satisfied in a configuration $c$, the node $v$ is enabled in $c$.

Proof: Let $c$ be a configuration in which $P_i(\{v\}) \lor P_i(\{v\})$ is not satisfied, i.e., $CD\{c\} \neq \emptyset \land |CD\{c\} \cup Cluster\{c\}| > Size\{Bound\}_6$. According to $v$’s hierarchical status in $c$, two cases are possible:

$\bullet$ Case 1 ($v$ is ordinary or nearly ordinary): $v$ is enabled in $c$, because Corr-O-g($v$) or Corr-N0-g($v$) is satisfied in $c$ ($CD\{c\} \neq \emptyset$).

$\bullet$ Case 2 ($v$ is cluster-head): According to the macros $CD\{c\}$, we have: $CD\{c\} = \emptyset \lor CD\{c\} = CD\{c\}$. If $CD\{c\} = \emptyset$ then $CD\{c\} \neq CD\{c\}$, Corr-CH-g($v$) is satisfied in $c$. Otherwise $CD\{c\} = CD\{c\} \neq \emptyset$. We have $0 < |CD\{c\}| \leq Size\{Bound\}_6 - |Cluster\{c\}|$ (by Equation 2). As $|CD\{c\}| > Size\{Bound\}_6 - |Cluster\{c\}|$ (by assumption), $CD\{c\} \neq CD\{c\}$. Corr-CH-g($v$) is satisfied in $c$.

Whatever the status of $v$, $v$ is enabled in $c$. 

8
Corollary 2. $A_1$ is an attractor from $C$ in one round.

**Proof:** A node $v$ that never satisfy $P_s(v) \lor P_t(v)$ during a computation $e$, is always enabled (Lemma 3). By fairness, $v$ will perform an action. After $v$’s move, $P_s(v) \lor P_t(v)$ is satisfied in $c_2$ (Lemma 4).

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 $P_s(v) \land \neg P_t(v)$, i.e., $CD_v(c_1) = 0 \lor [\text{Cluster}(c_1) > \text{SizeBound}_v]$. While $P_t(v)$ is not satisfied, $P_s(v)$ is verified ($A_1$ is closed, Lemma 1). So, in $c_1$, no node can join $v$’s cluster ($CD_v(c_1) = 0$).

Let $u$ be a node of $\text{Cluster}(c_1) (\forall v \land \text{Head}_u = v)$; so, $v \notin N'_v$ is forever satisfied (definition of $N'_v$). In $c_1$, the node $u$ is enabled whatever its status:
- $u$ is cluster-head: The guard $\text{Corr-CH-g}(u)$ is satisfied ($\text{Head}_u \neq u$). Thus, $u$ is enabled: $\text{Resignation}(u)$ or $\text{Corr-CH rule}$ is enabled.
- $u$ is nearly ordinary: The guard $\text{Corr-NO-g}(u)$ is satisfied ($\text{Head}_u \neq u$). Thus, $u$ is enabled: $\text{Election}(u)$, $\text{Affiliation}(u)$ or $\text{Corr-NO}(u)$ rule is enabled.
- $u$ is ordinary: If $S_{\text{Head}} \leq \text{SizeBound}_v$, then $S_v \neq \text{Size}_v$, and $v$ is enabled (the guard $\text{Corr-CH-g}(v)$ is verified). After $v$’s action, we have $S_{\text{Head}} = \text{Size}_v > \text{SizeBound}_v$; so, $\text{Change}(u)$ is satisfied. Therefore, $\text{MustBecomeHead}(u)$ or $\text{MustAffiliate}(u)$ is satisfied, and the rule $\text{Election}(u)$ or $\text{Affiliation}(u)$ is enabled.

The node $u$ stays enabled unless it performs an action. By fairness, $u$ eventually performs a rule, and gets $\text{Head}_u \in N'_v \cup \{u\}$ (see rules action), i.e., $\text{Head}_u \neq v$. This means that $u$ leaves $\text{Cluster}_u$. Thus, eventually a configuration of $A_2$ in which $|\text{Cluster}_u| \leq \text{SizeBound}_v$ 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 $S_v \leq \text{SizeBound}_v$, and $S_v > \text{SizeBound}_v$ update their variable $S_v$ to set $S_v = \text{Size}_v > \text{SizeBound}_v$. Thereafter, each node $u$ of $\text{Cluster}_u$ is enabled because it verifies the predicate $\text{Change}(u)$. 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 $\text{SizeBound}_v$; 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: $|\text{Cluster}_v| \leq |\text{CD}_v \cup \text{Cluster}_v| \leq \text{SizeBound}_v$. 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 $c_1 \rightarrow c_2$ exists where $c_2 \notin A_4$. Let $P_4a(v)$ be the predicate ($\text{HS}_v \neq O \land \text{Head}_v \in N_v$). Let $P_4b(v)$ be the predicate ($\text{HS}_v = O \lor (S_v \leq \text{SizeBound}_v \land \text{Head}_v = v$)). $P_4a(v)$ and $P_4b(v)$ are verified by $v$ in $c_1$. The value of $P_4a(v)$ and $P_4b(v)$ can be modified only by a $v$’s rule; and, after any $v$’s action from $c_1$, the predicates $P_4a(v)$ and $P_4b(v)$ still verified. So, $P_4a(v)$ and $P_4b(v)$ are also verified in $c_2$.

Thus, according to our assumption, there exists a node $v$ such that in $c_1$, ($\text{HS}_v = O \land \text{Head}_v \neq O$) but in $c_2$, ($\text{HS}_v = O \land \text{HS}_v = O$). Let $u$ be the $v$’s head in $c_1 (u \neq v)$. Only two actions require study: the execution of $\text{Affiliation}$ rule by $v$ or by $u$ during $c_s$ (to change $\text{Head}_v$ or to set $\text{HS}_v = O$). The node $u$ cannot perform $\text{Affiliation}$ rule during $c_s$, because in $c_1$, $S_{\text{Size}_u} \neq 0 (v \in \text{Cluster}_u)$. Assume that $v$ executes $\text{Affiliation}$ rule during $c_s$. Let $z$ be the cluster-head chosen by $v$ during $c_s$. In $c_1$, $\text{HS}_z = \text{CH} (z \in N'_v)$. During $c_s$, $z$ cannot perform $\text{Affiliation}$ rule (it can only do $\text{Resignation}$). In $c_2$, we have $\text{HS}_\text{Head} = O$.

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

**Proof:** Let $v$ be a node, and be a configuration of $A_3$. In $c$, $S_v \leq \text{SizeBound}_v$. So, in $c$ if $S_v > \text{SizeBound}_v$, then $S_v \neq \text{Size}_v$. Assume that $c$ does not belong to $A_4$. In $c$, $v$ has one of the following states:
- $\text{HS}_v = \text{CH} \land \text{Head}_v \neq v \land S_v > \text{SizeBound}_v$. The guard $\text{Corr-CH-g}(v)$ is satisfied. By fairness, $v$ will perform the $\text{Resignation}$ or $\text{Corr-CH}$, and after that $\text{HS}_v = O \land \text{Head}_v = v \land S_v \leq \text{SizeBound}_v$ holds.
- $\text{HS}_v = O \land \text{Head}_v \notin N_v \land \text{HS}_\text{Head} = O$: $\text{Change}(v)$ is satisfied; so, the guard $\text{MustBecomeHead}(v)$ or $\text{MustAffiliate}(v)$ is verified. As all computations are fair, $v$ eventually performs $\text{Election}$ or $\text{Affiliation}$ rule, and after that ($\text{HS}_v = \text{CH} \land \text{Head}_v = v \land S_v \leq \text{SizeBound}_v$) or ($\text{HS}_v = O \land \text{Head}_v \in N_v \land \text{HS}_\text{Head} \neq O$) is verified.
- $\text{HS}_v = \text{NO} \land \text{Head}_v \neq v \lor S_v > \text{SizeBound}_v$. The guard $\text{Corr-NO-g}(v)$ is satisfied. By fairness, $v$ will perform $\text{Election}$, $\text{Affiliation}$ or $\text{Corr-NO}$ rule. Upon performing $\text{Election}$ or $\text{Corr-NO}$ rule, $\text{HS}_v \neq O \land \text{Head}_v = v \land S_v \leq \text{SizeBound}_v$, holds. The execution of $\text{Affiliation}$ rule makes $\text{HS}_v = O \land \text{Head}_v \in N_v \land \text{HS}_\text{Head} = \text{CH}$ 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: $\text{Head}_v \neq v$ if and only if $\text{HS}_v = O$.

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

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

Definition 7 (Highly tolerated disruptions). The set of highly tolerated disruptions $\text{HTD}$ 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 SP.

**Lemma 8.** $SP_v$ is closed under the set of highly tolerated disruptions $HTD$.

**Proof:** Let $z$ be the head of a node $v$ ($z$ is a cluster-head or nearly ordinary). The safety predicate $SP_v$ 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.$z$.

$SP_v$ 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 $SP$ is preserved under any disruption of $HTD$.

**Theorem 1.** The protocol SG–BSC has a service guarantee despite the occurrence of $HTD$ 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 $A_k$, the safety predicate $SP$ is satisfied; so, any configuration of $A_k$ 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 $V_i = V - S et_i$ having the highest weight, named $v_h$, becomes cluster-head. The cluster of $v_h$ is filled out, and members of $v_h$’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 $S et_i$ have chosen their cluster-head and, they reached their final state (they have stabilized). The system is stabilized when $S et_i = V$. We define sets $L_i$ and $S et_i$ as follows:

**Notation 2.** $S et_0 = \emptyset$; $L_0 = A_1$; $V_i = V - S et_i$; Set of nodes that do not belong to $S et_i$.

Let $v_h$ be the node having the highest weight in $V_i$:

$L_{i+1} = L_i^\ast \cap \{c \in C | HS_{v_h} = CH \}$;

$Bound_i = Min(SizeBound_{v_h}, |N_{v_h} \cap V_i|)$;

$L_{i+1} = L_i \cap \{c \in C | \{Clusterv_{v_h} \} \cap Bound \}$;

$S et_{i+1} = S et_i \cup \{v_h\} \cup Clusterv_{v_h}$;

$L_{i+1} = L_i \cap \{c \in C | \forall v \in S et_i : (CD_v = \emptyset) \land (HS_v \neq O \land S_v = Size\_v) \lor (HS_v = O \land S_v = 0)\};$

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.

- $S et_0$ is the set of nodes having initially their terminal state. Usually, $S et_0 = \emptyset$.
- We name $v_{h0}$ the node having the highest weight in $V_0 = V - S et_0$. In Figure 4, $v_{h0} = 9$. 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 $L_1 = A_1 \cap \{c \in C | HS_{v_{h0}} = CH\}$ 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 $L_{i+1} = L_i \cap \{c \in C | Min(SizeBound_{v_{h0}}, |N_{v_{h0}} \cap V_i|) = |Cluster_{v_{h0}}|\}$ be the set of configurations where $v_{h0}$’s cluster is stable (no node will quit or join this cluster). Configuration (e) belongs to $L_{i+1}$.
- In the last step of phase 1, all nodes of $v_{h0}$’s cluster reach their terminal state (a configuration of $L_{i+1}$). The configuration (f) belongs to $L_{i+1}$, where $L_{i+1} = L_i \cap \{c \in C | \forall v \in S et_0 \cup \{v_{h0}\} \cup \{Cluster_{v_{h0}}\}; CD_v = \emptyset \}$.

At the end of first phase, set of nodes having their final state is $S et_1 = S et_0 \cup \{v_{h0}\} \cup \{Cluster_{v_{h0}}\}$. Each phase $i$ is similar to the first one: the node of $V_i = V - S et_i$ having the highest weight, named $v_h$, becomes cluster-head. The cluster of $v_h$ is filled out, and members of $v_h$’s cluster get their final state. At the end of $i^{th}$ phase, the set of nodes having their final state is $S et_i = S et_{i-1} \cup \{v_h\} \cup \{Cluster_{v_h}\}$. 

10
5.2. Proof of stabilization

We will prove that configurations set $L'_i$ is an attractor from C such that at the $f^i$ phase $S_{f^i} = V$.

Observation 4. In each phase $i$, the set $S_{f^i}$ increases up to contain all nodes.

If $(S_{f^i} \neq V)$ then $S_{f^i} \subseteq S_{f^{i+1}} \wedge S_{f^i} \neq S_{f^{i+1}}$ (4)

If $(v \in S_{f^i})$ then $Head_v \in S_{f^i}$ (5)

If $(v \in V_i)$ then $Head_v \notin S_{f^i}$ (6)

For any value of $i$, any configuration of $L_{i}$, $L'_{i}$, or $L''_{i}$ belongs to $A_d$, because $L_i$, $L'_i$, and $L''_i$ are subsets of $L'_0 = A_d$.

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 $L''_i$. In $c$ we have: for all $u \in N_{i+1}$, if $u \in V_i$, then $w_{i+1} > w_u$ (by definition of $v_{i+1}$); else (i.e., $u \in S_{f^i}$), $CD_u = \emptyset$. So, $N^*_u$ is empty forever. In $c$, there are two possibilities:

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

- $v_h$ is ordinary: from Observation 4, $Head_v \neq v_h$. According to Equation 6 and to the definition of $v_h$, we have: $w_{i+1} > w_{Head_v}$. So, $MustBecomeHead(v_h)$ is forever satisfied. By fairness, $v_h$ eventually performs the Election rule. After that we get: $HS_{v_h} = CH$.

In both cases $L_{i+1}$ is reached from $L''_i$. Furthermore, the value of $HS_{v_h}$ is never modified, because the rule $Resignation(v_h)$ is never enabled ($N^*_u = \emptyset$ 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 $c_1 \rightarrow c_2$, we have: $Cluster_{vh}(c_1) \subseteq Cluster_{vh}(c_2)$.

Proof: Let $u$ be a node of $Cluster_{vh}$ ($Head_u = v_h$). In $c_1$, $HS_u = O$ (Observations 4); and $\forall z \in N_u$, if $z \in V_i$ then $w_{Head_z} > w_z$; else (i.e., $z \in S_{f^i}$), $CD_z = \emptyset$. 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: $S_{Head_u} \leq Size_{Bound}_{Head_u}$ (Lemma 7), $v_h \in N_u$ (Lemma 7), $w_{Head_v} > w_u$ (by definition of $v_h$), and $HS_{v_h} = CH$ (Lemma 9). Thus, the guard $MustBecomeHead(u)$ is never verified and $u$ cannot become a cluster-head. According to that, once $L_{i+1}$ is reached, no node can leave $Cluster_{vh}$.

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 $L_{i+1}$, $HS_{v_h} = CH$. Let $u$ be a node of $\{N_{i+1} \cap V_i\}$. Thus, there exists an integer $j$, ($0 < j < i$), such that: $(u = v_h) \vee (u \in Cluster_{vh})$.

In $L_{j+1}$, $u$ can never leave $Cluster_{vh}$ (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.e., $Cluster_{vh} \subseteq \{N_{i+1} \cap V_i\}$. In $L_{j+1}$, $|Cluster_{vh}| \leq Size_{Bound}_{vh}$ (Observation 4), thus $|Cluster_{vh}| \leq Bound_i$ is forever verified.

Once $|Cluster_{vh}| = Bound_i$, $Cluster_{vh}$ is never modified (no node leaves $Cluster_{vh}$, see Lemma 10). Assume that $|Cluster_{vh}| < Bound_i$. $CD_{1vh} \neq \emptyset$ because $CD_{1vh}$ contains $Bound_i - |Cluster_{vh}|$ elements of $\{N_{i+1} \cap V_i\} - Cluster_{vh}$.

While $CD_{1vh} \neq CD_{2vh}$, the node $v_h$ is enabled (Corr-CH-g($v_h$) is verified). By fairness, $v_h$ performs Corr-CH rule, and gets $CD_{1vh} = CD_{1vh}$ or $CD_{1vh} = \emptyset$ (now, $CD_{1vh} \subseteq \{CD_{1vh} \cup Cluster_{vh}\}$). In the last case, Corr-CH-g($v_h$) is still verified till $v_h$ performs again the Corr-CH rule. After that we have: $CD_{1vh} = CD_{1vh} \neq \emptyset$. Let $v$ be a node of $CD_{1vh}$. We have $v_h \in N^*_v$: $N^*_v \neq \emptyset$. $v$ 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($z$) ($HS_{v_h} = CH$), and are enabled because $MustBecomeHead(z)$ or $MustAffiliate(z)$ is satisfied. After $z$’s action, $Head_v = v_h$; 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 $v_h$ has a higher weight than nodes of $N^*_v$, because $\forall z \in N_v$, if $w_z > w_{Head_z}$ then $z \in S_{f^i}$ and $z \notin N^*_v$ ($CD_0 = \emptyset$). By fairness $v$ performs the Affiliation rule, and it chooses $v_h$ as cluster-head. Eventually, when each node $v$ of $CD_{1vh}$ performs Affiliation rule, a configuration where $|Cluster_{vh}| = Bound_i$ 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, $v_h$ sets $CD_{1vh}$ to $CD_{2vh}$. During the third round, every cluster-head $v \in CD_{1vh}$ is enabled and it performs the resignation rule to become nearly ordinary. At the end of this round, each node $v$ of $CD_{1vh}$ 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 $CD_{1vh}$ perform the Affiliation rule to join the cluster of $v_h$.

Lemma 12. In $L'_{i+1}$, for any node $v$ of $Set_{f^i}$, $HS_v \neq NO$ and the Election, Affiliation, Resignation and Corr-NO rules are disabled.
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 2 there exists $j$ such that $Set_j = V$. Since $L_0 = A_2$ is an attractor from $C$ (Lemma 7), then for any value of $i$, the sets $L_i, L_j, L''_j$ 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 5).
- $CD_v = 0$ for any node $v$ (By definition of $L''_j$).
- If $v$ is a cluster-head then $Head_v = v$ (Observation 1), $CD_2 = 0$ (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-CH.

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 $c$, $CD_v = 0$ for any node $v$, and $v$ is not a nearly ordinary node ($L_v \subseteq L''_j$ and Lemma 12). In $c$, two cases are possible:

- **v is ordinary:** The guard MustAffiliate($v$) is unsatisfied in $c$. Thus, $Head_v \in N_c \land HS_{Head_v} = CH \land w_{Head_v} > w_v$. Therefore, the Affiliation property is fulfilled in $c$.
- **v is cluster-head:** In $A_3$, $Cluster_v \subseteq SizeBound_v$. 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 $z \in N_c$ such that: $HS_z = CH \land w_z < w_v \land Size_z < SizeBound_z$, i.e., the Cluster-head neighbor property is not satisfied. The node $v$ belongs to $CD_0$ because $w_{Head_v} = w_v < w_z$. Moreover, $CD_1 \neq 0$ because $Size_v < SizeBound_v$. Since, in the configuration $c$ $CD_2 = 0$; then according to the macro $CD_2, CD_2 \neq 0$ in $c$. Thus, Corr-CH($v$) is satisfied ($CD_v \neq 2CD_2$). The configuration $c$ is not terminal; there is a contradiction. We conclude that, in $c$, for any cluster-head $z \in N_c$ ($w_z > w_v$) $\land$ ($Size_v < SizeBound_v$), thus, the Cluster-head neighbor property is verified in $c$.

5.4. Stabilization time

The set $L_{r+1}$ is reached from $L''_j$ in at most one round (Corollary 6). The set $L_{r+1}$ is reached from $L_{r+1}$ 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 $L''_{r+1}$ and $L''_{r+1}$ are reached upon $L_{r+1}$ is reached (no additional rounds are required). Thus, $L''_{r+1}$ is reached from $L''_j$ in at most one round. So, in the worst case: (1) no cluster is empty: seven rounds are necessary to reach $L''_{r+1}$ from $L''_j$, for $0 < i < j$, and (2) each cluster contains exactly one ordinary node ($\lceil \frac{w}{v} \rceil$ clusters).

We conclude that, in the worst case, the configurations set $L''_{r+1}$ is reached from $L''_j$ after seven rounds, and $L''_j$ is reached from $L_0$ after $7 + j$ rounds, where $j$ is the number of built clusters $j = \lceil \frac{w}{v} \rceil$. 

Proof: Let $v$ be a node of $Set_{r+1}$. There exists an integer $j$, $0 < j \leq i$, such that $(v \in Set_{r+1}) \land (v \notin Set_j)$. So, $(v = vh_j) \lor (v \in Cluster_{vh_i})$.

In a configuration of $L_{r+1}$, if $(v = vh_j)$ then $v$ is a cluster-head (by definition of $L_{r+1}$ and Lemma 9), else (i.e. $v \in Cluster_{vh_i}$) $v$ is ordinary node (Observation 3).

Thus, in $L_{r+1}$ 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 $L''_j$ 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** ($v \in Cluster_{vh_i}$): According to definition of $L''_j$ and Lemma 10, $v$ can never execute the Election or Affiliation rule and leave the cluster. Resignation rule is also disabled for $v$ (see protocol rules).

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

Lemma 13. In $L''_{r+1}$, $CD_{2i} = 0$.

Proof: In $L''_{r+1}$, we have $|Cluster_{vh_i}| = Bound_i$ (Lemma 11). If $|Cluster_{vh_i}| = Bound_i$, then $CD_{2i} = 0$. 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 $w_{Head_v} \geq w_{vh_i}$. So, $CD_{2i} = 0$. Therefore, in $L''_{r+1}$, $CD_{2i} = 0$.

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

Proof: Let $v$ be a node of $Set_{r+1}$ but not of $Set_{r}$. From Lemma 12, $HS_v \neq NO$: thus two cases are possible:

- **v is ordinary:** If $CD_v \neq 0$ or $S_v \neq 0$, then $v$ is enabled (Corr-0-g($v$) is verified). By fairness, the node $v$ eventually performs Corr-0 rule. After that, $CD_v = 0 \land S_v = 0$ holds forever.
- **v is cluster-head:** By definition, $v$ is $vh_j$. In $L''_{r+1}$, we have $CD_{2i} = 0$ (Lemma 15). If $CD_{2i} \neq 0$ (i.e., $CD_{2i} \neq CD_{2i}$ or $S_v \neq Size_v$), then Corr-CH-g($v$) is verified. By fairness, $vh_j$ will perform Corr-CH rule. After that, $CD_{2i} = 0 \land S_v = Size_v$ holds forever.

Corollary 7. The configurations set $L''_{r+1}$ is reached from $L''_{r+1}$ 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 2 there exists $j$ such that $Set_j = V$. Since $L_0 = A_2$ is an attractor from $C$ (Lemma 7), then for any value of $i$, the sets $L_i, L_j, L''_j$ 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).
- $CD_v = 0$ for any node $v$ (By definition of $L''_j$).
- If $v$ is a cluster-head then $Head_v = v$ (Observation 1), $CD_2 = 0$ (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-CH.

In a configuration of $L''_j$, $v$ can never perform an action. Thus, a terminal configuration is reached.
The time to reach \( L_0^1 (A_2) \) is 4 rounds (see Section 4). Therefore, along any computation and from any initial configuration, the legitimate configuration is reached in at most \( \frac{7|\mathcal{X}|}{2} + 4 \) 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.

![Diagram](image)

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.

**Table 1: Default value of simulation parameters.**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulation time</td>
<td>100s</td>
</tr>
<tr>
<td>Number of nodes</td>
<td>70</td>
</tr>
<tr>
<td>Transmission range</td>
<td>250m</td>
</tr>
<tr>
<td>Network area</td>
<td>1200m*1200m</td>
</tr>
<tr>
<td>Speed</td>
<td>0m/s - 12m/s</td>
</tr>
<tr>
<td>Pause time</td>
<td>0.5s</td>
</tr>
<tr>
<td>( \omega_{\text{min}} )</td>
<td>50</td>
</tr>
<tr>
<td>( \omega_{\text{max}} )</td>
<td>80</td>
</tr>
<tr>
<td>( \Delta )</td>
<td>2</td>
</tr>
<tr>
<td>SizeBound</td>
<td>10</td>
</tr>
<tr>
<td>( \text{freq} )</td>
<td>2c/s</td>
</tr>
</tbody>
</table>

**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 \( \omega_{\text{min}} \) and \( \omega_{\text{max}} \). The node’s weight changes according to a frequency \( \text{freq} \), 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 \( \Delta \). 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 interesting 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 \( \text{freq} = 2 \) 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 \( \text{freq} = 2c/s \), 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 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 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 guarantees 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 \( HTD \) (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.
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 \( \frac{7}{2}N + 5 \) 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


