From self- to self-stabilizing with service guarantee 1-hop weight-based clustering *

Colette Johnen and Fouzi Mekhaldi

LaBRI, University of Bordeaux, CNRS. F-33405 Talence Cedex, France

Abstract. We propose a transformer building a silent self-stabilizing with service guarantee 1-hop clustering protocol $\mathcal{TP}$ of an input silent self-stabilizing 1-hop clustering protocol $\mathcal{P}$. From an arbitrary configuration, $\mathcal{TP}$ reaches a safe configuration in at most 3 rounds, where the following useful minimal service is provided: "each node belongs to a 1-hop cluster having an effective leader". During stabilization of $\mathcal{TP}$, the minimal service is preserved, so the clustering structure is available throughout the entire network. The minimal service is also maintained despite the occurrences of some external disruptions, called *highly tolerated disruptions*, denoted $\mathcal{HTD}$. $\mathcal{TP}$ reaches a terminal (also legitimate) configuration in at most $4 * S_{\mathcal{P}}$ rounds where $S_{\mathcal{P}}$ is the stabilization time of $\mathcal{P}$ protocol. Moreover, $\mathcal{TP}$ requires only 2 bits per node more than $\mathcal{P}$.

1 Introduction

Self-stabilization has a major limitation: during stabilization periods, a self-stabilizing protocol does not guarantee any property (except the eventual convergence) even if perturbations could be handled in a safe manner. Thus, self-stabilization is suited for distributed systems with intermittent disruptions, where the delay between successive disruptions is so large that the system can recover to a legitimate configuration providing its optimum service for some time. However, in large scale dynamic networks, the network topology changes very often, and the paradigm of self-stabilization is no more satisfying. Indeed, the system may be continuously disrupted, causing a total loss of service. As consequence, the availability and reliability of self-stabilizing systems are compromised when disruptions are frequent. To overcome these drawbacks, the paradigm *self-stabilization with service guarantee* has been recently introduced in [17,21,18].
A protocol $\mathcal{P}$ is self-stabilizing with service guarantee if: (1) $\mathcal{P}$ is self-stabilizing; (2) from an arbitrary configuration, $\mathcal{P}$ quickly reaches a safe configuration, where a safety property is satisfied, so a minimal service is provided; (3) the safety property (minimal service) holds during progress of $\mathcal{P}$ towards the optimum service (i.e., during stabilization despite actions of $\mathcal{P}$) and, (4) the safety property (minimal service) is also maintained despite the occurrences of some specific external disruptions, called *highly tolerated disruptions*, denoted $\mathcal{HTD}$.
Whatever the occurrences of $\mathcal{HTD}$ disruptions, the useful minimal service is still provided. Whereas, other disruptions are handled by self-stabilization, i.e.,

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

after their occurrences, the system may behave arbitrarily, but it will quickly reach a safe configuration. Therefore, the service guarantee property is provided through both: fast recovering to the minimal service, and preservation of the minimal service despite the occurrences of $\mathcal{HTD}$ disruptions.

Clustering. This work addresses the transformation of a silent self-stabilizing 1-hop weight-based clustering protocol to a self-stabilizing with service guarantee one. The clustering of networks consists of partitioning network nodes into non-overlapping groups called clusters. Each cluster has a single head, called leader, that acts as local coordinator of the cluster, and eventually a set of standard nodes. In 1-hop clusters, the standard nodes are neighbor (at distance 1) of their leader. Clustering is found very attractive in infrastructure-less networks, like ad-hoc networks, since it limits the responsibility of network management only to leaders, and it allows the use of hierarchical routing. This is why numerous clustering protocols were proposed in the literature [1,2,6,11,15,17,20,21,22,24].

When the clustering is weight-based, each node of the network has a weight value that can change during time. The weight value represents the capability of nodes to be leaders. Hence, in weight-based clustering protocols, leaders are chosen according to their weight value in order to be the most suitable nodes in their clusters. Protocols proposed in [1,6,16,17,20,21] are weight-based.

Related works. Self-stabilization with service guarantee is related to snap-stabilization [4], safe convergence [22] and super-stabilization [13]. The common goal of these approaches is to provide a desired safety property during the convergence phase, after the occurrence of one or several well defined events.
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* [10] (not *system-centric* as in safe convergence, super-stabilization and self-stabilization with service guarantee approaches). It ensures that the answer to a properly initiated request by the protocol is correct. This approach is thus suited for service-oriented protocols, but not to silent protocols like clustering protocols. The *safe convergence* ensures that (1) the system quickly converges to a safe configuration, and (2) the safety property stays satisfied during the stabilization under protocol actions. However, external disruptions are not handled in safe convergence. Let us study the self-stabilizing with service guarantee protocol [18] building the knowledge of 1-hop neighbor clusters. The stabilization time of this protocol is 4 rounds as the time to reach a safe configuration. In this case, the safe convergence contributes nothing compared to the self-stabilization (they become equivalents). The main specifity of [18] is the maintain of safety property in spite of disruptions made by 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 specific 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 stabilization phase. Self-stabilization with service guarantee provides and maintains the safety property even before stabilization, unlike super-stabilization. For example, the super-stabilizing coloring algorithm [13]

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, super-stabilizing protocol handles them as a self-stabilizing protocol.

Some transformers related to previous approaches were proposed. In [23], the proposed protocol transforms almost all non self-stabilizing protocols to self-stabilizing one. The method proposed in [8] transforms a self-stabilizing wave protocol with a unique initiator to a snap-stabilizing one. In [7], authors propose a snap-stabilizing version of four fundamental protocols: reset, snapshot, leader election, termination detection, based on a snap-stabilizing PIF (Propagation of Information with Feedback) algorithm. Thereafter, they propose a method to provide a snap-stabilizing version of any protocol. In [3], the proposed method transforms a self-stabilizing protocol constructing spanning tree and optimizing any arbitrary tree metric to a loop-free super-stabilizing protocol.

Motivation and Contributions. The stabilization time of weight-based clustering protocols is proportional to the network diameter [21]. Nevertheless, a crucial challenge of ad-hoc networks is the fast establishment and maintenance of clustering structure in spite of topological changes like node/link failures.

In this paper, we propose a generic scheme to transform a silent self-stabilizing 1-hop weight-based clustering protocol $\mathcal{P}$, to a silent self-stabilizing with service guarantee protocol, called transformed protocol $\mathcal{TP}$. $\mathcal{TP}$ quickly reaches, in at most 3 rounds, a safe configuration from any initial one, and thereafter it reaches a terminal configuration in at most $4 * S_{\mathcal{P}}$ rounds where $S_{\mathcal{P}}$ is stabilization time of $\mathcal{P}$ protocol. In a safe configuration, each standard node belongs to a cluster, and each cluster has an effectual leader; so the clustering structure is available throughout the entire network. This safety property holds during stabilization phases even despite the occurrence of $\mathcal{HTD}$ disruptions (Definition 5). Moreover, compared to $\mathcal{P}$ protocol, $\mathcal{TP}$ requires only 2 extra bits per node.

Paper outline. The rest of the paper is organised as follows. In section 2, communication and computation models are defined, and the general form of original protocol $\mathcal{P}$ is described. Transformed protocol $\mathcal{TP}$ is presented in section 3. In sections 4 , 5 and 6, we give the sketch proof of service guarantee, correctness and termination of $\mathcal{TP}$ protocol. Finally, in section 7, the memory space and time complexity of $\mathcal{TP}$ protocol as well as the futur works are discussed.

2 Model and Concepts

A distributed system S is an undirected graph $G = (V, E)$ where vertex set V is the set of (mobile) nodes and edge set E is the set of communication links. A link $(u, v) \in E$ if and only if u and v can directly communicate (links are bidirectional); so, u and v are neighbors. We note by N_v the set of v's neighbors: $N_v = \{u \in V \mid (u, v) \in E\}$. Furthermore, every node v in the network is assigned a unique identifier, and a weight value w_v (a real number). The weight value of a

node can increase or decrease during time reflecting changes in the node's state. 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*).

We use the *local shared memory model* introduced in [12]. Each node v maintains a set of local variables such that v can read its own variables and those of its neighbors, but it can modify only its variables. 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 is a set of *rules*. Each rule has the form: $Rule_i :< Guard_i > \longrightarrow < Action_i >$. The *guard* of a v's rule is a Boolean expression involving the state of the node v, and those of its neighbors. The *action* of a v's rule updates 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.

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 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 $\mathcal{C}$ the set of all possible configurations, and by $\mathcal{E}$ the set of all weakly fair computations. The set of weakly fair computations starting from a particular configuration $c \in \mathcal{C}$ 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 $A \subset \mathcal{C}$.

We say that a node v is *neutralized* during a computation step cs $c_i \rightarrow c_{i+1}$, if v is enabled in c_i and disabled in c_{i+1}, but it did not execute any action during cs. The neutralization of a node v happens when one v's neighbor changes its state during cs, and after this change, the guard of all v's actions are not verified.

We use the *round* notion to measure the time complexity. The first round of a computation $e = c_1, ..., c_j, ...$ is the minimal prefix $e_1 = c_1, ..., c_j$, 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.

Definition 1 (Attractor). *Let B_1 and B_2 be subsets of $\mathcal{C}$. 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 $\mathcal{L} \subseteq \mathcal{C}$, called set of legitimate configurations, such that the following conditions hold:*

- *$\mathcal{L}$ is an attractor from $\mathcal{C}$.*

- *Configurations of $\mathcal{L}$ match the specification problem.*

A self-stabilizing protocol is *silent* if once the system is stabilized, no node modifies its state.

Stabilization time. The stabilization time is the number of disjoint rounds of a computation reaching a legitimate configuration from any initial one.

Definition 3 (Self-stabilization with service guarantee). *Let $\mathcal{SP}$ be the safety predicate that stipulates the minimal service (safety property), and $\mathcal{HTD}$ be the set of highly tolerated disruptions. A self-stabilizing system has service guarantee despite $\mathcal{HTD}$ if and only if the set of configurations satisfying $\mathcal{SP}$ is:*

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

2.1 The original protocol $\mathcal{P}$

We are placing in the context of clustering protocols where nodes proclaim themselves leaders like [1,2,6,11,15,17,20,21,24], and not in the context of protocols where leaders are nominated by other nodes like [5,9].

The general form of the original silent self-stabilizing weight-based 1-hop clustering protocol $\mathcal{P}$ is described in Protocol 1. Such protocol has four class of rules. The Election, Affiliation and Resignation rules for a node v update at least the head identity of v's cluster (i.e., $\texttt{Head}(v)$). Whereas the Complementary rules (named $Complement(v)$) update other variables if there exist.

Protocol 1 : The original protocol $\mathcal{P}$ on node v.

Output variables

- $\texttt{Head}(v) \in N_v \cup \{v\}$; $\texttt{Head}(v)$ returns the head's identity of the v's cluster.
- $\texttt{NextHead}(v) \in N_v \cup \{v\}$; $\texttt{NextHead}(v)$ returns the identity of head that will be chosen by the affiliation or resignation rule if it is enabled. It return v if the Election rule is enabled. Otherwise, it returns $\texttt{Head}(v)$.

Rules

Election(v) : $\texttt{GE}(v) \longrightarrow \texttt{AE}(v)$; The election rule
Affiliation(v) : $\texttt{GA}(v) \longrightarrow \texttt{AA}(v)$; The affiliation rule
Resignation(v) : $\texttt{GR}(v) \longrightarrow \texttt{AR}(v)$; The resignation rule
Complement(v) : $\texttt{GC}(v) \longrightarrow \texttt{AC}(v)$; Complementary rules if there exist

The variable $\texttt{Head}(v)$ indicates the identity of v's head and, whether v is a leader (i.e., $\texttt{Head}(v) = v$) or v is a standard node (i.e., $\texttt{Head}(v) \neq v$).

Note that rules of $\mathcal{P}$ protocol are not necessarily explicitly written in this form, but they can be distinguished according to how they update $\texttt{Head}(v)$ variable. Any rule does not updating $\texttt{Head}(v)$ is classified as Complementary rule. Election rule is enabled only by standard nodes verifying the election guard GE (1^{st} Precondition). Upon execution of Election rule, the standard node becomes leader. Conversely, Resignation rule is enabled only by leaders verifying the resignation guard GR (2^{nd} Precondition), and after execution of Resignation rule, the leader chooses a new head and it becomes a standard node. Nodes having

Affiliation rule enabled are standard nodes verifying the affiliation guard $\texttt{GA}$ (3^{rd} Precondition). By performing this rule, the standard node changes its cluster. Both actions $\texttt{AE}$, $\texttt{AR}$ and $\texttt{AA}$ are called *clustering actions* because they modify $\texttt{Head}(v)$ and they set it to $\texttt{NextHead}(v)$. When Election rule is enabled, then $\texttt{NextHead}(v) = v$ (1^{st} Precondition). If Resignation or Affiliation rule is enabled, then $\texttt{NextHead}(v) \neq v$ (2^{nd} and 3^{rd} Preconditions), $\texttt{NextHead}(v) \neq \texttt{Head}(v)$ and $\texttt{NextHead}(v)$ is currently leader (4^{th} Precondition).

$\mathcal{P}$ is weight-based clustering protocol. In weight-based clustering protocols, each node v has a dynamic input value, its weight named w_v, representing its suitability to be leader. Such protocols select nodes having a higher weight to be leader, and try as soon as possible to assign standard nodes to the best leader in their neighborhood. Thus, the value of $\texttt{NextHead}$ in such protocols depends intrinsically on the weight of nodes (see 5^{th} Precondition on $\mathcal{P}$).

The fact that $\mathcal{P}$ is self-stabilizing and weight-based, is summarized by the following Preconditions 1-5, whereas Preconditions 6-7 are consequences of silence property of $\mathcal{P}$. The formal description of these preconditions in follows facilitates the proof of service guarantee, correctness and termination of $\mathcal{TP}$ protocol.

1. $\texttt{GE}(v) \Rightarrow \texttt{Head}(v) \neq v \wedge \texttt{NextHead}(v) = v$
2. $\texttt{GR}(v) \Rightarrow \texttt{Head}(v) = v \wedge \texttt{NextHead}(v) \neq v$
3. $\texttt{GA}(v) \Rightarrow \texttt{Head}(v) \neq v \wedge \texttt{NextHead}(v) \neq v$
4. $\texttt{GA}(v) \vee \texttt{GR}(v) \Rightarrow$
$$\texttt{NextHead}(v) \neq \texttt{Head}(v) \wedge \texttt{Head}(\texttt{NextHead}(v)) = \texttt{NextHead}(v) \quad (1)$$
5. The function updating $\texttt{NextHead}$ is based on node's weight,
$$(\texttt{NextHead}(v) \neq \texttt{Head}(v)) \Rightarrow (\texttt{NextHead}(v) = v) \vee (w_{\texttt{NextHead}(v)} > w_v) \quad (2)$$
6. Along a computation where a standard node v never changes of cluster (so, its $\texttt{Head}(v)$ value), v performs a finite number of time Complementary rules.
7. Along a computation where the cluster of a leader v does not change, v performs a finite number of time Complementary rules.

3 The transformed protocol $\mathcal{TP}$

During stabilization of $\mathcal{P}$ protocol, a node may not belong to a cluster. One goal of $\mathcal{TP}$ protocol is to avoid such situation: once a node is in a cluster, it will belong to a cluster having an effectual leader during all stabilization period despite the occurrence of $\mathcal{HTD}$ events. The main idea of transformation is to control the execution of $\mathcal{P}$ protocol by changing/adding some rules in order: (1) to form temporary clusters, (2) to delay actions by making cluster-heads resign only after their clusters become empty, and by avoiding standard nodes to affiliate with a currently resigning leader. Moreover, this transformation modifies the execution of $\mathcal{P}$ since it forces some nodes to become leaders although the Election rule is disabled in $\mathcal{P}$ protocol. This forced election does not impact the final clusters

produced by $\mathcal{TP}$ protocol compared to final clusters of $\mathcal{P}$ (see Correction proofs, Sec 5). Transformed protocol $\mathcal{TP}$ is described in Protocols 2 and 3.

Protocol 2 : Variables and predicates of the Transformed Protocol $\mathcal{TP}$.

Output variables

• $\texttt{Status}_v \in \{CH, O, NO, NCH\}$; Hierarchical status of node v. It can be Cluster-head (CH), Ordinary (O), Nearly Ordinary (NO) and Nearly Cluster-head (NCH).

Input variables

• $\texttt{Ready}_v \in \{RO, RCH\}$; It indicates if v is ready to become cluster-head ($\texttt{Ready}_v = RCH$) or ordinary ($\texttt{Ready}_v = RO$).

Predicates

• $\texttt{Is\_Leader}(v) \in \{T, F\}$; It indicates if v is a leader or a standard node.
If $\texttt{Head}(v) = v$ then v is leader ($\texttt{Is\_Leader}(v) = T$), otherwise v is a standard node ($\texttt{Is\_Leader}(v) = F$); i.e., $\texttt{Is\_Leader}(v) \equiv (\texttt{Head}(v) = v)$.

• $\texttt{ClusterEmpty}(v) \in \{T, F\}$; It indicates if the v's cluster is empty or not.
$\texttt{ClusterEmpty}(v) \equiv \forall u \in N_v, \texttt{Head}(u) \neq v$.

• $\texttt{MustAffiliate}(v) \in \{T, F\}$; It indicates if node v must affiliate with the NextHead or not. $\texttt{MustAffiliate}(v) \equiv \texttt{GA}(v) \wedge \texttt{Status}_{\texttt{NextHead}(v)} = CH$.

• $\texttt{MustResign}(v) \in \{T, F\}$; It indicates if node v has to resign and join the cluster headed by NextHead or not. $\texttt{MustResign}(v) \equiv \texttt{GR}(v) \wedge \texttt{Status}_{\texttt{NextHead}(v)} = CH$.

• $\texttt{MustBecomeHead}(v) \in \{T, F\}$; It indicates if the node v has to become cluster-head: if $\texttt{GE}(v)$ is enabled or v cannot affiliate with NextHead and it cannot join an existing cluster. $\texttt{MustBecomeHead}(v) \equiv \texttt{GE}(v) \vee (\neg\texttt{MustAffiliate}(v) \wedge \texttt{Status}_{\texttt{Head}(v)} \neq CH)$.

Our transformation is applied to a class of original clustering protocols that can have a deep difference between them. The original protocol may build a dominating set, independent dominating set, k-fold dominating set, capacitated dominating set, connected or weakly connected dominating set etc. The transformed protocol builds the same kind of clusters as the original protocol. The computations of original protocol are however modified to ensure the service guarantee to $\mathcal{TP}$ protocol despite $\mathcal{HTD}$ disruptions. Protocols GDMAC [1], building a k-fold dominating set, and BSC [20] building a capacitated dominating set, are transformed respectively to R-GDMAC [21], and R-BSC [17] using our transformer.

To ensure the service guarantee, $\mathcal{TP}$ protocol maintains, in addition to variables of $\mathcal{P}$ protocol, a variable Status that indicates the hierarchical status of a node. The hierarchical status of a node v is : *cluster-head* ($\texttt{Status}_v = CH$), *ordinary node* ($\texttt{Status}_v = O$), *nearly ordinary* ($\texttt{Status}_v = NO$), or *nearly cluster-head* ($\texttt{Status}_v = NCH$). The status transition diagram of a node v is illustrated in Figure 1, where transitions are the rules executed by v (and defined in Protocol 3).

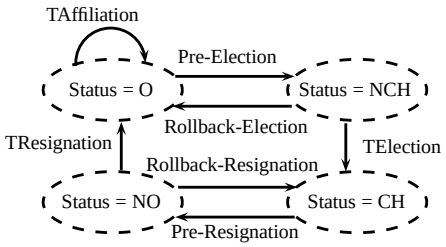

Fig. 1. Status transition in $\mathcal{TP}$ protocol

Protocol 3 : Rules of the Transformed Protocol $\mathcal{TP}$.

Correct1(v) : $\texttt{Is\_Leader}(v) \wedge (\texttt{Status}_v = O \vee \texttt{Status}_v = NCH) \longrightarrow \texttt{Status}_v := CH$

Correct2(v) : $\neg\texttt{Is\_Leader}(v) \wedge (\texttt{Status}_v = CH \vee \texttt{Status}_v = NO) \longrightarrow \texttt{Status}_v := O$

Pre-Election(v) : $\texttt{Status}_v = O \wedge \neg\texttt{Is\_Leader}(v) \wedge \texttt{MustBecomeHead}(v) \longrightarrow \texttt{Status}_v := NCH;$

TElection(v) : $\texttt{Status}_v = NCH \wedge \neg\texttt{Is\_Leader}(v) \wedge \texttt{Ready}_v = RCH \wedge \texttt{MustBecomeHead}(v) \longrightarrow \texttt{Status}_v := CH;\ \texttt{AE}(v);$

Rollback-Election(v) : $\texttt{Status}_v = NCH \wedge \neg\texttt{Is\_Leader}(v) \wedge \neg\texttt{MustBecomeHead(v)} \wedge \neg\texttt{MustAffiliate}(v) \longrightarrow \texttt{Status}_v := O;$

Pre-Resignation(v) : $\texttt{Status}_v = CH \wedge \texttt{Is\_Leader}(v) \wedge \texttt{MustResign}(v) \longrightarrow \texttt{Status}_v := NO;$

TResignation(v) : $\texttt{Status}_v = NO \wedge \texttt{Is\_Leader}(v) \wedge \texttt{ClusterEmpty}(v) \wedge \texttt{Ready}_v = RO \wedge \texttt{MustResign}(v) \longrightarrow \texttt{Status}_v := O;\ \texttt{AR}(v);$

Rollback-Resignation(v) : $\texttt{Status}_v = NO \wedge \texttt{Is\_Leader}(v) \wedge \neg\texttt{MustResign}(v) \longrightarrow \texttt{Status}_v := CH;$

TAffiliation(v) : $\neg\texttt{Is\_Leader}(v) \wedge \texttt{MustAffiliate}(v) \longrightarrow \texttt{Status}_v := O;\ \texttt{AA}(v);$

TComplement(v) : $\texttt{GC}(v) \longrightarrow \texttt{AC}(v)$; `// Complementary rules are not changed.`

The value of `Ready` variable is an input to $\mathcal{TP}$ protocol, and it is updated by an upper-layer hierarchical protocol, called $\mathcal{UHP}$. `Ready` does not have any impact on the transformation of $\mathcal{P}$ to $\mathcal{TP}$, i.e., $\mathcal{TP}$ is self-stabilizing with service guarantee without using `Ready` variable. `Ready` allows just the control of $\mathcal{TP}$ actions by $\mathcal{UHP}$ in order to ensure the service guarantee of $\mathcal{UHP}$ protocol. For example, $\mathcal{UHP}$ can be the knowledge of neighbor clusters protocol proposed in [18], where the minimal service is "the permanent availability of paths leading to the head of each neighbor cluster". `Ready` is thus an interface that enables the implantation of self-stabilizing with service guarantee protocols on the top of $\mathcal{TP}$ protocol, as hierarchical routing protocols. The value RO (resp. RCH) of $\texttt{Ready}_v$ indicates that v is ready to become ordinary (resp. cluster-head) without violating some properties on $\mathcal{UHP}$. For ordinary nodes the default value of `Ready` is RO, and for cluster-heads the default value is RCH.

Assumption 1 *Let v be a node. If $\texttt{Status}_v = NCH$ (resp. $\texttt{Status}_v = NO$) and $\texttt{Ready}_v = RO$ (resp. $\texttt{Ready}_v = RCH$), there exist successive enabled actions from the $\mathcal{UHP}$ protocol that set $\texttt{Ready}_v$ to RCH (resp. RO) in a finite time.*

Predicates and Rules. Correction rules Correct1(v) and Correct2(v) update initially the value of $\texttt{Status}(v)$ according to the value of $\texttt{Is\_Leader}(v)$ predicate. Only one of these rules is enabled at a time by a node v. After execution of one of these rules, both rules are disabled forever on v.

Affiliation process. In $\mathcal{P}$ protocol, a node v affiliates to `NextHead`'s cluster if $\texttt{GA}(v)$ is satisfied. However, if v and $\texttt{NextHead}(v)$ perform respectively Affiliation and Resignation rules during the same computation step, v will be affiliated to a standard node (v is now orphan, because its head is not leader). To avoid generating orphan nodes, $\mathcal{TP}$ protocol authorises the affiliation of v to cluster of `NextHead` only if $\texttt{GA}(v) \wedge \texttt{Status}_{\texttt{NextHead}(v)} = CH$ (i.e., $\texttt{MustAffiliate}(v)$).

Resignation process. For the same reason above, it is not enough that a leader v satisfying $\texttt{GR}(v)$ resigns its leadership. Otherwise, v could be orphan, and it could generate orphan nodes after its resignation. This is why in $\mathcal{TP}$ protocol, a leader v must satisfy the predicate $\texttt{MustResign}(v)$, and its cluster should be empty before becoming a standard node. The resignation process is thus done in two steps. First, a cluster-head v satisfying $\texttt{MustResign}(v)$ has the Pre-Resignation rule enabled. By the execution of Pre-Resignation rule, v becomes nearly ordinary (it still behaves as leader). In this state (i.e., $\texttt{Status}_v = NO$), no node u having $\texttt{NextHead}_u = v$ can join the v's cluster because $\neg\texttt{MustAffiliate}(u)$ and $\neg\texttt{MustResign}(u)$ are satisfied. Furthermore, the members of v's cluster have to leave their cluster, because they satisfy $\texttt{MustAffiliate} \vee \texttt{MustBecomeHead}$, and so they eventually quit the v's cluster. In the other hand, while v is nearly ordinary, $\mathcal{UHP}$ protocol will update $\texttt{Ready}$ to RO in a finite time (Assumption 1). Once the v's cluster is empty (i.e., $\texttt{ClusterEmpty}(v) = T$) and $\texttt{Ready}_v = RO$, the rule TResignation(v) is enabled. By performing TResignation(v) rule, v becomes ordinary, and the Resignation action $\texttt{AR}(v)$ is executed. If $\texttt{MustResign}(v)$ becomes unsatisfied when $\texttt{Status}_v = NO$, then Rollback-Resignation(v) rule is enabled. Execution of Rollback-Resignation stops the resignation process.
These conditions guarantee that during the construction/maintenance of clusters, no cluster-head abandons its leadership and generates orphan nodes.

Election process. A standard node v has to become leader if $\texttt{MustBecomeHead}(v)$ is verified: either due to the satisfaction of $\texttt{GE}(v)$, or because v has to leave its cluster (the v's head is nearly ordinary) but v cannot affiliate with another cluster. The election process is done in two steps. First, an ordinary node satisfying $\texttt{MustBecomeHead}(v)$ has the Pre-Election rule enabled. After its execution, v takes the nearly cluster-head status (it still behaves as a standard node). While v is nearly cluster-head, the protocol $\mathcal{UHP}$ will update $\texttt{Ready}$ to RCH in a finite time (Assumption 1). Once $\texttt{Ready}_v = RCH$ and $\texttt{MustBecomeHead}(v)$ is satisfied, the rule TElection is enabled for v. By executing TElection(v), v becomes cluster-head, and it performs the Election action $\texttt{AE}(v)$. If $\texttt{MustBecomeHead}(v)$ is no more satisfied when $\texttt{Status}_v = NCH$, then Rollback-Election(v) rule is enabled. Its execution leads v to ordinary status and stops the election process.

4 Service guarantee of the transformed protocol $\mathcal{TP}$

In this section, we prove that $\mathcal{TP}$ protocol quickly reaches a safe configuration, in at most 3 rounds. Moreover, the safety property is preserved under any action of $\mathcal{TP}$ protocol and also despite the occurrence of $\mathcal{HTD}$ disruptions. Some proofs are omitted due to lack of space. They can be found in [19].

Lemma 1. *$A_1 = \{c \in \mathcal{C} \mid \forall v \in V : \big(\texttt{Is\_Leader}(v) \Rightarrow \texttt{Status}_v \in \{CH, NO\}\big) \wedge \big(\neg\texttt{Is\_Leader}(v) \Rightarrow \texttt{Status}_v \in \{NCH, O\}\big)\}$ is an attractor from $\mathcal{C}$ in one round.*

Observation 1 *In a configuration of A_1, the rules Correct1(v) and Correct2(v) are disabled for any node v.*

Definition 4 (Safety Predicate). *Let us define the safety predicate $\mathcal{SP}$ as follows:* $\mathcal{SP}_v \equiv \texttt{Head}(\texttt{Head}(v)) = \texttt{Head}(v)$
$\mathcal{SP} \equiv \forall v \in V : \mathcal{SP}_v$

Notation 1 *Let c be a configuration, and $\mathcal{X}$ be a variable or a predicate. We note by $\mathcal{X}[c]$ the value of $\mathcal{X}$ in the configuration c.*

Lemma 2. *Following the execution of TElection, TResignation or TAffiliation rule by a node v, $\mathcal{SP}_v$ is satisfied.*

Proof. Let c_1 be a configuration of A_1, and cs be a computation step of $\mathcal{TP}$ protocol $c_1 \xrightarrow{cs} c_2$. Let v be a node. During cs, if v performs the TElection rule, the predicate $\mathcal{SP}_v$ is verified in c_2 ($\texttt{Head}(v)[c_2] = v$).
Let us study the case where v performs TResignation or TAffiliation rule during cs. We note u the head selected by v during cs ($\texttt{NextHead}(v)[c_1] = u$). In c_1, we have $\texttt{Status}_u = CH$, otherwise predicates $\texttt{MustResign}(v)$ and $\texttt{MustAffiliate}(v)$ are not satisfied in c_1. $\mathcal{SP}_v[c_2]$ is satisfied because u cannot modify the value of $\texttt{Head}(u)$ by performing TResignation or TAffiliation rule during cs. ■

Lemma 3. *The set of configurations $A_2 = A_1 \cap \{c \in C \mid \mathcal{SP}$ is satisfied $\}$ is closed under any computation step of the $\mathcal{TP}$ protocol.*

Proof. Let c_1 be a configuration of A_2, and cs be a computation step of $\mathcal{TP}$ protocol $c_1 \xrightarrow{cs} c_2$. Let v be a node. During cs, there are two possibilities.
• v **did not change its head during** cs. Let u be the head of v in c_1, i.e., $u = \texttt{Head}(v)[c_1] = \texttt{Head}(v)[c_2]$, and $\texttt{Head}(u)[c_1] = u$. TElection(u) and TAffiliation(u) rules are disabled in c_1. So, TResignation(u) is the only rule that modifies the value of $\texttt{Head}(u)$. However, TResignation(u) is disabled in c_1 because $\texttt{ClusterEmpty}(u)[c_1]$ is not satisfied. Thus, $\mathcal{SP}_v$ stays satisfied in c_2.
• v **changes its head during** cs. Note that the Pre-Election, Rollback-Election, Pre-Resignation, Rollback-Resignation, TComplementary rules do not change the v's head identity. During cs, if v performs the other rules, $\mathcal{SP}_v$ becomes verified in c_2 (according to Lemma 2).
We conclude that A_2 is closed under any computation step of $\mathcal{TP}$ protocol. ■

Theorem 1. *A_2 is an attractor for $\mathcal{TP}$ protocol from A_1 in at most two rounds.*

Corollary 1. *A safe configuration is reached in at most 3 rounds.*

Proof. Each configuration of A_2 is safe. The remaining of the proof follows directly from Lemma 1 and Theorem 1. ■

Definition 5 (Highly Tolerated Disruptions). *The set of highly tolerated disruptions* HTD *handled by the protocol $\mathcal{TP}$ is:*

- *the change of node's weight,*
- *the crash of standard nodes,*
- *the failure of a link between (1) two leaders, or (2) two standard nodes,*
- *the joining of sub-networks verifying the predicate $\mathcal{SP}$.*

Theorem 2. *$\mathcal{SP}$ is closed under any disruption of $\mathcal{HTD}$.*

Proof. Let v be a standard node (v is ordinary or nearly cluster-head), and u its head (u is cluster-head or nearly ordinary). Let $c \in A_2$. Starting from c, $\mathcal{SP}_v$ will be not verified only if one of the following events occurs: u's removal from the network or crash, or failure of the communication link between u and v. Therefore, $\mathcal{SP}$ is preserved under any disruption of $\mathcal{HTD}$. ■

5 Correctness of the transformed protocol $\mathcal{TP}$

In this section, we prove that a terminal configuration of $\mathcal{TP}$ protocol is not due to a deadlock situation, but it corresponds to a terminal configuration of $\mathcal{P}$.

Theorem 3. *In a terminal configuration c of $\mathcal{TP}$ protocol, no action of $\mathcal{P}$ protocol is enabled.*

Proof. Let u, v, w be nodes, and let c_t be a terminal configuration of $\mathcal{TP}$ protocol. According to Theorem 1, c_t belongs to $A2$. In the configuration c_t, all rules of $\mathcal{TP}$ protocol are disabled.

Assume that in c_t, v satisfies $\neg\texttt{Is\_Leader}(v)$. In the configuration c_t we have:
• $\texttt{Status}_v = NCH \vee \texttt{Status}_v = O$, since $c_t \in A_1$.
• $\neg\texttt{MustBecomeHead}(v)$ is satisfied, otherwise the Pre-Election or TElection rule is eventually enabled according to Observation 1.
• $\texttt{Status}_v = O$, because otherwise the rule Rollback-Election is enabled.
• $\neg\texttt{MustAffiliate}(v)$ is satisfied, otherwise the rule TAffiliation is enabled.
• $\neg\texttt{GE}(v) \wedge \texttt{Status}_{\texttt{Head}(v)} = CH$, since $\neg\texttt{MustBecomeHead}(v)$ is satisfied.
We conclude that in c_t, $\neg\texttt{Is\_Leader}(v) \Rightarrow$

$$\texttt{Status}_v = O \wedge \neg\texttt{GE}(v) \wedge \neg\texttt{MustAffiliate}(v) \wedge \texttt{Status}_{\texttt{Head}(v)} = CH \qquad (3)$$

In addition, according to 2^{nd} and 3^{rd} Preconditions, we have

$$\texttt{Is\_Leader}(v) \Rightarrow \neg\texttt{GA}(v) \wedge \neg\texttt{GE}(v) \Rightarrow \neg\texttt{MustAffiliate}(v) \wedge \neg\texttt{GE}(v)$$

Therefore, in c_t, we have :

$$\forall v \in V : \neg\texttt{MustAffiliate}(v) \wedge \neg\texttt{GE}(v) \qquad (4)$$

Assume that in c_t the node w satisfies $\texttt{Is\_Leader}(w)$, and it is nearly ordinary ($\texttt{Status}_w = NO$). According to our assumptions, in c_t we have:
• $\texttt{MustResign}(w)$ is satisfied, otherwise the rule Rollback-Resignation is enabled.
• $\neg\texttt{ClusterEmpty}(w)$ is satisfied, otherwise TResignation(w) is eventually enabled (Observation 1). Thus, $\exists u \in N_w : \texttt{Head}(u) = w$ (i.e., $\neg\texttt{Is\_Leader}(u)$). We have, $\texttt{Status}_{\texttt{Head}(u)} = NO$. According to Equation 3, node u does not exist.
• There is a contradiction, in c_t $\texttt{Is\_Leader}(w)$ implies $\texttt{Status}_w \neq NO$.

Assume now that in c_t w is cluster-head, thus in c_t we have:
• $\neg\texttt{MustResign}(w)$ is satisfied, otherwise Pre-Resignation(w) is enabled.
We establish that in c_t,

$$\texttt{Is\_Leader}(w) \Rightarrow \texttt{Status}_w = CH \wedge \neg\texttt{MustResign}(w) \qquad (5)$$

According to 1^{st} Precondition, $\neg\texttt{Is\_Leader}(w) \Rightarrow \neg\texttt{GR}(w) \Rightarrow \neg\texttt{MustResign}(w)$. Therefore, in c_t, we have:

$$\forall v \in V : \neg \texttt{MustResign}(v) \tag{6}$$

According to Equation 1, in c_t we have: $\texttt{GA}(v) \Rightarrow \texttt{Is\_Leader}(\texttt{NextHead}(v))$. Thus, $\texttt{Status}_{\texttt{NextHead}(v)} = CH$ (Equation 5). We conclude that in c_t,

$$\neg \texttt{MustAffiliate}(v) \Rightarrow \neg \texttt{GA}(v) \tag{7}$$

Similarly, according to Equation 1, in c_t we have: $\texttt{GR}(w) \Rightarrow \texttt{Is\_Leader}(\texttt{NextHead}(w))$. Thus, $\texttt{Status}_{\texttt{NextHead}(w)} = CH$ (Equation 5). We conclude that in c_t,

$$\neg \texttt{MustResign}(w) \Rightarrow \neg \texttt{GR}(w) \tag{8}$$

In c_t, $\texttt{GC}(v)$ guards are disabled because TComplementary(v) rules are disabled.

In terminal configuration c_t, the guards $\texttt{GE}(v)$ (Equation 4), $\texttt{GA}(v)$ (Equations 4 and 7), $\texttt{GR}(v)$ (Equations 5 and 8) and $\texttt{GC}(v)$ are disabled for any node v. This is a terminal configuration for $\mathcal{P}$. ■

6 Termination of the transformed protocol $\mathcal{TP}$

The proof of termination of $\mathcal{TP}$ protocol poses a technical challenge. Indeed, some times the rule TElection in $\mathcal{TP}$ protocol may be enabled whereas the Election rule in $\mathcal{P}$ protocol is disabled, i.e., $\texttt{GE}$ is not verified but $\texttt{MustBecomeHead}$ is verified. The execution of TElection rule when $\texttt{MustBecomeHead} \wedge \neg\texttt{GE}$ allows to empty a cluster headed by a Nearly-ordinary node, and so it ensures the convergence of $\mathcal{TP}$ protocol.

Requirement 1 *For the following, we assume that* $A_p = A_2 \cap \{c \in C \mid \forall v \in V, P1(v) \wedge P2(v)\}$ *is an attractor for* $\mathcal{TP}$ *protocol from* A_2 *where:*

$$P1(v) \equiv (\texttt{GA}(v) \vee \texttt{GE}(v)) \wedge (\texttt{Head}(\texttt{Head}(v)) = \texttt{Head}(v)) \Rightarrow w_{\texttt{NextHead}(v)} > w_{\texttt{Head}(v)}$$

$$P2(v) \equiv (\forall u \in V, w_u < w_v \text{ or } u \text{ will never perform a clustering action}) \Rightarrow \text{The value of } \texttt{GR}(v) \text{ does not change while } v \text{ does not perform an action.}$$

The predicate $P1$ is related to the fact that $\mathcal{P}$ is weight-based: a standard node of a well-formed cluster (its head is a leader) changes of cluster only to affiliate to a better leader. The predicate $P2$ is related to silent and weigh-based properties of $\mathcal{P}$: a leader v is neutralized only by an action of a stronger node (its weight is larger than v's weight).

Termination scheme : Let e be a computation of $\mathcal{TP}$ protocol starting from a configuration of $A_2 \cap A_p$. Along e, the stabilization of nodes of V is done in steps. At the end of the i^{th} step, a suffix e_i of e is reached where all nodes of S_i executes only Pre-Election and Rollback-Election rules. We define the set S_i, and the suffix e_i as follows:

- $S_0 = \emptyset$; $e_0 = e$; $i \geqslant 1$;
- $V_i = V - S_{i-1}$;
- Let vi be the node of V_i having the highest weight.

- Let e_i be a suffix of e_{i-1}, such that along e_i the following *stabilization properties* are always satisfied for the node vi:
  1. $\texttt{Status}_{vi} \in \{CH, NCH, O\}$, and vi will never change its head identity.
  2. If vi is cluster-head, then vi is disabled forever, and the vi's cluster is stable (i.e., no node joins or leaves the cluster headed by vi).
  3. If vi is ordinary or nearly cluster-head, then vi only executes Pre-Election and Rollback-Election rules.
- $S_i = S_{i-1} \cup \{vi\}$.

Lemma 4. *For all $i \geqslant 1$, the suffix e_i of e_{i-1} exists assuming that the suffix e_{i-1} of e_0 exists.*

Theorem 4. *All computations of $\mathcal{TP}$ protocol, starting from a configuration of $A_2 \cap A_p$, reach a terminal configuration.*

Proof. Let $j = |V|$ be an integer. The suffix e_j exists (where *stabilization properties* are satisfied for all nodes of V), and it is reached by any computation of $\mathcal{TP}$ protocol (Lemma 4). Along e_j, nodes may only execute Pre-Election and Rollback-Election rules. So, no node executes a clustering action (i.e., `AA`, `AE`, `AR`, and `AC` actions), and the value of guards $\texttt{GA}(v)$, $\texttt{GE}(v)$, $\texttt{GR}(v)$, and $\texttt{GC}(v)$ does not change for any node v. Furthermore, along e_j, $\forall v \in V, \texttt{Status}_v \neq NO$.
Assume that e_j is infinite. So, there exists a set of nodes, denoted $\texttt{Inf} \neq \emptyset$, that perform infinitely often Pre-Election and Rollback-Election rules. Let v be the node of `Inf` having the highest weight.
Along e_j, each time v satisfies $\texttt{MustBecomeHead}(v)$ (to perform Pre-Election rule) then the guard $\texttt{GE}(v)$ is satisfied, because $\texttt{Status}_{\texttt{Head}(v)} = CH$. Since no node performs a clustering action, the node v satisfying $\texttt{GE}(v)$ stays enabled along e_j unless it performs a clustering action. By fairness, v executes TElection rule after Pre-Election rule and it leaves its cluster. This is impossible along e_j. We conclude that $\texttt{GE}(v)$ is not verified along e_j. Moreover, along e_j we have $\texttt{Status}_{\texttt{Head}(v)} = CH$. Thus, `MustBecomeHead(v)` is never satisfied along e_j.
Therefore, along e_j, Pre-Election(v) is disabled forever, and after the execution of Rollback-Election(v) rule, v is disabled forever.

We conclude that v does not perform infinitely often the Pre-Election and the Rollback-Election rules: $\texttt{Inf} = \emptyset$. Therefore, e_j reaches a terminal configuration. ■

7 Complexity measures and concluding remarks

Time complexity. A comparison between the time complexity of $\mathcal{P}$ and $\mathcal{TP}$ protocols is illustrated in Table 1, where $\mathcal{UHP}$ rules are rules of $\mathcal{UHP}$ protocol updating the variable *Ready*, and U is the time required by $\mathcal{UHP}$ rules to achieve such update. We conclude that an upper bound of the stabilization time of $\mathcal{TP}$ protocol is $(4 + 2U) * S_{\mathcal{P}}$, where $S_{\mathcal{P}}$ is the stabilization time of $\mathcal{P}$ protocol.

Memory space complexity. Let $M_{\mathcal{P}}$ be the memory requirement of protocol $\mathcal{P}$ at each node. The protocol $\mathcal{TP}$ differs from $\mathcal{P}$ by the variable `Status` added

at each node. This variable has 4 values, so it can be coded by 2 bits. Thus, the memory space complexity of $\mathcal{TP}$ protocol is $M_{\mathcal{P}} + 2$ bits per node.

Protocol $\mathcal{P}$		Protocol $\mathcal{TP}$	
Rule	Number of rounds	Rule	Number of rounds
Complementary	1 round	TComplementary	1 round
Affiliation	1 round	TAffiliation	1 round
Election	1 round	Pre-Election + $\mathcal{UHP}$ rules + TElection	$2 + U$ rounds
Resignation	1 round	Pre-Resignation + (Pre-Election + $\mathcal{UHP}$ rules + TElection **or** TAffiliation) + $\mathcal{UHP}$ rules + TResignation	$4 + 2U$ rounds

Table 1. Comparison between time complexity of $\mathcal{P}$ and $\mathcal{TP}$ protocols

The proposed scheme constructs a silent self-stabilizing with service guarantee 1-hop clustering protocol $\mathcal{TP}$ starting from a silent self-stabilizing one $\mathcal{P}$. In at most 3 rounds (Corollary 1), $\mathcal{TP}$ provides the following useful minimal service: "each node belongs to a cluster having an effectual leader". The service guarantee property of $\mathcal{TP}$ protocol ensures that this minimal service stays provided during the stabilization phase, even despite the occurrences of disruptions $\mathcal{HTD}$ (see Definition 5). Thus, the hierarchical organization of the network is quickly available and it is maintained over the time, which allows the continuity of operation of upper-layer hierarchical protocols.

Futur works. The presented transformer is adapted only to self-stabilizing 1-hop weight-based protocols. A first generalization of this work is the design of a transformer dealing with k-hops weight-based protocols (i.e. the cluster-head being at distance at most k of its cluster's members). A second generalization is the design of a transformer adapted to any k-hops protocol; for instance [14] where the selection of cluster-heads is randomized and not weight-based.

References

1. S. Basagni. Distributed and mobility-adaptive clustering for multimedia support in multi-hop wireless networks. In *International Vehicular Technology Conference (VTC'99)*, pages 889–893, 1999.
2. D. Bein, A. K. Datta, C. R. Jagganagari, and V. Villain. A self-stabilizing link-cluster algorithm in mobile ad hoc networks. In *International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN'05)*, pages 436–441, 2005.
3. L. Blin, M. Potop-Butucaru, S. Rovedakis, and S. Tixeuil. Loop-free super-stabilizing spanning tree construction. In *Stabilization, Safety, and Security of Distributed Systems (SSS'10)*, volume 6366, pages 50–64, 2010.
4. A. Bui, A. K. Datta, F. Petit, and V. Villain. Snap-stabilization and PIF in tree networks. *Distributed Computing*, 20:3–19, 2007.
5. E. Caron, A. K. Datta, B. Depardon, and L. L. Larmore. self-stabilizing k-clustering algorithm for weighted graphs. *Journal of Parallel and Distributed Computing*, 70:1159–1173, 2010.

6. M. Chatterjee, S. K. Das, and D. Turgut. WCA: A weighted clustering algorithm for mobile ad hoc networks. *Journal of Cluster Computing*, 5(2):193–204, 2002.
7. A. Cournier, A.K. Datta, F. Petit, and V. Villain. Enabling snap-stabilization. In *Conference on Distributed Computing Systems (ICDCS'03)*, pages 12 – 19, 2003.
8. A. Cournier, S. Devismes, and V. Villain. From self- to snap- stabilization. In *Stabilization, Safety, and Security of Distributed Systems (SSS'06)*, volume 4280, pages 199–213. 2006.
9. A. K. Datta, L. L. Larmore, and P. Vemula. A self-stabilizing o(k)-time k-clustering algorithm. *The Computer Journal*, 53:342–350, 2010.
10. S. Delaët, S. Devismes, M. Nesterenko, and S. Tixeuil. Snap-stabilization in message-passing systems. In *ICDCN'09*, volume 5408, pages 281–286, 2009.
11. M. Demirbas, A. Arora, V. Mittal, and V. Kulathumani. A fault-local self-stabilizing clustering service for wireless ad hoc networks. *IEEE Transactions on Parallel and Distributed Systems*, 17:912–922, 2006.
12. E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11):643–644, 1974.
13. S. Dolev and T. Herman. Superstabilizing protocols for dynamic distributed systems. *Chicago J. Theor. Comput. Sci.*, 1997, 1997.
14. S. Dolev and N. Tzachar. Empire of colonies: Self-stabilizing and self-organizing distributed algorithm. *Theoretical Computer Science*, 410:514–532, 2009.
15. V. Drabkin, R. Friedman, and M. Gradinariu. Self-stabilizing wireless connected overlays. In *Conference On Principles Of Distributed Systems (OPODIS'06), Springer LNCS 4305*, pages 425–439, 2006.
16. M. Gerla and J. T. Tsai. Multicluster, mobile, multimedia radio network. *Journal of Wireless Networks*, 1(3):255–265, 1995.
17. C. Johnen and F. Mekhaldi. Robust self-stabilizing construction of bounded size weight-based clusters. In *Euro-Par'10, Springer LNCS 6271*, pages 535–546, 2010.
18. C. Johnen and 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)*, pages 41–50, 2011.
19. C. Johnen and F. Mekhaldi. From self- to self-stabilizing with service guarantee 1-hop weight-based clustering. Technical Report RR1462-12, LaBRI, 2012. `http://hal.archives-ouvertes.fr/`.
20. C. Johnen and L. H. Nguyen. Self-stabilizing construction of bounded size clusters. In *International Symposium on Parallel and Distributed Processing with applications (ISPA'08)*, pages 43–50, 2008.
21. C. Johnen and L. H. Nguyen. Robust self-stabilizing weight-based clustering algorithm. *Theoretical Computer Science*, 410(6-7):581–594, 2009.
22. S. Kamei and H. Kakugawa. A self-stabilizing approximation for the minimum connected dominating set with safe convergence. In *Conference On Principles Of Distributed Systems (OPODIS'08), Springer LNCS 5401*, pages 496–511, 2008.
23. S. Katz and K. J. Perry. Self-stabilizing extensions for meassage-passing systems. *Distributed Computing*, 7:17–26, 1993.
24. N. Mitton, E. Fleury, I. Guérin-Lassous, and S. Tixeuil. Self-stabilization in self-organized multihop wireless networks. In *International Conference on Distributed Computing Systems Workshops (WWAN'05)*, pages 909–915, 2005.