Self-Stabilizing Computation and preservation of Knowledge of neighbor clusters

Colette Johnen
Université Bordeaux, LaBRI UMR 5800,
F-33405 Talence, France
Email : johnen@labri.fr

Fouzi Mekhaldi
Université Paris-Sud, LRI UMR 8623,
F-91405 Orsay, France
Email : mekhaldi@lri.fr

***Abstract*—The area of self-stabilization in large scale networks has been received increasing attention among researchers, since self-stabilization provides a foundation for self-properties, including self-healing, self-organizing and self-adaptive. This paper makes contributions in two areas. First, we describe a new extended approach of self-stabilization, called self-stabilization with service guarantee. Second, we propose a self-stabilizing protocol computing and preserving the knowledge of neighbor clusters, called `CNK`. A cluster-head maintains about each neighbor cluster: the identity of its head, paths leading to it, and the list of members. The most interesting property of `CNK` is the service guarantee during the stabilization phase. `CNK` quickly provides, in at most 4 rounds, the following minimal useful service: "each cluster-head knows valid paths leading to heads of all its neighbor clusters". `CNK` protocol preserves the minimal service despite changes in the clustering structure (creation of new clusters, restructuring or crumbling of existing clusters). The knowledge of neighbor clusters is thus highly available. This knowledge is enough to allow the continuity functioning of hierarchical protocols as hierarchical routing protocols.**

I. Introduction

A mobile Ad-hoc or sensor network is a distributed multi-hops network which consists of mobile hosts that move arbitrarily, and communicate between them via wireless technologies without preexisting infrastructure. The flat architecture of such networks is not scalable, because all nodes are considered equal and they take the same part in the network management, like routing and forwarding tasks. The clustering was introduced for improving scalability by supporting self-organization and enabling hierarchical routing.

The clustering consists of partitioning network nodes into non-overlapping groups called clusters. Each cluster has a single head that acts as local coordinator of the cluster, and eventually a set of ordinary nodes. So, clustering creates a first hierarchical level (level 1) of a flat topology (level 0). The clustering problem is well studied in the context of multi-hop wireless networks [1]. In this paper, the studied problem is not the clustering, but the knowledge of neighbour clusters assuming the existence of an under-layer clustering protocol.

Motivation. Multi-level hierarchies can be obtained progressively: clusters of level i are regarded as nodes of level $i+1$, and the construction is started again in this level. Hence, the creation of a level $i+1$ requires the knowledge of neighbor clusters of level i.

Furthermore, hierarchical protocols consider the cluster (not the node) as the basic entity of the network. For example, hierarchical packet forwarding is achieved from a cluster to a neighbor one, until reaching the destination cluster. Thus, routing packets between distant nodes in clustering architecture requires also an efficient knowledge of neighbor clusters. This knowledge is computed by our protocol `CNK`.

In another hand, clustering protocols need to be self-adaptive in order to deal with topology changes like: links creation, links failure, nodes departure and nodes arrival. Thereby, the obtained hierarchical structure will be itself dynamic, due to: creation of new clusters, disbanding of clusters, and change on the composition of a cluster after ordinary nodes switching from a cluster to another one. As consequence, all hierarchical protocols must be adaptive to changes in the hierarchical structure.

For all these reasons, discovering and maintaining efficiently the neighborhood of each cluster becomes a necessity. Moreover, as changes in the hierarchical structure may occur frequently, it is vital to avoid disruption of hierarchical protocols. Thus, the knowledge of a valid path to neighbor clusters should be always available in spite of modifications in the clustering structure. Hence, our `CNK` protocol is a self-stabilizing protocol; moreover it ensures a service guarantee.

Self-stabilization with service guarantee. 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 perturbations in a well-defined manner. Self-stabilization

is an approach to achieve the fault-tolerance. A self-stabilizing protocol, regardless of its initial state, converges in finite time without any external intervention to a legitimate state, from which the intended behavior is exhibited. Self-stabilizing protocols are attractive because they do not require any correct initialization; they can recover from any transient failure, and they are insensitive to dynamic topology reconfigurations. Nevertheless, during convergence periods, self-stabilizing protocols do not guarantee any property. Thus, self-stabilization is suited for distributed systems with intermittent disruptions, where the delay between two successive disruptions is so large that the system recovers and provides the full (optimum) service for some times. However, in large mobile networks where the topology changes very often, the paradigm of self-stabilization is no more satisfying. Indeed, the delay between disruptions is no longer enough for the convergence. So, the system may be continually disrupted and it never provides the optimum service. This situation may generate a total loss of service. As consequence, the availability and reliability of self-stabilizing systems may be compromised.

A protocol is self-stabilizing with service guarantee if (1) it is self-stabilizing; (2) it reaches a configuration where the minimal service is provided; (3) the minimal service is preserved during progress of the protocol toward the optimum service (i.e., during convergence to a legitimate configuration) and, (4) it is maintained despite the occurrence of some specific disruptions, called highly tolerated disruptions HTD. As tiny it is the delay between two consecutive occurrences of HTD disruptions, the useful minimal service stays provided. In this approach, disruptions highly tolerated are captured and handled by the service guarantee mechanism, in such a way that the minimal service stays provided. Whereas, the occurrence of other disruptions, is handled by the self-stabilization mechanism, i.e., after their occurrence, the system may behave arbitrary, but it converges to a configuration providing a minimal service. So, the service guarantee is provided via both the recovering to a minimal useful service, and its preservation despite the occurrence of HTD disruptions.

Contribution. The knowledge of neighbour clusters is required in many distributed protocols for hierarchical routing and multi-levels clustering, such as [3]. In this paper, we propose a protocol for Clusters Neighbor Knowledge (CNK) that is self-stabilizing with service guarantee. CNK protocol assumes the existence of a 1-hop clustering protocol, that provides and maintains the hierarchical structure. On each head, CNK protocol builds and maintains the knowledge of its neighbor clusters in a self-stabilizing manner. The knowledge stored by a head v about each neighbor cluster C, once CNK protocol has stabilized is the following: (i) the head identity of C, (ii) the path between v and the head of C, and (iii) the members list of C.

The minimal useful service for CNK is defined as follows: "*each head of cluster knows heads of all neighbor clusters and valid paths leading to them*". The goal is to maintain the minimal service in spite of clustering structure changes. The clustering protocol actions, i.e., changes in the hierarchical structure, are not transient events. The minimal delay between their occurrences is unbounded; it depends on the clustering protocol and on the network topology dynamism. Therefore, the set of HTD disruptions handled by CNK is actions done by the clustering protocol (defined in section II).

To preserve the minimal useful service, CNK protocol requires from the clustering protocol two properties described in section IV. Some clustering protocols meet these two properties, like [16] and [13].

In hierarchical networks, the packet routing is achieved from a cluster to a neighbor cluster, until reaching the destination cluster. Thus, the useful service highly available provided by CNK is sufficient for upper layer hierarchical protocols, as hierarchical routing protocols.

Related Works. The self-stabilization with service guarantee is related to the super-stabilization [8], robust self-stabilization [17], [16], [13] and the safe convergence [18].

A super-stabilizing protocol ensures that (1) a safety predicate is satisfied in spite of a single topology change that occurs from a legitimate configuration and, (2) this safety predicate stays satisfied during the convergence to a legitimate configuration assuming that no more topology change event occurs during the convergence. As a legitimate configuration is also safe, a self-stabilizing with service guarantee protocol is super-stabilizing.

A robust self-stabilizing or self-stabilizing with safe convergence protocol quickly reaches a safe configuration where a minimal service is provided. The safety property is preserved during the convergence to a legitimate configuration. In case of a robust self-stabilization, the safety property is also preserved despite of the occurrence of HTD events. The goal of service guarantee within CNK protocol is not to quickly converge to a safe configuration where the minimal service is provided, because the convergence to a legitimate configuration is fast enough (it is done in constant time). The main objective is to maintain the minimal service in spite of clustering structure changes. Thus, a self-stabilizing with service

guarantee protocol ensures the safe convergence and the robustness properties.

Many 1-hop clustering algorithms have been proposed in the literature. A large number of them are self-stabilizing [2], [7], [10], [6], [9], [15], [19]. There are also robust self-stabilizing clustering algorithms [16], [13], and self-stabilizing with safe convergence [18].

Algorithms building neighborhood knowledge in flat architectures are presented in [5], [11], [12]. In [5], algorithms computing 2-hops neighborhood in wireless networks are presented (they are based on geographic position or distance). In [11], it is presented a self-stabilizing mechanism implementing an algorithm requiring distance-two knowledge in a standard network where nodes only communicate with their 1-hop neighbors. In [12], this mechanism is extended to distance-k knowledge. This mechanism assumes centralized scheduler (during a computation step only one node does an action). None of these algorithms guarantees any property after a topology change.

The paper is organized as follows. In section II, we describe the model. Specification of knowledge of neighbor clusters problem is defined in section III. In section IV, we present the required interaction between clustering and CNK protocols in order to provide the service guarantee. The two modules of CNK: computation of knowledge tables, and service guarantee mechanism are presented respectively in sections V and VI. A sketch of proof is provided in section VII. The detailed proofs are omitted due to lack of space. They can be found in [14]. Finally, simulation results are presented in section VIII.

II. Model

A distributed system S is modeled by an undirected graph $G\ (V,\ E)$, where V is the set of (mobile) nodes and E is the set of edges. There is an edge $(u,v) \in E$, if u and v can communicate between them (links are bidirectional). In this case, u and v are neighbors. We note N_v the set of v's neighbors (the neighborhood): $N_v = \{u \in V \mid (u,v) \in E\}$. The internal nodes among a path connecting two nodes u and v, are called gateways.

We use the *local shared memory* as communication model. Each node v maintains a set of local variables such that v can read its own variables and those of its neighbors, but can modify only its own one. The content's of a node's local variables determine its *state*, and the union of all local states determines the *configuration* of the system. The *program* of each node is given as a set of *rules* of the form: $\{Rule_i : Guard_i \longrightarrow Action_i\}$. A rule can be executed by a node v only if it is *enabled*, i.e., its guard is satisfied. A node is said to be enabled if it has at least one rule 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 their actions, and the system reaches the configuration c_{i+1} from c_i. A *computation* is a sequence of configurations $e = c_0, c_1, ..., c_i, ...$, where c_{i+1} is reached from c_i by one computation step. A computation e is maximal if it is infinite, or if it reaches a terminal configuration. A computation e is *weakly fair*, if for any node v that is infinitely often enabled along e, it eventually performs an action. In this paper, we study only weakly fair computations. We note by $Conf$ the set of all configurations, and by $\mathcal{E}$ the set of all weakly fair computations. The set of weakly fair computations starting from a particular configuration $c \in Conf$ is denoted $\mathcal{E}_c$. $\mathcal{E}_A$ is the set of weakly fair computations whose the initial configuration belongs to $A \subset Conf$.

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 node v enabled in c_1, either executes a rule or becomes disabled during 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. The round complexity of a computation is the number of disjoint rounds of this computation.

Definition 1 (Attractor): Let B_1 and B_2 be subsets of configurations of $Conf$. B_2 is an attractor from B_1, if and only if the following conditions hold:

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

Definition 2 (Self-stabilization): A system S is self-stabilizing if and only if there exists a set $\mathcal{L}$ of configurations, named legitimate configurations, such that:

- $\mathcal{L}$ is an attractor from $Conf$.
- Configurations of $\mathcal{L}$ satisfy the specification of problem.

Definition 3 (Stabilization with service guarantee): Let $\mathcal{P}$ be the predicate that stipulates the minimal service. Let $\mathcal{HTD}$ be a set of disruptions. A self-stabilizing protocol has service guarantee despite $\mathcal{HTD}$ disruptions if and only if the set of configurations satisfying $\mathcal{P}$ is:

- Closed under any computation step.
- Closed under any disruption of $\mathcal{HTD}$.

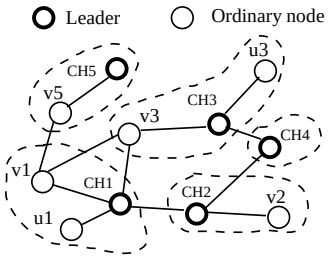


Figure 1. Clustered network example

Definition 4 (Highly Tolerated Disruptions): The set of $\mathcal{HTD}$ disruptions handled by CNK protocol is:

- Selection of new cluster-heads.
- Resignation of cluster-heads.
- Switching of nodes from a cluster to another one.

III. Specification of "Knowledge of Neighbor Clusters" in 1-hop cluster structure

In this paper, we focus on the knowledge of neighbor clusters. Obviously, a clustering architecture should be built and maintained over the time. We assume the existence of a self-stabilizing 1-hop clustering protocol, which runs simultaneously with our protocol CNK. The clustering protocol gathers network nodes into 1-hop clusters. Each cluster has a single head, and a set of ordinary nodes which are neighbor of their heads.
In 1-hop clustering structure, two clusters C_1, C_2 are neighbor, if there exist two nodes $x \in C_1$ and $y \in C_2$ that are neighbors ($(x,y) \in E$). Thus, in this structure, two leaders u and v of neighbor clusters are at most at distance 3. Furthermore, the path between the two leaders u and v should not contain a leader.

Example. In Figure 1, although $CH3$ and $CH2$ are at distance 3, their clusters are not neighbor; because, all paths between $CH2$ and $CH3$ contain a leader.

Definition 5: **The $k-$neighborhood** of a node $v \in V$, denoted N_v^k, is the set of nodes that are at distance less or equal than k from v.

Definition 6: **The kR-neighborhood** of a node v (for k restricted neighborhood), denoted RN_v^k, is the set of nodes in v's k-neighborhood reached by at least a path in which the gateway(s) is (resp. are) not leader(s) (for definition of leader, see Definition 8).

The knowledge built by CNK protocol, has to verify the *completeness* and *correctness* properties.

- **Completeness**: Each leader knows all paths leading to all leaders within its 3R-neighborhood.
- **Correctness**: Each leader knows only valid paths leading to leaders within its 3R-neighborhood. Furthermore, each leader knows the exact member list of its neighbor clusters.

Definition 7 (Legitimate configurations): In a legitimate configuration, the completeness and correctness properties are satisfied.

The interest of self-stabilization with service guarantee in the CNK protocol depends mainly on the two factors:

- **The definition of minimal service:** the minimal service is the completeness property.
- **The list of highly tolerated disruptions:** all actions done by the clustering protocol (see definition 4).

Let us study the hierarchical structure presented in figure 1. We assume that the completeness property is verified: $CH1$ knows paths leading to $CH2$, $CH3$, and $CH5$, but not paths leading to $u3$, and $CH4$. The clustering protocol changes the clustering structure: $u3$ becomes leader and $CH3$ becomes ordinary. Now, $CH1$ has two new neighbor clusters: those of $u3$ and $CH4$, but it does know any paths leading to these leaders. So, the completeness property is no more satisfied. To avoid this situation, CNK bridles the occurrence of HTD disruptions using an interaction with the clustering protocol.

IV. Requirements on the clustering protocol

As said previously, CNK assumes the existence of an under-layer 1-hop clustering protocol. CNK requires cooperation from the clustering protocol to ensure the service guarantee. For this reason, the clustering protocol must meet two properties: robustness and proper interactions with CNK. Some 1-hop clustering protocols already follow the two properties like [16] and [13]. For other self-stabilizing protocols, we believe that is feasible to design a transformer providing a compatible version with CNK protocol.

Proper interactions (illustrated in Figure 2). The variable *status* indicates the hierarchical status of a node. It is updated only by the clustering protocol, and its value is an input to the CNK protocol. Conversely, the variable *Ready* is updated only by the CNK protocol, and its value is an input to the clustering protocol.
The usual hierarchical status of a node v are : *cluster-head* ($Status_v = CH$), and *ordinary node* ($Status_v = O$). Two intermediate hierarchical status are introduced: *nearly ordinary* ($Status_v = NO$), and *nearly cluster-head* ($Status_v = NCH$). The status transition diagram now is as follows: when a cluster-head want to resign its role, it takes the nearly ordinary status (NO). Whereas, when an ordinary node want to become cluster-head, it takes the nearly cluster-head status (NCH). By taking an intermediate status (NCH or NO), the clustering protocol "sends a request" to CNK protocol. Then, the clustering protocol waits the approval from CNK. This authorization

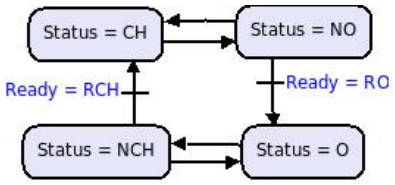


Figure 2. Proper interactions between clustering and CNK protocol.

Name	Destination	G1	G2	List	HS	PIF
Type	ID	ID or $\perp$	ID or $\perp$	$\{IDs\}$	CH, NCH or NO	C, B or F
Field notation	dest	g1	g2	list	hs	pif

Figure 3. Scheme of Knowledge Table

"is communicated" by CNK protocol through the variable *Ready*.

The value *RO* (resp. *RCH*) of $Ready_v$ indicates that v is ready to be ordinary (resp. cluster-head) without violating the completeness property (i.e., minimal service).

For ordinary nodes the default value of *Ready* is *RO*, and for cluster-heads the default value is *RCH*. Only the updating of *Ready* variable allows CNK protocol to ensure the preservation of completeness property.

A nearly cluster-head v can become cluster-head, only if $Ready_v$ is set to *RCH*; but it may return to the ordinary status at any moment (even if $Ready_v = RCH$). Similarly, a nearly ordinary node v can become ordinary only if $Ready_v$ is set to *RO*, but it may return to the cluster-head status at any moment (even if $Ready_v = RO$).

While a node v has the nearly ordinary status, it still behaves as a leader of cluster. Furthermore, v is regarded by CNK protocol as a future ordinary node: it may be a gateway on a path between leaders. In the other hand, a nearly cluster-head u behaves both as an ordinary node and as a cluster-head. In this status, u maintains a pseudo-cluster which is empty (u is the only node in its cluster).

***Definition** 8 (**leader / pseudo-leader**):* A leader v is a node having the status of cluster-head or nearly ordinary. A pseudo-leader v is a node having the nearly cluster-head status.

V. CNK PROTOCOL : COMPUTATION OF THE KNOWLEDGE TABLES

Each node v builds and maintains a Knowledge Table KT_v having the structure presented in Figure 3. This table contains the list of leaders and pseudo-leaders within v's 3R-neighborhood, associated with a path leading to them, as well as the composition of their cluster (or pseudo-cluster).

Each record of KT_v is identified by the fields $dest$, $g1$ and $g2$; it is the primary key of KT_v. In follows, we specify by (x,y,z) a record where $dest = x, g1 = y, g2 = z$.

The destination field of KT_v contains the identity of leaders and pseudo-leaders (whose the status is not *ordinary*) which are at distance 3 at most from v. The gateways used to reach the destination are stored in the first and second gateway fields ($g1$ and $g2$). The value of $g1$ (resp. $g2$) in a record $(u,g1,g2)$ of KT_v is the first (resp. second) gateway on the path from v to u if it exists; otherwise, the value is $\perp$. The list (resp. status) field contains the list of cluster members (resp. hierarchical status) of the destination node. The utility of pif field is discussed in section VI.

According to Figure 1, once the Knowledge Tables have being computed, the record $(CH5, v1, v5)$ belongs to KT_{CH1}, and $(CH1, v5, v1)$ belongs to KT_{CH5}.

The CNK protocol variables and macros are presented in Protocol **5**. The variable HS_v indicates the hierarchical status of a node v according to CNK protocol. HS_v value should be similar to the value of $Status_v$. Notice that only the Clustering protocol updates the variable $Status_v$, and only the CNK updates variables HS_v and $Ready_v$.

Protocol 1 : Variables and macros on node v.

Input variables (from the clustering layer)

$Status_v \in \{CH, O, NO, NCH\}$; The hierarchical status of v.
$Head_v \in \{IDs\}$; The cluster-head's identity of v.

Output variables (towards the clustering layer)

$Ready_v \in \{RO, RCH\}$; (defined in section IV)

Shared variables

$HS_v \in \{CH, O, NO, NCH\}$; The status of v : local copy (within CNK) of $Status_v$.
KT_v; The Knowledge Table. Its scheme is presented in Figure 3.

Macros

$Cluster_v$ :: **if** $HS_v \in \{CH, NO\}$ **then** $\{z \in N_v : Head_z = v\} \cup \{v\}$;
if $HS_v = NCH$ **then** $\{v\}$;

$Insert(dest, g1, g2, List, status)$, adds a record to KT_v, such that the pif field is set to C.

$Delete(x,y,z)$, removes from KT_v the record identified by $dest = x, g1 = y$ and $g2 = z$.

$Update(x,y,z,ls,st)$ :: Update KT_v Set $list := ls$; $hs := st$; within the record identified by $dest = x,\ g1 = y,\ g2 = z$

$UpdatePIF(x,y,z,t)$ :: Update KT_v Set $pif := t$ where $dest = x,\ g1 = y,\ g2 = z$

$UpdateReady$ :: **if** $HS_v = CH$ **then** $Ready_v := RCH$;
if $HS_v = O$ **then** $Ready_v := RO$;

Each leader uses only the Knowledge Table of its neighbors to compute its one. Every table KT_v is updated by 4 rules. Each rule Ri ($i \in [1,3]$) has 3 kinds of sub-rules: insertion of a new record (Ri_1), updating a record (Ri_2, and Ri_4), and deleting a record (Ri_3, and Ri_5 if it exists).

The rule $R0_1(v)$ adds the record $(v, \perp, \perp)$ to KT_v if it does not exist although v is a leader or a pseudo-leader. $R0_3$ is enabled for ordinary nodes ($Status_v = O$), till KT_v contains the record $(v, \perp, \perp)$. A leader or a pseudo-leader

v having an incorrect HS_v value, is enabled (rule $R0_2$ if $(v,\perp,\perp) \in KT_v$ otherwise rule $R0_1$). The rule $R0_4(v)$ updates the fields *list* and *hs* of the record associated to v's cluster in KT_v.

The rule $R1$ ensures that each node v (whatever its status) has an accurate record about clusters whose the head is at distance 1 from v. Similarly, the rule $R2$ maintains the validity of records whose the destination is at distance 2 from v, according only to the content of the Knowledge Tables of v's neighbors. v keeps (or adds) the record $(u,z,\perp)$ to KT_v only if z (a v's neighbor) is not a cluster-head and if KT_z contains a record about the u's cluster.

$\mathbf{R0_1}(v) :: (Status_v \neq O) \wedge (v,\perp,\perp) \notin KT_v$
$\longrightarrow HS_v := Status_v;\ UpdateReady;\ Insert(v,\perp,\perp,Cluster_v,HS_v);$

$\mathbf{R0_2}(v) :: (Status_v \neq O) \wedge (v,\perp,\perp) \in KT_v \wedge (HS_v \neq Status_v)$
$\longrightarrow HS_v := Status_v;\ UpdateReady;\ Update(v,\perp,\perp,Cluster_v,HS_v);$
$UpdatePIF(v,\perp,\perp,C);$

$\mathbf{R0_3}(v) :: (Status_v = O) \wedge \big(HS_v \neq Status_v \vee (v,\perp,\perp) \in KT_v\big)$
$\longrightarrow HS_v := Status_v;\ UpdateReady;\ Delete(v,\perp,\perp);$

$\mathbf{R0_4}(v) :: (Status_v \neq O) \wedge \exists(v,\perp,\perp,List,hs) \in KT_v \wedge (HS_v = Status_v)\ \wedge$
$\big(List \neq Cluster_v \vee hs \neq HS_v\big) \longrightarrow Update(v,\perp,\perp,Cluster_v,HS_v);$

$\mathbf{R1_1}(v) : \exists u \in N_v \wedge \exists(u,\perp,\perp,list,hs) \in KT_u \wedge (u,\perp,\perp) \notin KT_v$
$\longrightarrow Insert(u,\perp,\perp,list,hs);$

$\mathbf{R1_2}(v) : \exists(u,\perp,\perp,list',hs') \in KT_v \wedge (u \in N_v) \wedge \exists(u,\perp,\perp,list,hs) \in KT_u\ \wedge$
$(hs' \neq hs) \longrightarrow Update(u,\perp,\perp,list,hs);\ UpdatePIF(u,\perp,\perp,C);$

$\mathbf{R1_3}(v) : \exists(u,\perp,\perp) \in KT_v \wedge (u \neq v) \wedge \big(u \notin N_v \vee (u,\perp,\perp) \notin KT_u\big)$
$\longrightarrow Delete(u,\perp,\perp);$

$\mathbf{R1_4}(v) : \exists(u,\perp,\perp,list',hs') \in KT_v \wedge (u \in N_v) \wedge \exists(u,\perp,\perp,list,hs) \in KT_u\ \wedge$
$(hs' = hs) \wedge (list' \neq list) \longrightarrow Update(u,\perp,\perp,list,hs);$

$\mathbf{R2_1}(v) : \exists z \in N_v \wedge (HS_z \neq CH) \wedge \exists(u,\perp,\perp,list,hs) \in KT_z \wedge (u \neq z) \wedge (u \neq v)\ \wedge$
$(u,z,\perp) \notin KT_v \longrightarrow Insert(u,z,\perp,list,hs);$

$\mathbf{R2_2}(v) : \exists(u,z,\perp,list',hs') \in KT_v \wedge (z \in N_v) \wedge (u \neq z) \wedge (u \neq v) \wedge (HS_z \neq CH)\ \wedge$
$\exists(u,\perp,\perp,list,hs) \in KT_z\ \wedge (hs' \neq hs)$
$\longrightarrow Update(u,z,\perp,list,hs);\ UpdatePIF(u,z,\perp,C);$

$\mathbf{R2_3}(v) : \exists(u,z,\perp) \in KT_v \wedge (z \neq \perp) \wedge \big(z \notin N_v \vee z = u \vee z = v \vee u = v\ \vee$
$HS_z = CH \vee (u,\perp,\perp) \notin KT_z\big) \longrightarrow Delete(u,z,\perp);$

$\mathbf{R2_4}(v) : \exists(u,z,\perp,list',hs') \in KT_v \wedge (z \in N_v) \wedge (u \neq z) \wedge (u \neq v) \wedge (HS_z \neq CH)\ \wedge$
$\exists(u,\perp,\perp,list,hs) \in KT_z\ \wedge (hs' = hs) \wedge (list' \neq list) \longrightarrow Update(u,z,\perp,list,hs);$

$\mathbf{R3_1}(v) : (HS_v \neq O) \wedge \exists w \in N_v \wedge (HS_w \neq CH) \wedge \exists(u,z,\perp,list,hs) \in KT_w\ \wedge$
$(u \neq v) \wedge (z \neq v) \wedge (z \neq \perp) \wedge (u,w,z) \notin KT_v \longrightarrow Insert(u,w,z,list,hs);$

$\mathbf{R3_2}(v) : (HS_v \neq O) \wedge \exists(u,w,z,list',hs') \in KT_v \wedge (w \in N_v) \wedge (u \neq v)\ \wedge$
$(z \neq v) \wedge (z \neq \perp) \wedge (HS_w \neq CH) \wedge \exists(u,z,\perp,list,hs) \in KT_w \wedge (hs' \neq hs)$
$\longrightarrow Update(u,w,z,list,hs);\ UpdatePIF(u,w,z,C);$

$\mathbf{R3_3}(v) : (HS_v \neq O) \wedge \exists(u,w,z) \in KT_v \wedge (w \neq \perp) \wedge (z \neq \perp) \wedge \big(w \notin N_v\ \vee$
$u = v \vee z = v \vee w = v \vee HS_w = CH \vee (u,z,\perp) \notin KT_w\big) \longrightarrow Delete(u,w,z);$

$\mathbf{R3_4}(v) : (HS_v \neq O) \wedge \exists(u,w,z,list',hs') \in KT_v \wedge (w \in N_v) \wedge (u \neq v)\ \wedge$
$(z \neq v) \wedge (z \neq \perp) \wedge (HS_w \neq CH) \wedge \exists(u,z,\perp,list,hs) \in KT_w \wedge (hs' = hs)\ \wedge$
$(list' \neq list) \longrightarrow Update(u,w,z,list,hs);$

$\mathbf{R3_5}(v) : (HS_v = O) \wedge \exists(u,w,z) \in KT_v \wedge (w \neq \perp) \wedge (z \neq \perp) \longrightarrow Delete(u,w,z);$

The rule $R3$ ensures that a leader or a pseudo-leader v maintains correct knowledge about clusters whose the head u (leader or pseudo-leader), is at distance 3 from v. v keeps (or adds) the record (u,w,z) to KT_v only if w (a v's neighbor) is not a cluster-head and if KT_w contains a record about the u's cluster (u is at distance 2 from w). An ordinary node removes all records about clusters whose the head is at distance 3 (rule $R3_5$).

By considering nearly ordinary nodes as gateways, and nearly cluster-heads as pseudo-leaders, the previous rules build larger tables than those required to have the completeness property. This feature is important in order to preserve the completeness property in spite of changes in the hierarchical structure.

VI. CNK PROTOCOL : SERVICE GUARANTEE MECHANISM

In this section, we present the mechanism that updates the variable *Ready* in order to preserve the completeness property in spite of reorganization of clusters.

A. How the variable Ready is updated ?

Due to an incorrect initial configuration, a node v may need to correct the value of $Ready_v$. If v is an ordinary node then $Ready_v$ has to be *RO* (the rule $RC_O(v)$ does the correction if necessary). Idem, if v is cluster-head then $Ready_v$ has to be *RCH* (the rule $RC_{CH}(v)$ does the correction if necessary).

$\mathbf{RC_O}(v) : (HS_v = O) \wedge (HS_v = Status_v) \wedge (Ready_v = RCH) \longrightarrow Ready_v := RO;$

$\mathbf{RC_{CH}}(v) : (HS_v = CH) \wedge (HS_v = Status_v) \wedge (Ready_v = RO) \longrightarrow Ready_v := RCH;$

Updating the variable *Ready* requires a careful study in two cases: (1) a nearly cluster-head that sets *Ready* to *RCH* and, (2) a nearly ordinary that sets *Ready* to *RO*.

According to the specification of completeness property, a nearly cluster-head v does not know leaders of its 3R-neighborhood, and it is not known by these leaders. However, once it sets $Ready_v$ to *RCH*, v may become leader at any moment by an action of the clustering protocol. In this new status, the node v must know and be known by all leaders of its 3R-neighborhood, otherwise the completeness property will be falsified. Therefore, before setting its variable $Ready_v$ to *RCH*, v should know the paths to leaders of its 3R-neighborhood, and these leaders should know the reverse paths leading to v. CNK allows v to update its variable $Ready_v$ only if this knowledge is established. This extra knowledge can be achieved only by a Propagation of Information with Feedback (PIF) within the v's 3-neighborhood.

A nearly ordinary node u also requires a propagation of information with feedback before setting its variable

$Ready_u$ to RO. Let us illustrate this feature by an example. Let a configuration c illustrated in Figure 1, and satisfying the completeness property where $CH4$ has the status NO ($Ready_{CH4} = RCH$). In c, $CH4$ knows a path to $CH2$ and to $CH3$; whereas, $CH2$ may not know a path to $CH3$ and vice versa. Once $CH4$'s $Ready$ is set to RO, $CH4$ may become ordinary at any time; and $CH2$ and $CH3$ should know each other. Otherwise, the completeness property will be falsified after clustering protocol changing of $CH4$'s status to ordinary. Therefore, the update of $Ready$ variable by a nearly cluster-head or a nearly ordinary node is associated to the end of a Propagation of Information with Feedback (PIF).

To implement the service guarantee mechanism, we adapt the PFC snap-stabilizing PIF algorithm [4] for topologies organized as oriented tree. The PFC algorithm is snap-stabilizing. Starting from any configuration, if the PIF process terminates, then all nodes have received the propagated information.

In PFC algorithm, each node behaves according to whether it is the root, an internal or a leaf of the tree. The root is a distinguished node, responsible to initiate the PIF process. Every tree node v maintains a PIF-state variable S_v, having three possible values $\{B,F,C\}$. B value indicates that v is in the broadcast phase, F value is associated to the feedback phase, and C value is associated to the cleaning phase. At least 3 states per node are required to achieve a PIF (proved in [4]). Any initiated PIF, terminates in $2h+1$ rounds (h is the height of the tree).

B. Adapted PFC algorithm

The key idea of our solution is to consider the set of knowledge tables as a forest of PIF-trees. A PIF-tree is a tree composed of records of a knowledge tables. Each record of KT is a node of one PIF-tree. The PIF-state of each record is the pif field of this record.

***Definition** 9 (**records of a PIF-tree**):*

- The record $(v,\perp,\perp)$ of KT_v is a root of a PIF-tree.
- The parent of $(v,\perp,\perp)$ of KT_z is the root record $(v,\perp,\perp)$ of KT_v.
- The parent of the record $(v,z,\perp)$ of KT_w is the record $(v,\perp,\perp)$ of KT_z.
- The parent of the record (v,w,z) in KT_u is the record $(v,z,\perp)$ of KT_w.

Notice that, the root of the record (v,w,z) of KT_u is $(v,\perp,\perp)$ of KT_v. Each node v may have at most one root record. The height of any PIF-tree is less or equal than 3; thus any PIF process requires at most 7 rounds.

***Definition** 10 (**Leaves of a PIF-tree**):*

- The record $(v,\perp,\perp)$ of KT_z is leaf if the node z is a cluster-head, or it does not have any descendant (i.e., $HS_z = CH \ \vee \ N_z/\{v\} = \emptyset$).
- The record $(v,z,\perp)$ of KT_w is a leaf if the node w is a cluster-head or all its descendant are ordinaries (i.e., $HS_w = CH \ \vee \ \forall u \in N_w/\{v,z\} : HS_u = O$).
- The record (v,z,w) of KT_u is always a leaf.

The adapted PFC algorithm is presented in Protocol 2. For all paths $v-z-w-u$, the structure of the PIF-tree rooted on v, according to definitions 9 and 10, is shown in Figure 4.

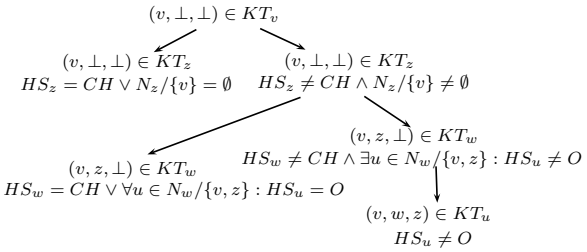


Figure 4. Structure of a PIF-tree rooted at $(v,\perp,\perp)$ of KT_v

The rules computing knowledge tables (i.e., $Ri, i \in [0,3]$) have priority over rules of PIF algorithm (except correction rules RC and IC). This priority ensures that when a node v participate to a PIF process by performing a PIF rule, v first gets all paths leading to leaders and pseudo-leaders known by its neighbors. Therefore, at the end of the PIF process, the root knows paths to all leaders and pseudo-leaders having participated to the PIF. This priority is established in the Algorithm by the predicate $Disabled$: a node v can perform a PIF rule only if $Disabled(v)$ is satisfied.

A root initiates a PIF by performing the broadcast rule RB. Nodes of the tree which are at distance 1 from the root participate to this phase by performing rule IB. This rule can be performed on a record only if it is leaf or all its descendants are in the cleaning state. The feedback is initiated by a node in two cases : if it is a leaf at distance 1 or 2 from the root (IF-$d1$ and IF-$d2$), or it is at distance 2 from the root but all its descendants have a record about the root (IF-$d2$).

At the end of the PIF, RF-$guard$ is satisfied by the root; thus, the rule RR_{CH} or RR_O is enabled. Once one rule is performed, $Ready$ is set to RCH or RO. Now, the clustering protocol may change the node status.

Notice that the adapted PFC algorithm has other rules: RC and IC rules. These rules set the pif field of a record to C. This action starts a cleaning phase in v's sub-tree: all nodes of its sub-tree will take the C state. The cleaning phase is used (1) to reset records of the PIF-tree, or (2) to abort the current PIF no more needed.

Protocol 2 : PIF Algorithm on node v

Note: Gi_j is the guard of rule Ri_j defined in section V.

Predicates:

Disabled$(v) \equiv \forall i \in [1,3], \neg G0_i(v) \wedge \neg Gi_1(v) \wedge \neg Gi_2(v)$

Constraint1$(v) \equiv \forall (z,\perp,\perp,hs_v,pif_v) \in KT_v \wedge (hs_v \neq NCH \vee pif_v \neq C)$
$\Rightarrow \forall u \in N_v/\{z\}, (z,v,\perp,hs_u,pif_u) \in KT_u \wedge (hs_u = hs_v) \wedge$
$(pif_v = C \vee pif_u \neq C)$

Constraint2$(v) \equiv \forall (z,w,\perp,hs,pif) \in KT_v \wedge (hs \neq NCH \vee pif \neq C)$
$\Rightarrow \forall u \in N_v/\{z,w\}, HS_u = O \vee (z,v,w) \in KT_u$

Constraint3$(v,u) \equiv \forall (z,\perp,\perp,hs_u,pif_u) \in KT_u \wedge$
$(hs_u \neq NCH \vee pif_u = F) \Rightarrow (z,u,\perp,hs_v,pif_v) \in KT_v \wedge$
$(hs_v \neq NCH \vee pif_v \neq C)$

Rules:

/* Initiating the broadcast by the root */

RB$(v) :: (v,\perp,\perp,C) \in KT_v \wedge \Big((HS_v = NCH \wedge Ready_v = RO) \vee$
$(HS_v = NO \wedge Ready_v = RCH)\Big) \wedge \big(\forall u \in N_v, (v,\perp,\perp,C) \in KT_u\big) \wedge$
$Disabled(v) \longrightarrow UpdatePIF(v,\perp,\perp,B)$;

/* Termination of the PIF and updating the *Ready* variable */

RF-Guard$(v) \equiv (v,\perp,\perp,hs_v,B) \in KT_v \wedge$
$(\forall u \in N_v, (v,\perp,\perp,hs_u,F) \in KT_u) \wedge (hs_v = hs_u) \wedge Disabled(v)$

RR$_{\mathbf{CH}}$ $(v) :: HS_v = NCH \wedge RF\text{-}Guard(v)$
$\longrightarrow Ready_v := RCH;\ UpdatePIF(v,\perp,\perp,F)$;

RR$_{\mathbf{O}}$ $(v) :: HS_v = NO \wedge RF\text{-}Guard(v) \wedge Constraint1(v) \wedge$
$Constraint2(v) \longrightarrow Ready_v := RO;\ UpdatePIF(v,\perp,\perp,F)$;

/*Participating to the Broadcast by nodes at distance 1 from the root*/

IB$(v) :: \exists (u,\perp,\perp,hs_v,C) \in KT_v \wedge u \in N_v \wedge (u,\perp,\perp,pif) \in KT_u \wedge$
$(pif \neq C) \wedge \Big(HS_v = CH \vee (\forall z \in N_v/\{u\} : (u,v,\perp,hs_z,C) \in KT_z \wedge$
$hs_z = hs_v)\Big) \wedge Disabled(v) \wedge Constraint2(v) \wedge Constraint3(v,u)$
$\longrightarrow UpdatePIF(u,\perp,\perp,B)$;

/*Propagation of Feedback by nodes at distance 2 and 1 from the root*/

IF-d1$(v) :: \exists (u,\perp,\perp,hs_v,B) \in KT_v \wedge u \in N_v \wedge (u,\perp,\perp,pif) \in KT_u \wedge$
$pif \in \{B,F\} \wedge \Big(HS_v = CH \vee (\forall z \in N_v/\{u\} : (u,v,\perp,hs_z,F) \in KT_z \wedge$
$hs_z = hs_v)\Big) \wedge Disabled(v) \wedge Constraint2(v) \wedge Constraint3(v,u)$
$\longrightarrow UpdatePIF(u,\perp,\perp,F)$;

IF-d2$(v) :: \exists (u,z,\perp,hs_v,C) \in KT_v \wedge z \in N_v \wedge (u,\perp,\perp,pif) \in KT_z \wedge$
$pif \in \{B,F\} \wedge (HS_z \neq CH) \wedge \Big(HS_v = CH \vee$
$\forall w \in N_v/\{u,z\}, HS_w = O \vee ((u,v,z,hs_w) \in KT_w \wedge hs_w = hs_v)\Big) \wedge$
$Disabled(v) \longrightarrow UpdatePIF(u,z,\perp,F)$;

/* Correction rules : deal with incorrect initial configurations, and initiate the cleaning phase. */

RC$(v) :: (v,\perp,\perp,pif) \in KT_v \wedge \big((HS_v = CH \wedge pif \in \{B,F\}) \vee$
$(HS_v = NCH \wedge Ready_v = RO \wedge pif = F) \vee$
$(HS_v = NO \wedge Ready_v = RCH \wedge pif = F)\big) \longrightarrow UpdatePIF(v,\perp,\perp,C)$;

IC-d1$(v) :: \exists (u,\perp,\perp,hs_v,pif_v) \in KT_v \wedge (pif_v \neq C) \wedge (u \in N_v) \wedge$
$(u,\perp,\perp,hs_u,C) \in KT_u \wedge Disabled(v) \longrightarrow UpdatePIF(u,\perp,\perp,C)$;

IC-d2$(v) :: \exists (u,z,\perp,hs_v,pif_v) \in KT_v \wedge (pif_v \neq C) \wedge (z \in N_v) \wedge$
$(u,\perp,\perp,hs_z,C) \in KT_z \wedge Disabled(v) \longrightarrow UpdatePIF(u,z,\perp,C)$;

VII. Sketch proofs

The proof of self-stabilization with service guarantee of CNK protocol is omitted due to lack of space. All detailed proofs of convergence, closure and time complexity can be found in [14].

Since the leader definition is based on the *Status* variable. The completeness satisfiability depends on the value of both clustering and CNK protocol variables (*Status* and KT). Thus, the completeness property may be compromised by an action of the clustering protocol even if its was satisfied before this action. Let us illustrate this feature by an example based on a configuration c presented in Figure 1. Assume that in c, $v5$ is a nearly cluster-head, $Ready_{v5} = RCH$, and KT_{v5} does not contain any record at destination of $CH3$. In c, $v5$ is not leader, but $v5$ may become a leader at any time by an action of the clustering protocol (see Figure 2). The completeness property is satisfied in c; nevertheless it is no longer satisfied after that $v5$ becomes leader.
Therefore, to prove that CNK is self-stabilizing with service guarantee, we have to define a predicate that (1) is closed under any action of both clustering and CNK protocols, and (2) ensures the fulfilment of the completeness property. To achieve that, we define the Strong-Completeness predicate.

***Definition** 11 (**Quasi-leader, and Quasi-ordinary**):*

- $QL(v) \equiv (HS_v = CH) \vee (HS_v = NO) \vee$
$(HS_v = NCH \wedge Ready_v = RCH)$
- $QO(v) \equiv (HS_v = O) \vee (HS_v = NCH) \vee$
$(HS_v = NO \wedge Ready_v = RO)$

A node v that satisfies $QL(v)$, will be called quasi-leader, and a node v that satisfies $QO(v)$, will be called quasi-ordinary. The definitions of quasi-leader and quasi-ordinary nodes are given regardless the status of v within clustering protocol, i.e., the variable *Status*. Each quasi-leader (resp. quasi-ordinary) v, acts as a leader of cluster (resp. an ordinary node). Notice that a node v may be both quasi-leader and quasi-ordinary when v has a pending request to change its status. In this case, v is leader of its cluster, and it may be a gateway if necessary.

***Definition** 12 (**k-QR-neighborhood**):* The k-QR-neighborhood of a node v (for k quasi-restricted neighborhood), denoted QRN_v^k, is the set of nodes from v's k-neighborhood reached by a path where the gateway(s) is (resp. are) quasi-ordinary(ies).

***Definition** 13 (**Strong-Completeness predicate**):* The Strong-Completeness predicate is satisfied if and only if each quasi-leader knows all quasi-leaders within its 3-QR-neighborhood, and all paths leading to them.

The Strong-completeness predicate depends only on CNK protocol variables (it is uncorrelated on the variable *status*). Thus, no action of the clustering protocol can falsify or can satisfy the Strong-completeness property.

The sketch proof of convergence and closure is as follows. First, we present a predicate $P1$ defined on the clustering and CNK protocols variables. We prove that the set of configurations $SC1$, in which $P1$ is satisfied, is an attractor for both protocols. Furthermore, we prove that in a configuration of $SC1$, if the Strong-completeness predicate is satisfied then the completeness property is also satisfied. In a second time, we describe the Strong-completeness predicate involving only variables of the protocol CNK. We prove that the set of configurations $SC2$ (a subset of $SC1$), in which the Strong-completeness predicate is satisfied, is an attractor of the CNK protocol. Finally, we conclude that any computation of clustering and CNK protocols has a suffix where the completeness property (so, the minimal service) is always verified.

Time complexity. The convergence from any configuration to a configuration satisfying the Strong-completeness (so, the Completeness) property is achieved in at most 4 rounds.
A request from the clustering protocol (i.e. a node wanting to be cluster-head, or to be ordinary) is satisfied in at most 7 rounds: it is the number of rounds required to achieve the PIF process.
After the last modification of hierarchical status, 4 rounds are enough to reach a terminal configuration (that is also a legitimate configuration).

VIII. Simulation results & concluding remarks

In order to show the interest of self-stabilization with service guarantee approach, we study a comparison between a naive version of CNK (without service guarantee mechanism), denoted N-CNK, and the self-stabilizing with service guarantee version of CNK. Our simulation experiments are carried out thanks to the NS2.34 simulator. For the naive version N-CNK, the self-stabilizing clustering protocol [16] is used. Otherwise, the self-stabilizing with service guarantee clustering protocol used is [13]. Both clustering protocols are weight-based. They assume that each node is assigned a weight value, that can increase or decrease during time. The higher the weight of a node, the better appropriate this node is for the role of cluster-head. To achieve this feature in our simulation, we use the following model of weight variation.
Each node randomly chooses its initial weight w between two values $Wmin(=50)$ and $Wmax(=80)$. The node's weight changes according to a frequency $freq(=0.5)$, 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 $\Delta(=2)$. Thus, the new weight of a node is chosen randomly between $w-\Delta$ and $w+\Delta$.

Obviously, to achieve this study, CNK protocol was adapted to the message passing model. Each node v broadcasts periodically (once per 0.6 second) to its neighbors a message containing its state (its *status* and KT). Based on this message, v's neighbors decide to update their table or not. If within a certain period of time (1.85 second), no message is received from a node, this node is assumed to be no more neighbor.

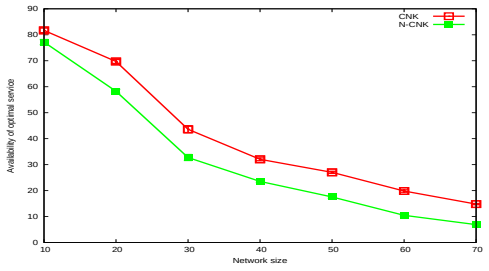
Our network is composed of mobile nodes, with a propagation radio range of $250m$, randomly placed within a $1700m * 1700m$ area. The average number of neighbors per node is equal to 5.

Mobility model. Each node moves randomly according to the *Random Waypoint model*. 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 (uniformly distributed between 0 m/s and 5 m/s). Upon reaching this destination, another random speed and destination are targeted after a pause time (0.5 second). The process is repeated until the simulation ends.

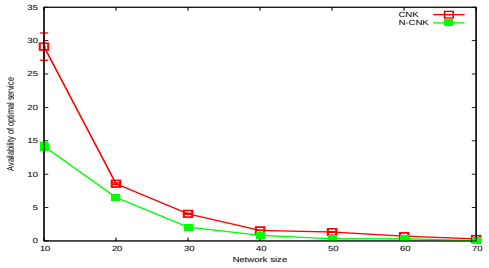
For both version of CNK, identical mobility and weight variation scenarios are used in order to gather fair results. Furthermore, to get accurate results, each simulation is driven with twenty different runs. Observed metrics are then averaged on these different runs. A confidence interval is also computed using the confidence level 95% (values are negligible; they do not appear in figures).

A. Availability of optimum service

The availability of optimum service is inversely proportional to the size of the network. In large scale network, the optimum service is rarely available in both protocols for both static and dynamic networks (Figures 5(a) and 5(b)). However, it stays more available in CNK than N-CNK protocol.
Due to node's weight change and mobility of nodes, the hierarchical structure is continuously reconstructed. As a result the member list of neighbor clusters changes very often, and some paths stored in knowledge tables may become invalid as soon as a node from this path becomes cluster-head or moves out its neighbors. Therefore, the optimum service is often broken. In this case, it is interesting to provide and maintain at least an useful minimal service.

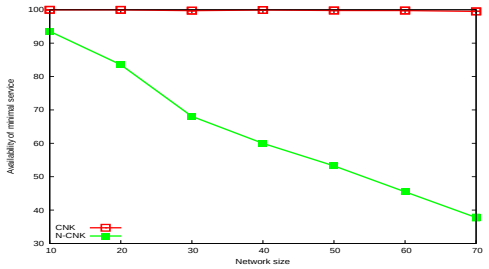


(a) Network with static nodes

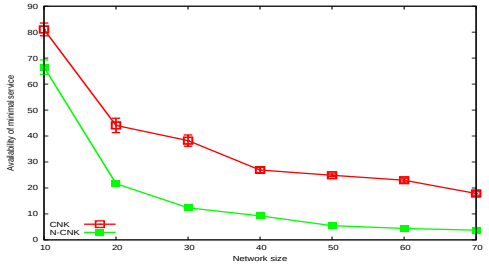


(b) Network with mobile nodes

Figure 5. Availability of optimum service



(a) Network with static nodes



(b) Network with mobile nodes

Figure 6. Availability of minimal service

B. Availability of minimum service

By increasing the network size, keeping nodes static (speed=0) but changing the hierarchical structure (freq=0.5), the minimal service (completeness property), once provided, is preserved by CNK protocol during all its execution time. In contrast, N-CNK protocol suffers from the absence of minimum service more and more when the network size increases. Indeed, in a network of 70 static nodes, the minimal service in N-CNK protocol is unavailable during 60% of time (Figure 6(a)).

The appearance of new neighbor clusters are handled in transparency by CNK protocol such that a leader has always a path to leaders of neighbor clusters. Whereas in N-CNK, a leader may have several neighbor clusters, but it does not know any path leading to their leaders. Due to the mobility of nodes, two clusters become neighbors as soon a path is created between their leaders. As mobility of nodes is unpredictable, the minimal service is affected in both protocols. However, the minimal service stays more available in CNK protocol (Figure 6(b)).

REFERENCES

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

[2] D. Bein, A. K. Datta, C. R. Jagganagari, and V. Villain. A self-stabilizing link-cluster algorithm in mobile ad hoc networks. In *ISPAN'05*, pages 436–441, 2005.

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

[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] Gruia Calinescu. Computing 2-hop neighborhoods in ad hoc wireless networks. In *ADHOC-NOW'03, springer LNCS 2865*, pages 175–186, 2003.

[6] A. Datta, S. Devismes, and L. Larmore. A self-stabilizing $o(n)$-round k-clustering algorithm. In *SRDS'09*, 2009.

[7] Murat Demirbas, Anish Arora, Vineet Mittal, and Vinodkrishnan Kulathumani. A fault-local self-stabilizing clustering service for wireless ad hoc networks. *IEEE Transactions on Parallel and Distributed Systems*, 17:912–922, 2006.

[8] S. Dolev and T. Herman. Superstabilizing protocols for dynamic distributed systems. *Chicago J. Theor. Comput. Sci.*, 1997, 1997.

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

[10] V. Drabkin, R. Friedman, and M. Gradinariu. Self-stabilizing wireless connected overlays. In *OPODIS'06, Springer LNCS 4305*, pages 425–439, 2006.

[11] M. Gairing, W. Goddard, S. T. Hedetniemi, P. Kristiansen, and A. A. McRae. Distance-two information in self-stabilizing algorithms. *Parallel Processing Letters*, 14(3-4):387–398, 2004.

[12] W. Goddard, S. T. Hedetniemi, D. P. Jacobs, and V. Trevisan. Distance-k knowledge in self-stabilizing algorithms. *Theoretical Computer Science*, 399:118–127, 2008.

[13] 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.

[14] C. Johnen and F. Mekhaldi. Self-stabilizing computation and preservation of knowledge of neighbor clusters. Technical Report N^{o}1537, LRI, 2010.

[15] C. Johnen and L. H. Nguyen. Self-stabilizing construction of bounded size clusters. In *ISPA'08*, pages 43–50, 2008.

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

[17] C. Johnen and S. Tixeuil. Route preserving stabilization. In *SSS'03, Springer LNCS 2704*, pages 184–198, 2003.

[18] S. Kamei and H. Kakugawa. A self-stabilizing approximation for the minimum connected dominating set with safe convergence. In *OPODIS'08, Springer LNCS 5401*, pages 496–511, 2008.

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