RAPPORT DE RECHERCHE

L R I

SELF-STABILIZING COMPUTATION AND PRESERVATION OF KNOWLEDGE OF NEIGHBOR CLUSTERS

JOHNEN C / MEKHALDI F

Unité Mixte de Recherche 8623
CNRS-Université Paris Sud – LRI

09/2010

Rapport de Recherche N° 1537

CNRS – Université de Paris Sud
Centre d'Orsay
LABORATOIRE DE RECHERCHE EN INFORMATIQUE
Bâtiment 490
91405 ORSAY Cedex (France)

Self-Stabilizing Computation and preservation of Knowledge of neighbor clusters

Colette Johnen[1] and Fouzi Mekhaldi[2]

[1] Université Bordeaux, LaBRI UMR 5800, F-33405 Talence, France
[2] Université Paris-Sud, LRI UMR 8623, F-91405 Orsay, France

Abstract

In this paper, we propose a self-stabilizing protocol computing and preserving the knowledge of neighbor clusters, called CNK protocol.

The cluster-heads maintain for each neighbor cluster: the identity of their head, the paths leading to them, and the list of members.
The most interesting property of CNK is the service guarantee during the stabilization phase. CNK protocol quickly provides the following minimal service: "each cluster-head knows the valid paths leading to the heads of its neighbor clusters". This service is provided after 4 rounds.

CNK protocol preserves the minimal service in spite of changes in the clusters structure (creation of new cluster, 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 a hierarchical routing protocol.

Keywords: Self-Stabilization, Service guarantee, Clustering, Knowledge of neighbor clusters.

Résumé

Le partitionnement en clusters est proposé dans les réseaux mobiles sans infrastructure pour améliorer leurs performances. Comme les protocoles de partitionnement sont adaptatifs aux changements topologiques, la structure hiérarchique produite sera dynamique : des clusters peuvent apparaître et disparaître au cours du temps. Par conséquent, tous les protocoles hiérarchiques doivent être également adaptatifs à ces changements.

Dans cet article, nous proposons un protocole de connaissance de voisinage des clusters, baptisé CNK. CNK permet à chaque leader de cluster de connaître tous ses clusters voisins : l'identité de leurs leaders, les chemins menant à eux, ainsi que la liste de leurs membres.

CNK est auto-stabilisant. De plus, il possède une propriété intéressante, à savoir la garantie de service pendant la stabilisation. Un service minimum utile est rapidement offert : "*un leader connaît l'identité de tous les leaders des clusters voisins, ainsi que des chemins menant à eux*". CNK préserve ce service minimum malgré l'occurrence des changements de la structure hiérarchique, tel que : la création et destruction de clusters, et changements de composition des clusters. La connaissance des clusters voisins est donc hautement disponible, ce qui assure la continuité de fonctionnement des protocoles de couches supérieures, comme le routage hiérarchique.

Mots-clés: Auto-stabilisation, Garantie de service, Partitionnement, Voisinage d'un cluster.

1 Introduction

A mobile Ad hoc or sensor network is a distributed multi-hop network which consists of mobile hosts that move arbitrarily, and communicate between them via wireless technologies without any pre-existing fixed infrastructure. The flat architecture on a large scale multi-hop network 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 problem of clustering is well studied in the literature, and several clustering protocols have been proposed in the context of multi-hop wireless networks. The problem studied here is not the clustering, but the knowledge of neighbour clusters assuming the existence of an under-layer clustering protocol. A survey on clustering protocols can be found on [1].

Motivation. Multi-level hierarchies is 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, the hierarchical packet forwarding is achieved from a cluster to a neighbor cluster, until reaching the destination cluster. Thus, routing packets between distant nodes requires also the knowledge of neighbor clusters. This knowledge is computed by our protocol CNK.

In another hand, since 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 need to be self-adaptive to modifications in the hierarchical structure.

For all these reasons, discovering and maintaining efficiently the neighbour of each cluster becomes a necessity. Moreover, as changes in the hierarchical structure occurs 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, CNK protocol is a self-stabilizing protocol; moreover it guarantees a minimal service.

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 a fault or perturbation in a well-defined manner. Self-stabilization is an approach to achieve the fault-tolerance against transient faults. A self-stabilizing protocol, regardless of its initial state, converges in finite time to a legitimate state, where 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 reconfiguration. Nevertheless, during all the convergence period, self-stabilizing protocols do not guarantee any property even if the perturbations could be handled in a safe manner. If some standard events occur very often, the availability and reliability for a self-stabilizing protocol may be compromised, unlike a self-stabilizing protocol with service guarantee.
A protocol is self-stabilizing with service guarantee if (1) an useful minimal service is quickly provided, and holds during progress of the protocol toward the optimum service (i.e., during convergence

to a legitimate configuration), and (2) this useful minimal service is still provided after multiple occurrences of some events, called *admissible events*. The minimal delay between occurrences of admissible events may be tiny. Nevertheless, the useful service has to be always provided. In this approach, events considered as admissible must be known, and taken into account during the design of the protocol to ensure that the useful minimal service is still guaranteed in spite of occurrences of admissible events. Whereas, the occurrence of inadmissible or unknown events, is handled by the self-stabilization mechanism.

Contribution. We propose a protocol for Clusters Neighbor Knowledge (CNK) that is self-stabilizing with guarantee of service.
CNK protocol needs a 1-hop clustering protocol, that provides to each node information about the current organization (i.e., its hierarchical status and the identity of its cluster-head). 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 a 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 goal of CNK protocol is not to converge quickly to a safe configuration where a 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 a 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 the occurrences of these events is unbounded (it depend on the clustering protocol and on the network topology dynamism). Therefore, the list of the admissible events by CNK protocol is the follow: (1) selection of a new cluster-head, (2) resignation of a cluster-head, and (3) switching of an ordinary node from a cluster to another one.
The admissible events are handled by CNK protocol, in such a way that the following useful service is preserved: *a head of cluster knows heads of all neighbor clusters and a valid path leading to them.* To ensure this useful service, CNK protocol requires of the clustering protocol two properties described in section 4. Some clustering protocols provide these two properties [2,3]. In hierarchical networks, the packet forwarding 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 routing protocols, to perform correctly their tasks. Nevertheless, this service is not optimal, because a head may store some invalid paths.

Related Works. The self-stabilization with service guarantee is related to the fault-containment [4], 1-strong self-stabilization [5], super-stabilization [6], robust self-stabilization [7,2,3] and the safe convergence [8,9]. The fault-containment, confines the effect of a single fault to the constant-distance neighborhood of the faulty nodes. So, the remaining part of the system behaves correctly. A 1-strong self-stabilizing algorithm guarantees that a single memory corruption fault cannot be propagated, and the next system transition leads to a legitimate configuration. A super-stabilizing algorithm ensures a safety property after one perturbation from a legitimate configuration. A protocol robust or having a "safe convergence” quickly reaches a safe configuration where a minimal service is provided. The safety property is preserved during the convergence to a legitimate configuration. Moreover, the safety property is also preserved after the occurrences of admissible events, in case of a robust self-stabilizing protocol.
Many clustering algorithms have been proposed in the literature for ad-hoc networks. A large number of these algorithms are self-stabilizing [10,11,12,13,14,15,16]. Robust self-stabilizing clustering algorithms [2,3], and self-stabilizing with safe convergence [8,9] are also proposed.

Algorithms building neighborhood knowledge in flat architectures are presented in [17,18,19]. In [17], algorithms computing 2-hops neighborhood in wireless networks are presented (they are based on geographic position or distance). In [18], 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 [19], 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 knowledge of neighbour clusters is assumed in many distributed protocols for hierarchical routing and multi-levels clustering, such as [20]. However, as far as we know, there is not protocol computing and preserving in a distributed manner the knowledge of neighbor clusters.

The rest of the paper is organized as follows. In section 2, we describe the model. The specification of knowledge of neighbor clusters problem is defined in section 3. In section 4, we present the interaction required between the clustering protocol and CNK protocol in order to ensure the self-stabilization with service guarantee of CNK. The two modules of CNK: computation of knowledge tables, and service guarantee mechanism are presented respectively in sections 5 and 6. In section 7, we illustrates the functioning of the CNK protocol. The detailed proofs are provided in sections 8 and 9. Finally, we conclude in Section 10.

2 Model

A distributed system S is modeled by an undirected graph. A graph G is defined by $(V,\ E)$ in which, V is the set of (mobile) nodes and E is the set of edges. There is an edge $(u, v) \in E$, if and only if u and v can communicate between them (links are bidirectional). If $(u, v) \in E$, we say that u and v are neighbors. N_v is the set of neighbors (the neighborhood) of the node v: $N_v = \{u \in V \mid (u, v) \in E\}$. The internal nodes among a path connecting two nodes u and v, are called gatways.

We use the *state model* of computation. Each node has an identifier ID. 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 on v. A node is said to be enabled if at least one of its rules is enabled. In a *terminal configuration*, no node is enabled.
The evaluation of the rule guard and the action performing is done in an atomic step. During the *computation step* $c_i \to c_{i+1}$, one or several enabled nodes perform an atomic step from c_i to reach c_{i+1}. 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 is *fair*, if for any node v that is continuously enabled along this computation, eventually performs an action. In this paper, we study only fair computations. We note by $Conf$ the set of all configurations, and by $\mathcal{E}$ the set of all fair computations. The set of fair computations starting from a particular configuration $c \in Conf$ is denoted $\mathcal{E}_c$. $\mathcal{E}_A$ is the set of fair computations whose the initial configuration belongs to $A \subset Conf$.
A node v is neutralized in the computation step cs, $c_i \to c_{i+1}$, if v is enabled in c_i and not enabled in c_{i+1}, but did not execute any action during cs.
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 neutralized along e_1. Let e_2 be the suffix of e such that $e = e_1e_2$. The second round of e is the first round of e_2, and so one. The round complexity of a computation is the number of disjoint rounds in the 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:***
  $\forall e \in \mathcal{E}_{B_1}(e = c_1, c_2, ...), \exists i \geqslant 1 : c_i \in B_2.$
  $\forall c \in B_1,$ *If* $(\mathcal{E}_c = \emptyset)$ *then* $c \in B_2$.
- ***Closure:*** $\forall e \in \mathcal{E}_{B_2}(e = c_1, ...), \forall i \geqslant 1 : c_i \in B_2.$

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

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

Definition 3. ***(Stabilization with service guarantee)***. *Let $\mathcal{P}$ be the predicate that stipulates the minimal service. Let $\mathcal{AE}$ be a set of admissible events that can occur in the system. A self-stabilizing protocol is said with service guarantee under $\mathcal{AE}$ if and only if the set of configurations satisfying $\mathcal{P}$ is:*

- *closed under any computation step.*
- *closed under any admissible event of $\mathcal{AE}$.*

Definition 4. ***(Admissible events)***. *The set of admissible events $\mathcal{AE}$ handled by CNK protocol are the following actions of clustering protocol:*

- *Selection of a new cluster-head.*
- *Resignation of a cluster-head.*
- *Switching of a node from a cluster to another one.*

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

In this paper, we focus on the knowledge of neighbor clusters problem. 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 v and u 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.

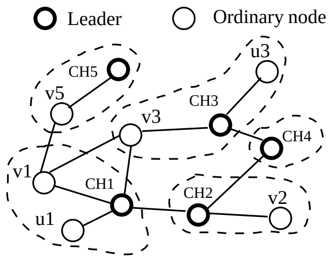

Fig. 1. Clustered network example

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

Definition 6. ***The*** k***R-neighborhood*** *of a node v (for k restricted neighborhood), denoted RN_v^k, is the set of nodes in v's k-neighborhood reachable by at least a path in which the gateway(s) is (resp. are) not leader(s). Let $Leader(v)$ be a predicate that stipulates whether the node v is leader (see Definition 8 in section 4). RN_v^k is defined by induction as follows:*

$$\begin{cases} RN_v^1 = N_v \\ RN_v^{k+1} = RN_v^k \ \cup \ \{u \in V \mid \exists z \in RN_v^k : \neg Leader(z) \wedge u \in N_z\} \end{cases}$$

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:

1. The definition of minimum service: the minimum service is the completeness property.
2. The list of admissible events: all actions done by the clustering protocol, including selection and resignation of a leader, and switching of an ordinary node from a cluster to another one.

Let us study the hierarchical structure presented in figure 1. We assume that the completeness property is verified: $CH1$ stores the paths leading to $CH2$, $CH3$, and $CH5$, but not paths leading to $v2$, $u3$, and $CH4$. The clustering protocol changes the clustering structure: $u3$ becomes leader and $CH3$ becomes ordinary. Now, $CH1$ has two neighbor cluster-heads: $u3$ and $CH4$, but it does know any paths leading to these nodes. The specification of the problem is violated: the completeness is no more satisfied. To avoid this situation, the CNK protocol bridles the occurrences of the admissible events to prevent the violation of the completeness property. Thus, proper interactions between CNK and the clustering protocol are required to maintain the cluster neighborhood knowledge during changes in the hierarchical organization.

4 Requirements on the clustering protocol

As said previously, CNK assumes the existence of an under-layer 1-hop clustering protocol. CNK is not based on a particular protocol. More precisely, CNK does not need to know the election, resignation and affiliation criteria of clustering protocol. However, 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 protocol. Some 1-hop clustering protocols already follow the two properties [2,3]. For other self-stabilizing protocols, we believe that is feasible to design a transformer building a robust self-stabilizing version of a 1-hop clustering protocol compatible with CNK protocol.

Proper interactions. (illustrated in Figure 2)
The variable *status* indicates the hierarchical status of a node, is updated only by the clustering protocol (this variable value is an input of the CNK protocol). conversely, the variable *Ready* is updated only by the CNK protocol (this variable value is an input of 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* ($Status_v = NCH$).
A cluster-head wanting to resign its role, takes the nearly ordinary status (NO). Whereas, an ordinary node wanting to become cluster-head, takes the nearly cluster-head status (NCH).

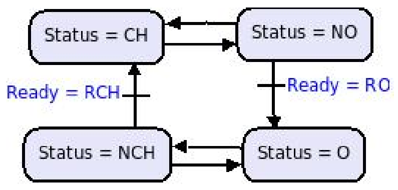

Fig. 2. Proper interactions between clustering and CNK protocol.

By taking an intermediate status (NCH or NO), the clustering protocol "sends a request" to CNK protocol. Then, the clustering protocol waits the approval of the CNK protocol. This authorization "is communicated" by CNK protocol through the variable *Ready*. The value RO of $Ready_v$ indicates that v is ready to be ordinary, and the value RCH indicates that v is ready to be cluster-head. Only the updating of the *Ready* variables allows CNK protocol to ensure the preservation of completeness property.
For ordinary nodes the default value of *Ready* is RO, and for cluster-heads the default value is RCH.
A nearly cluster-head v can become cluster-head, only if $Ready_v = RCH$; 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 $Ready_v = RO$, but it may return to the cluster-head status at any moment (even if $Ready_v = RO$).

A nearly ordinary node v 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.
$Leader(v) \equiv Status_v \in \{CH, NO\}$.
A pseudo-leader v is a node having the nearly cluster-head status.
$Pseudo\text{-}Leader(v) \equiv Status_v \in \{NCH\}$.

5 CNK protocol : computation of the Knowledge Tables

5.1 Structure of Knowledge Table

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 $dest = x, g1 = y, g2 = z$.

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

Fig. 3. Scheme of Knowledge Table

The destination field of KT_v contains the identity of nodes whose the status is not *ordinary*. 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. The utility of pif field is discussed in section 6.
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's variables, and macros are presented in Protocol **1**. Notice that only the Clustering protocol updates the variable $Status$; and only the CNK protocol updates the variable HS.

Each leader uses the Knowledge Table of its neighbors, to compute its cluster neighborhood. Every table KT_v is updated by 4 rules. Each rule Ri 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). Each rule $Ri(v)$ ($i > 0$) allows the node v to have a record about clusters whose the head (leader and pseudo-leader) is at distance i from v.

By performing a rule $R0$, a leader or pseudo-leader v (1) updates, deletes or inserts the record in KT_v about its own cluster, i.e. the record $(v, \perp, \perp)$, (2) it modifies the value of its variable *Ready*, and/or (3) it corrects the value of its variable HS_v if necessary. - the value of HS_v should be similar

Protocol 1 : Variables and macros on node v.

Input variables (from the clustering layer)

$Status_v \in \{CH, O, NO, NCH\}$; The 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 4)

Shared variables

$HS_v \in \{CH, O, NO, NCH\}$; The status of v, local copy 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 $Ready_v := RO$;
```

```
$\mathbf{R0_1}(v)$ :: $(Status_v \neq O) \wedge (v, \perp, \perp) \notin KT_v \longrightarrow HS_v := Status_v;\ UpdateReady;\ Insert(v, \perp, \perp, Cluster_v, HS_v)$;
$\mathbf{R0_2}(v)$ :: $(Status_v \neq O) \wedge (v, \perp, \perp) \in KT_v \wedge (HS_v \neq Status_v)$
        $\longrightarrow HS_v := Status_v;\ UpdateReady;\ Update(v, \perp, \perp, Cluster_v, HS_v);\ UpdatePIF(v, \perp, \perp, C)$;
$\mathbf{R0_3}(v)$ :: $(Status_v = O) \wedge \big(HS_v \neq Status_v \vee (v, \perp, \perp) \in KT_v\big) \longrightarrow HS_v := Status_v;\ UpdateReady;\ Delete(v, \perp, \perp)$;
$\mathbf{R0_4}(v)$ :: $(Status_v \neq O) \wedge \exists (v, \perp, \perp, List, hs) \in KT_v \wedge (HS_v = Status_v) \wedge \big(List \neq Cluster_v \vee hs \neq HS_v\big)$
        $\longrightarrow Update(v, \perp, \perp, Cluster_v, HS_v)$;
```

to the value of $Status_v$ -. If necessary, the execution of $R0$ updates the pif, $list$ and hs fields of $(v, \perp, \perp) \in KT_v$.

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 whose the HS_v value is incorrect, 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 $R2(v)$ maintains the validity of records whose the destination is at distance 2 of v according 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.

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

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

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

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

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

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

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

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

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

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

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

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

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

The rule $R3$ ensures that a leader or a pseudo-leader v maintains correct knowledge about clusters whose the head u, 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 of w). An ordinary node removes record 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 algorithm builds larger tables than the table required to have the completeness property. This feature is important in order to preserve the completeness property during the resignation and election processes of the clustering protocol (i.e. during the move from NO status to O, and the move from NCH to CH status).

6 CNK protocol : Service Guarantee Mechanism

In this section, we present the mechanism used to preserve the completeness property by CNK protocol in spite of reorganization of the clusters.

6.1 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 $Ready_v$ value has to be RO (the rule $RC_O(v)$ does the correction). Idem, if v is cluster-head then $Ready_v$ value has to be RCH (the rule $RC_{CH}(v)$ does the correction).

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

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

The two following updating of *Ready* variable require a careful study: (1) a nearly cluster-head sets its variable *Ready* to RCH, or (2) a nearly ordinary node sets its variable *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 can be achieved only by a Propagation of Information with Feedback (PIF) within the v's 3-neighborhood. v initiates a wave (called the propagation wave): every node in v's 2-neighborhood receiving this wave forwards the wave to its neighbors. The termination of the propagation is detected by v via the feedback process. After the PIF initialization, at least 5 rounds are needed to terminate a PIF process in v's 3-neighborhood. During the propagation phase, leaders learn the paths toward v, whereas the feedback phase allows v to get the reverse paths.

A nearly ordinary node u requires also a propagation of information with feedback before setting its variable $Ready_u$ to RO. Let us illustrate this feature by an example. In Figure 1, the clustering protocol sets the status of the cluster-head $CH4$ to NO ($Ready_{CH4} = RCH$). In a configuration satisfying the completeness property, $CH4$ knows a path to $CH2$ and to $CH3$. Whereas, $CH2$ may not know a path to $CH3$ and vice versa. Once $CH4$'s *Ready* variable has the value *RO* - $CH4$ may become ordinary at any time -, $CH2$ and $CH3$ should know each other. Otherwise, the completeness property will be falsified after clustering protocol changing of the status of $CH4$ to ordinary. Thus, 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).

The PIF mechanism was introduced in [21]. To implement the service guarantee mechanism, we adapt the PFC snap-stabilizing PIF algorithm for oriented trees, presented in [22]. The PFC protocol is snap-stabilizing, i.e., starting from any configuration, it always behaves according to its specifications. More precisely, from any configuration, at the end of a PIF process, all nodes have received the propagated information.

The PFC algorithm is non-uniform. Each tree 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 Sv, having three possible values $\{B, F, C\}$. B value indicates that the node is in the broadcast phase, F value feedback is associated to the feedback phase, and the C value to the cleaning phase. At least, 3 states per node are needed to achieve a PIF (it is proved in [22]). Any initiated PIF, terminates in $2h + 1$ rounds (h is the height of the tree).

6.2 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 where each node is a record of a knowledge table. Each record of KT is a node of one and only one PIF-tree. The PIF-state of each record is stored in the pif field of this record.

Definition 9 (records of a PIF tree).

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

Notice that, the root of the record (u, w, z) of KT_v is $(u, \perp, \perp)$ of KT_u. 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 requires at most 7 rounds.

Definition 10 (Leaf of PIF-tree).

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

The adapted PFC algorithm is presented in Protocol 2, whereas the structure of the PIF-tree is shown in Figure 4.

Protocol 2 : PIF Algorithm on node v

Note: Gi_j is the guard of rule Ri_j defined in section 5.

Predicates:

```
**Disabled**$(v)$ :: $\forall i \in [1,3], G0_i(v) = False \wedge Gi_1(v) = False \wedge Gi_2(v) = False$

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

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

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

Rules:

```
/* Initiating the broadcast by the root */

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

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

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

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

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

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

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

/* Propagation of Feedback by nodes at distance 2 and 1 from the root. */
**IF-d1**$(v)$ :: $\exists (u, \perp, \perp, hs_v, B) \in KT_v \wedge u \in N_v \wedge (u, \perp, \perp, pif) \in KT_u \wedge pif \in \{B, F\} \wedge$
    $\big((HS_v = CH) \vee (\forall z \in N_v/\{u\} : (u, v, \perp, hs_z, F) \in KT_z \wedge hs_z = hs_v)\big) \wedge Disabled(v) = True \wedge$
    $Constraint2(v) \wedge Constraint3(v, u) \longrightarrow UpdatePIF(u, \perp, \perp, F)$;

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

/* 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**$(v)$ :: $\exists (u, \perp, \perp, hs_v, pif_v) \in KT_v \wedge (pif_v \neq C) \wedge (u \in N_v) \wedge (u, \perp, \perp, hs_u, C) \in KT_u \wedge Disabled(v) = True$
        $\longrightarrow UpdatePIF(u, \perp, \perp, C)$;

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

$$\begin{tikzcd}
 & (v,\perp,\perp) \in KT_v \arrow[dl] \arrow[dr] & & \\
\begin{array}{c}(v,\perp,\perp) \in KT_z \\ HS_z = CH \vee N_z/\{v\} = \emptyset\end{array} & & \begin{array}{c}(v,\perp,\perp) \in KT_z \\ HS_z \neq CH \wedge N_z/\{v\} \neq \emptyset\end{array} \arrow[dl] \arrow[dr] & \\
 & \begin{array}{c}(v,z,\perp) \in KT_w \\ HS_w = CH \vee \forall u \in N_w/\{v,z\} : HS_u = O\end{array} & & \begin{array}{c}(v,z,\perp) \in KT_w \\ HS_w \neq CH \wedge \exists u \in N_w/\{v,z\} : HS_u \neq O\end{array} \arrow[d] \\
 & & & \begin{array}{c}(v,w,z) \in KT_u \\ HS_u \neq O\end{array}
\end{tikzcd}$$

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

The rules computing the knowledge table (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 performs a PIF rule, v knows all paths leading to leaders and pseudo-leaders known by its neighbors. Therefore, at the end of the PIF, the root knows the path 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 the 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 the rule IB. This rule can be performed on a record only if the descendants of this record 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 not leaf at distance 2 from the root but all its descendants which are non ordinary 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. Upon the execution of the rule RR_{CH} or RR_O, the variable *Ready* is set to RCH or RO. Now, the clustering protocol may change the node hierarchical status.

Notice that the adapted PFC v's algorithm has other rules: RC and IC rules. These rules set the pif field of a record in KT_v 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.

7 Illustration of the functioning of CNK protocol

In this section, we illustrate the functioning of the two modules of CNK protocol : computing the knowledge tables, and the service guarantee mechanism.

7.1 Illustration of computing the knowledge tables

Let study the network example shown in Figure 5.
In the initial configuration (5.a), the network is partitioned into two clusters. Z and W are cluster-heads (i.e., $Status_Z = Status_W = CH$), whereas U and V are ordinaries (i.e., $Status_U = Status_V = O$). The node V affiliates with the Z's cluster, and U affiliates with the W's cluster. Assume that

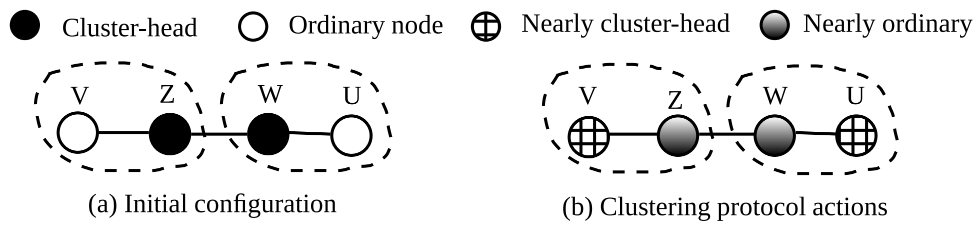

Fig. 5. An example to show the computation of knowledge tables and service guarantee mechanism

initially the knowledge tables are empty for every node. The computation of knowledge tables is done as follows:

• **In the first round** : the cluster-head Z (resp. W) performs the rule $R0_1$: It sets its variables HS_Z (resp. HS_W) to CH, and $Ready_Z$ (resp. $Ready_W$) to RCH, and it inserts in its knowledge table KT_W (resp. KT_Z) a record about its cluster.

• **In the second round** : all nodes are enabled, because the rule $R1_1$ is enabled for every node. During this round, the nodes U and W get a record about the cluster of Z. Similarly, the nodes Z and V get a record about the cluster of W.

The reached configuration satisfies the completeness and correctness properties; it is a legitimate configuration. The change in content of the knowledge tables during the two rounds is illustrated in the follwing table.

	KT_V	KT_Z	KT_W	KT_U
Initially				
Round 1		$HS_Z = CH$ $Ready_Z = RCH$ $(Z, \perp, \perp, CH, C)$	$HS_W = CH$ $Ready_W = RCH$ $(W, \perp, \perp, CH, C)$	
Round 2	$(Z, \perp, \perp, CH, C)$	$HS_Z = CH$ $Ready_Z = RCH$ $(Z, \perp, \perp, CH, C)$ $(Z, \perp, \perp, CH, C)$ $(W, \perp, \perp, CH, C)$	$HS_W = CH$ $Ready_W = RCH$ $(W, \perp, \perp, CH, C)$ $(W, \perp, \perp, CH, C)$ $(Z, \perp, \perp, CH, C)$	$(W, \perp, \perp, CH, C)$

7.2 Illustration of the service guarantee mechanism.

Starting from the configuration satisfying the completeness property obtained above, let us study the changes done by the clustering protocol shown in Figure 5.b. The nodes Z and W want to be ordinaries. They take the nearly-ordinary status (i.e., $Status_Z = Status_W = NO$). Similarly, U and V want to become cluster-heads; so, they take the nearly cluster-head status (i.e., $Status_U = Status_V = NCH$).

Assume that $Ready_Z = Ready_W = RCH$ and $Ready_U = Ready_V = RO$. The node Z (resp. W) can become ordinary only if $Ready_Z = RO$ (resp. $Ready_W = RO$). As the same, U (resp. V) can become cluster-head only if $Ready_U = RCH$ (resp. $Ready_V = RCH$).

In follows we illustrate step by step, the actions done by nodes in order to update the value of *Ready*.

• **In round 1** : The nearly cluster-head V (resp. U) performs the rule $R0_1$ and adds a record about its pseudo-cluster to its table KT. The nearly ordinary node Z (resp. W) performs the rule $R0_2$ to set the hs field of the record about its cluster to NO.

• **In round 2** : The nearly cluster-head V (resp. U) performs the two rules $R1_2$ and $R2_1$. It sets to NO the hs field of the record about Z's (resp. W's) cluster (rule $R1_2$), and adds a record about the W's (resp. Z's) cluster taking Z (resp. W) as first gateway (using the rule $R2_1$). The nearly ordinary Z (resp. W) performs the two rules $R1_1$ and $R1_2$. It sets to NO the hs field of the record about W's (resp. Z's) cluster (rule $R1_2$), and adds a record about the V's (resp. U's) cluster (using the rule $R1_1$).

• **In round 3** : The nearly cluster-head V (resp. U) performs the two rules $R2_2$ and RB. It sets to NO the hs field of the record about W's (resp. Z) cluster (rule $R2_2$), and it initiates the PIF process by updating the pif field of the record of V's (resp. U's) cluster to B. The nearly ordinary Z (resp. W) performs the two rules $R2_1$ and RB. It adds a record about the U's (resp. V's) cluster taking W (resp. Z) as gateway (rule $R2_1$), and it starts a PIF process by updating the pif field of the record of Z's (resp. W's) cluster to B.

• **In round 4** : The nearly cluster-head V (resp. U) performs the two rules $R3_1$ and $IF\text{-}d1$. It adds a record about the U's (resp. V's) cluster taking Z (resp. W) as first gateway and W (resp. Z) as second gateway (rule $R3_1$). Furthermore, it participates to the PIF process by returning the feedback to Z (resp. W) using the rule $IF\text{-}d1$.
The nearly ordinary Z (resp. W) performs the rule IB twice to participate to the PIF processes started by V and W (resp. U and Z).

• **In round 5** : All the nodes perform the rule $IF\text{-}d2$. The nearly cluster-head V (resp. U) initiates the feedback towards W (resp. Z); whereas, the node Z (resp. W) initiates the feedback towards U (resp. V).

• **In round 6** : Only the nodes Z and W perform the rule $IF\text{-}d1$. The nearly ordinary Z participates to the feedback towards W and V; whereas, the node W participates to the feedback towards Z and U.

• **In round 7** : Each node is able to achieve its PIF process, and then updates its variable *Ready*. The nearly ordinary Z (resp. W) terminates its PIF by performing the rule RR_O. By this action, the variable $Ready_Z$ (resp. $Ready_W$) is set to RO. The nearly cluster-head V (resp. U) terminates its PIF by performing the rule RR_{CH}, and then it sets the variable $Ready_V$ (resp. $Ready_U$) to RCH.

The change in content of the knowledge tables during the 7 rounds is illustrated in the following table.
After the round 7, the clustering protocol is authorized to change the hierarchical status of nodes V and U to CH, and those of Z and W to O. During these rounds, and after the hierarchical status change done by the clustering protocol the completeness property stays preserved.

8 Strong-Completeness predicate

The completeness satisfiability depends on the value of both clustering protocol and CNK protocol variables (*Status* and KT). Thus, the completeness property may be compromised by an action

	KT_V	KT_Z	KT_W	KT_U
Round 1	$(Z, \perp, \perp, CH, C)$ $(V, \perp, \perp, NCH, C)$	$(Z, \perp, \perp, NO, C)$ $(W, \perp, \perp, CH, C)$	$(Z, \perp, \perp, CH, C)$ $(W, \perp, \perp, NO, C)$	$(W, \perp, \perp, CH, C)$ $(U, \perp, \perp, NCH, C)$
Round 2	$(Z, \perp, \perp, NO, C)$ $(W, Z, \perp, CH, C)$ $(V, \perp, \perp, NCH, C)$	$(Z, \perp, \perp, NO, C)$ $(W, \perp, \perp, NO, C)$ $(V, \perp, \perp, NCH, C)$	$(Z, \perp, \perp, NO, C)$ $(W, \perp, \perp, NO, C)$ $(U, \perp, \perp, NCH, C)$	$(Z, W, \perp, CH, C)$ $(W, \perp, \perp, NO, C)$ $(U, \perp, \perp, NCH, C)$
Round 3	$(Z, \perp, \perp, NO, C)$ $(W, Z, \perp, NO, C)$ $(V, \perp, \perp, NCH, B)$	$(Z, \perp, \perp, NO, B)$ $(W, \perp, \perp, NO, C)$ $(V, \perp, \perp, NCH, C)$ $(U, W, \perp, NCH, C)$	$(Z, \perp, \perp, NO, C)$ $(W, \perp, \perp, NO, B)$ $(U, \perp, \perp, NCH, C)$ $(V, Z, \perp, NCH, C)$	$(Z, W, \perp, NO, C)$ $(W, \perp, \perp, NO, C)$ $(U, \perp, \perp, NCH, B)$
Round 4	$(Z, \perp, \perp, NO, F)$ $(W, Z, \perp, NO, C)$ $(V, \perp, \perp, NCH, B)$ (U, Z, W, NCH, C)	$(Z, \perp, \perp, NO, B)$ $(W, \perp, \perp, NO, B)$ $(V, \perp, \perp, NCH, B)$ $(U, W, \perp, NCH, C)$	$(Z, \perp, \perp, NO, B)$ $(W, \perp, \perp, NO, B)$ $(V, Z, \perp, NCH, C)$ $(U, \perp, \perp, NCH, B)$	$(Z, W, \perp, NO, C)$ $(W, \perp, \perp, NO, F)$ (V, W, Z, NCH, C) $(U, \perp, \perp, NCH, B)$
Round 5	$(Z, \perp, \perp, NO, F)$ $(W, Z, \perp, NO, F)$ $(V, \perp, \perp, NCH, B)$ (U, Z, W, NCH, C)	$(Z, \perp, \perp, NO, B)$ $(W, \perp, \perp, NO, B)$ $(V, \perp, \perp, NCH, B)$ $(U, W, \perp, NCH, F)$	$(Z, \perp, \perp, NO, B)$ $(W, \perp, \perp, NO, B)$ $(V, Z, \perp, NCH, F)$ $(U, \perp, \perp, NCH, B)$	$(Z, W, \perp, NO, F)$ $(W, \perp, \perp, NO, F)$ (V, W, Z, NCH, C) $(U, \perp, \perp, NCH, B)$
Round 6	$(Z, \perp, \perp, NO, F)$ $(W, Z, \perp, NO, F)$ $(V, \perp, \perp, NCH, B)$ (U, Z, W, NCH, C)	$(Z, \perp, \perp, NO, B)$ $(W, \perp, \perp, NO, F)$ $(V, \perp, \perp, NCH, F)$ $(U, W, \perp, NCH, F)$	$(Z, \perp, \perp, NO, F)$ $(W, \perp, \perp, NO, B)$ $(V, Z, \perp, NCH, F)$ $(U, \perp, \perp, NCH, F)$	$(Z, W, \perp, NO, F)$ $(W, \perp, \perp, NO, F)$ (V, W, Z, NCH, C) $(U, \perp, \perp, NCH, B)$
Round 7	$(Z, \perp, \perp, NO, F)$ $Ready_V = RCH$ $(W, Z, \perp, NO, F)$ $(V, \perp, \perp, NCH, F)$ (U, Z, W, NCH, C)	$(Z, \perp, \perp, NO, F)$ $Ready_Z = RO$ $(W, \perp, \perp, NO, F)$ $(V, \perp, \perp, NCH, F)$ $(U, W, \perp, NCH, F)$	$(Z, \perp, \perp, NO, F)$ $Ready_W = RO$ $(W, \perp, \perp, NO, F)$ $(V, Z, \perp, NCH, F)$ $(U, \perp, \perp, NCH, F)$	$(Z, W, \perp, NO, F)$ $Ready_U = RCH$ $(W, \perp, \perp, NO, F)$ (V, W, Z, NCH, C) $(U, \perp, \perp, NCH, F)$

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 v_5 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 the completeness property is no longer satisfied after that $v5$ becomes leader.
Therefore, we have to find a predicate that (1) is closed under any action of the clustering protocol and of the CNK protocol and that (2) ensures the fulfilment of the completeness property. To achieve that we define the Strong-Completeness predicate.

Definition 11 (Quasi-leader, and Quasi-ordinary). *Let the predicates,*
- $QL(v) \equiv (HS_v = CH) \vee (HS_v = NO) \vee (HS_v = NCH \wedge Ready_v = RCH)$
- $QO(v) \equiv (HS_v = O) \vee (HS_v = NCH) \vee (HS_v = NO \wedge Ready_v = RO)$

A node v satisfies $QL(v)$, will be called quasi-leader, and a node v that satisfies $QO(v)$, it 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. ***The k-QR-neighborhood*** *of a node v (for k quasi-restricted neighborhood), denoted QRN_v^k, is the nodes of v's k-neighborhood reached by a path where the gateway(s) is (resp. are) quasi-ordinary(ies). It is defined by induction as follows:*

$$\begin{cases} QRN_v^1 = N_v \\ QRN_v^{k+1} = QRN_v^k \ \cup \ \{u \in V \mid \exists z \in QRN_v^k : QO(z) \wedge u \in N_z\} \end{cases}$$

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 value depends only on the value of the CNK variables (i.e. it is uncorrelated of the values of the variables *status* and *Head*). Thus no action of the clustering protocol can falsify (or can satisfy) the Strong-completeness predicate.

9 Proofs of the stabilization with service-guarantee

The proof has three parts. First, we define the attractor $SC1$, subsection 9.1. We prove that if the Strong-completeness predicate is satisfied in a configuration of $SC1$, then the completeness property is verified. In the subsection 9.2, we present an attractor $SC2$ (a subset of $SC1$) in which the Strong-completeness predicate is always satisfied. In the last subsection, we conclude that any computation of clustering and CNK protocols has a suffix where the completeness property is always verified

9.1 $SC1$ attractor

Lemma 1. *$I0 = \{c \in Conf \mid \forall v \in V : (HS_v \neq CH \vee Ready_v = RCH) \wedge (HS_v \neq O \vee Ready_v = RO)\}$ is closed under any action of the clustering and CNK protocol.*

Proof. Let c_1 be a configuration of $I0$. Assume that a computation step cs exists: $c_1 \xrightarrow{cs} c_2$, such that $c_2 \notin I0$. We will prove that cs does not exist. According to our assumptions, in c_2, there is at least a node v which satisfies $(HS_v = CH \wedge Ready_v = RO) \vee (HS_v = O \wedge Ready_v = RCH)$. Two cases are possible:

- **Case 1: In c_1, $HS_v \neq CH \wedge HS_v \neq O$ is satisfied.**
As long as $HS_v \neq CH \wedge HS_v \neq O$ is satisfied, the configuration belongs to $I0$. Let during cs, v updates HS_v to the CH or O value. The only rules can update HS_v value are $R0_1(v)$, $R0_2(v)$ and $R0_3(v)$. However, these rules update also the $Ready_v$ value, using the macro *UpdateReady*. This macro ensures that when HS_v is set to CH, then $Ready_v$ is updated to RCH; and when HS_v is set to O, then $Ready_v$ is updated to RO. Therefore, c_2 cannot be reached from c_1.
- **Case 2: In c_1, $HS_v = CH \vee HS_v = O$ is satisfied.**
If during cs, the node v sets HS_v to NO or NCH, then the reached configuration belongs to $I0$. If the node v updates HS_v from CH (resp. O) to O (resp. CH), then the reached configuration belongs also to $I1$ (see proofs of Case 1).
Let during cs, v updates $Ready_v$ without updating HS_v. Notice that, the only rule can set $Ready_v$ to RO (resp. RCH) without updating HS_v is RR_O and RC_O (resp. RR_{CH} and RC_{CH}). The rules RR_O and RR_{CH} are disabled in c_1, because $HS_v \notin \{NCH, NO\}$. The execution of RC_O and RC_{CH} reaches a configuration of $I0$. Thus, c_2 cannot be reached from c_1.

The computation step cs does not exist. There is a contradiction. $I0$ is closed under any computation step. □

Lemma 2. *$I0$ is an attractor from $Conf$ in the composition of the clustering and CNK protocol.*

Proof. Let c be a configuration of $Conf$, but not of $I0$. In c, there is at least a node v which satisfies $(HS_v = CH \wedge Ready_v = RO) \vee (HS_v = O \wedge Ready_v = RCH)$. In c, two cases are possible:

- **Case 1: $Status_v = HS_v$.**
The node v does not need to adjust the value of HS_v. The rule $RC_O(v)$ or $RC_{CH}(v)$ (depending on whether $HS_v = O$ or $HS_v = CH$) is enabled, and it stays enabled while $Status_v = HS_v$ and $Ready_v$ is not updated. By fairness, the node v performs the rule $RC_O(v)$ or $RC_{CH}(v)$. After execution of this action, a configuration of $I0$ is reached, because $(HS_v = O \wedge Ready_v = RO)$ or $(HS_v = CH \wedge Ready_v = RCH)$ holds.
- **Case 2: $Status_v \neq HS_v$.**
Depending on the value of $Status_v$, and whether $(v, \perp, \perp)$ exists in KT_v or not, a rule among $R0_1$, $R0_2$ and $R0_3$ is enabled. This rule stays enabled while $Status_v \neq HS_v$. By fairness, the node v performs one rule; and two sub-cases may occur:
  - v sets HS_v to NCH or NO. So, a configuration of $I0$ is reached.
  - v sets HS_v to CH or O. As both rules $R0_1$, $R0_2$, $R0_3$ update $Ready_v$ using the macro *UpdateReady*. So, if HS_v is set to CH, then $Ready_v$ is updated to RCH, otherwise, $Ready_v$ is updated to RO. Thus, a configuration of $I0$ is also reached.

According these cases, a configuration of $I0$ is reached from c after any computation step. $I0$ is closed under any computation step (Lemma 3). Thus, $I0$ is an attractor from $Conf$. □

Observation 1

1. *In a configuration of $I0$, the rules RC_{CH} and RC_O are disabled for any node v.*
2. *After performing the rule RR_{CH} by a node v, v becomes (or stays) quasi-leader, i.e., $RR_{CH}(v)$ does not lead v to the non quasi-leader status.*
3. *After performing the rule RR_O by a node v, v becomes (or stays) quasi-ordinary.*
4. *The rules action except action of the rules $R0_1$, $R0_2$, $R0_3$, RR_O, and RR_{CH} by v do not modified the satisfiability of $QL(v)$ and $QO(v)$.*

Lemma 3. *$I1 = I0 \cap \{c \in Conf \mid \forall v \in V : Leader(v) \Rightarrow QL(v)\}$ is closed under any any action of the clustering and CNK protocol.*

Proof. Let c_1 be a configuration of $I1$. Assume that a computation step cs exists: $c_1 \xrightarrow{cs} c_2$, such that c_2 does not belong to $I1$. We will prove that cs does not exist. According to our assumptions, in c_2, there is at least a node v which satisfies $Leader(v) \wedge \neg QL(v)$. Two cases are possible:

- **Case 1: In c_1, v is leader**. As consequence, v is also quasi-leader.
If during cs, the node v performs a clustering rule and becomes non leader, then the reached configuration belongs to $I1$. So, to reach c_2, v remains leader, but it becomes non quasi-leader. This change cannot be done by the rules $RC_{CH}(v)$, $RC_O(v)$, $RR_O(v)$ and $RR_{CH}(v)$ (see Observation 1). Furthermore, the rules $R0_1(v)$, $R0_2(v)$, and $R0_3(v)$ always set HS_v to $Status_v$. As in c_1, v is leader. Thus, during cs, any update of HS_v ensures that $HS_v \in \{CH, NO\}$, i.e. $QL(v)$, holds in c_2. There is a contradiction.

- **Case 2: In c_1, v is non leader** (i.e. $Status_v = O \vee Status_v = NCH$). Two sub-cases are possible:
  - **In c_1, v is non quasi-leader** (i.e., $HS_v = O \vee (HS_v = NCH \wedge Ready_v = RO)$).
  During cs, v has to perform a clustering rule to become leader, but it should stay non quasi-leader. Notice that in c_1, if $HS_v = O$ then $Ready_v = RO$ ($c_1 \in I0$). Thus, in c_1, $Ready_v = RO$ whatever the value of HS_v. As long as $Ready_v = RO$, no clustering rule can set the variable $Status_v$ to CH, and so to NO (see Figure 2). So, v cannot become leader as long as v is a non quasi-leader. There is a contradiction.
  - **In c_1, v is quasi-leader** (i.e., $HS_v \in \{NO, CH\} \vee (HS_v = NCH \wedge Ready_v = RCH)$).
  As long as v is quasi-leader, the configuration belongs to $I1$. In order to reach c_2, the node v should change simultaneously its status from non leader to leader, and from quasi-leader to non quasi-leader.
  According to Observation 1, during cs, the node v did not perform the rules RC_{CH}, RC_O, RR_{CH} and RR_O. The only rules may be performed by v during cs, are the clustering rules and the rules $R0_1$, $R0_2$.
  In c_1, if $Status_v \neq NCH \vee Ready_v \neq RCH$, the node v cannot become leader by performing a clustering rule (see figure 2).
  Assume that $Status_v = NCH \wedge Ready_v = RCH$ is satisfied in c_1. The rule $R0_3(v)$ is disabled in c_1, whereas the execution of rule $R0_1(v)$ or $R0_2(v)$ reaches a configuration where $HS_v = NCH \wedge Ready_v = RCH$, i.e., $QL(v)$ holds in c_2. Therefore, a configuration in which v is leader but non quasi-leader is not reached from c_1.

The computation step cs does not exist. We conclude that $I1$ is closed under any computation step. □

Lemma 4. *$I1$ is an attractor from $I0$.*

Proof. Let c be a configuration of $I0$, but not of $I1$. In c, there is at least a node v that satisfies $Leader(v) \wedge \neg QL(v)$.
As c belongs to $I0$, in c we have: $Status_v \in \{CH, NO\} \wedge HS_v \neq Status_v \wedge Ready_v = RO$.
If v performs a clustering action to become non leader, then the reached configuration belongs to $I1$.
In c, one rule among $R0_1(v)$ and $R0_2(v)$ is enabled, and it stays enabled as long as $HS_v \neq Status_v$.
By fairness, the node v performs this rule. Upon execution of this rule, HS_v is set to $Status_v$. So, the reached configuration belongs to $I1$, because v is quasi-leader ($HS_v \in \{NO, CH\}$).
According to that, a configuration of $I1$ is reached from c after a computation step. $I1$ is closed under any computation step (Lemma 3). Thus, $I1$ is an attractor from $I0$. $\square$

Lemma 5. *$SC1 = I1 \cap \{c \in Conf \mid \forall v \in V : \neg Leader(v) \Rightarrow QO(v)\}$ is closed under any any action of the clustering and CNK protocol.*

Proof. Let c_1 be a configuration of $SC1$. Assume that a computation step cs exists: $c_1 \xrightarrow{cs} c_2$, such that c_2 does not belong to $SC1$. We will prove that cs does not exist. According to our assumptions, in c_2, there is at least a node v which satisfies $\neg Leader(v) \wedge \neg QO(v)$. Two cases are possible:

- **Case 1: In c_1, v is non leader**. As consequence, v is also a quasi-ordinary.
If during cs, the node v becomes leader, then the reached configuration belongs to $SC1$. So, to reach c_2, v remains non leader, but it becomes non quasi-ordinary.
This change cannot be done by the rules $RC_{CH}(v)$, $RC_O(v)$, $RR_{CH}(v)$, and $RR_O(v)$ (see Observation 1). So, during cs, v updates HS_v to be non quasi-ordinary using one rule among $R0_1(v)$, $R0_2(v)$, and $R0_3(v)$. However, these rules set HS_v to $Status_v$. As in c_1, v is non leader. Thus, any update of HS_v ensures that $HS_v \in \{O, NCH\}$, i.e. $QO(v)$ holds. According to that, v cannot become non quasi-ordinary as long as it is non leader.
- **Case 2: In c_1, v is Leader** (i.e., $Status_v \in \{NO, CH\}$).
As long as v is leader, the configuration belongs to $SC1$. Two sub-cases are possible:
  - **In c_1, v is a non quasi-ordinary** (i.e., $HS_v = CH \vee (HS_v = NO \wedge Ready_v = RCH)$). In order that v reaches c_2, v must stay non quasi-ordinary, but it changes its status from leader to a non leader by performing a clustering rule.
  Notice that in c_1, if $HS_v = CH$ then $Ready_v = RCH$ ($c_1 \in I0$). Thus, in c_1, $Ready_v = RCH$ whatever the value of HS_v. According to the status transition diagram shown in figure 2, as long as $Ready_v = RCH$, no clustering rule can change the $Status_v$ value to O, and so to NCH. So, v cannot become non leader as long as v is a non quasi-ordinary. Thus, c_2 is not reached from c_1.
  - **In c_1, v is quasi-ordinary** (i.e. $HS_v = O \vee HS_v = NCH \vee (HS_v = NO \wedge Ready_v = RO)$).
  As long as v is quasi-ordinary, the configuration belongs to $SC1$. So, to reach c_2, the node v simultaneously changes its status from leader to a non leader, and from quasi-ordinary to non quasi-ordinary.
  During cs, the node v did not perform a rule among RC_{CH}, RC_O, RR_{CH}, and RR_O (see Observation 1). So, the only rules performed by v during cs are both a clustering rule, and one rule from $R0_1(v)$, $R0_2(v)$ and $R0_3(v)$.
  In c_1, if $Status_v \neq NO \vee Ready_v \neq RO$ is satisfied, the node v cannot become non leader by performing a clustering rule (see Figure 2).
  Let in c_1, $Status_v = NO \wedge Ready_v = RO$. In c_1, the rule $R0_3(v)$ is disabled. Moreover, if the rule $R0_1(v)$ or $R0_2(v)$ is performed, even concurrently with a clustering rule, HS_v is set to $Status_v$.

So, $QO(v)$ holds in c_2. According to that, a configuration in which v is non leader and non quasi-ordinary is not reached from c_1.

According to these cases, the computation step cs does not exist. Thus, $SC1$ is closed under any computation step. □

Lemma 6. *$SC1$ is an attractor from $I1$.*

Proof. Let c be a configuration of $I1$, but not of $SC1$. In c, there is a node v which satisfies $\neg Leader(v) \wedge \neg QO(v)$.
As c belongs to $I1$, in c we have: $Status_v \in \{O, NCH\} \wedge HS_v \neq Status_v \wedge Ready_v = RCH$.
If v becomes leader by performing a clustering rule, then the reached configuration belongs to $SC1$.
Assume that during e, v did not become leader. In c, one rule among $R0_1$, $R0_2$, $R0_3$ is enabled, and it stays enabled as long as $HS_v \neq Status_v$.
By fairness, the node v performs this action. After that, HS_v is set to $Status_v$. So, the reached configuration belongs to $SC1$, because v is quasi-ordinary ($HS_v \in \{O, NCH\}$).
According to that, a configuration of $SC1$ is reached from $I1$ after any computation step. $SC1$ is closed (Lemma 5). Thus, $SC1$ is an attractor from $I1$. □

Theorem 1. *Let c be a configuration of $SC1$, which satisfies the Strong-completeness predicate. The completeness property is also satisfied in c.*

Proof. In c, each quasi-leader knows all paths towards all quasi-leaders of its 3-QR-neighborhood.
In c, each leader v is a quasi-leader (Lemma 2). Thus, in c, each leader knows paths to all leaders of its 3-QR-neighborhood.
In c, each non leader is quasi-ordinary (Lemma 6). So, in c, the v's 3R-neighborhood belongs to the v's 3-QR-neighborhood. This proof can be done by induction on $k \in [1, 3]$ value; the inductive step is $\forall v \in V, RN_v^k \subseteq QRN_v^k$. We conclude that in c, each leader knows all paths to the leaders of its 3R-neighborhood.
Therefore, the completeness property is satisfied in c.

Observation 2 *In a configuration of $SC1$, the following results are obvious:*

- *If a node v is non quasi-leader, then v is non leader; i.e., $\neg QL(v) \Rightarrow \neg Leader(v)$ (Lemma 3).*
- *If a node v is non quasi-ordinary, then v is leader; i.e., $\neg QO(v) \Rightarrow Leader(v)$ (Lemma 5).*

Lemma 7. *In a configuration of $SC1$, only the rule RR_{CH} can lead a node from non quasi-leader status to the quasi-leader status.*

Proof. Let v be a node. Let c be a configuration of $SC1$, where v is non quasi-leader. As $c \in I1$, then v is non leader (see Observation 2). Starting from c, if v updates HS_v (using the rules $R0_1(v)$, $R0_2(v)$, and $R0_3(v)$), then it stays non quasi-leader.
Thus, only the update of $Ready_v$ to RCH using $RR_{CH}(v)$ rule, can lead v to the quasi-leader status.

Lemma 8. *In a configuration of $SC1$, only the rule RR_O can lead a node from non quasi-ordinary status to the quasi-ordinary status.*

Proof. Let v be a node. Let c be a configuration of $SC1$, where v is non quasi-ordinary. As $c \in SC1$, then v is leader (see Observation 2). Starting from c, if v updates HS_v (using the rules $R0_1(v)$, $R0_2(v)$, and $R0_3(v)$), then it stays non quasi-ordinary.
Thus, only the update of $Ready_v$ to RO using the rule $RR_O(v)$ can lead v to the quasi-ordinary status.

9.2 $SC2$ attractor

In follows, we define formally the Strong-completeness predicate, called SP. Thereafter, we present the attractor $SC2$ where SP is always satisfied.

Definition 14 (the Strong-completeness predicate SP).
- $SP \equiv \forall v \in V,\ Sp0(v)\ \wedge Sp1(v)\ \wedge Sp2(v)\ \wedge Sp3(v)$
- $Sp0(v) \equiv \neg QL(v)\ \vee\ (v,\perp,\perp) \in KT_v$
- $Sp1(v) \equiv \neg QL(v)\ \vee\ \forall z \in N_v,\ (v,\perp,\perp) \in KT_z$
- $Sp2(v) \equiv \forall (v,z,w) \in Path, \neg QL(v) \vee \neg QO(z) \vee (v,z,\perp) \in KT_w$
- $Sp3(v) \equiv \forall (v,z,w,u) \in Path, \neg QL(v) \vee \neg QO(z) \vee \neg QO(w) \vee \neg QL(u) \vee (v,w,z) \in KT_u$

Description of SP: The predicate SP stipulates the Strong-completeness property.
$\forall v \in V, Sp0(v)$ is satisfied if every quasi-leader v has a record about its own cluster in its KT_v.
$\forall v \in V, Spi(v) (i \in \{1,2\})$ is satisfied if every node v has in KT_v a record about quasi-leaders within its i-QR-neighborhood.
$\forall v \in V, Sp3(v)$ is satisfied if every quasi-leader v has in KT_v a record about quasi-leaders within its 3-QR-neighborhood. Note that, a non quasi-leader v satisfies forever $Spi(v), i \in [0,3]$. The predicate SP is satisfied if for every node v, $Spi(v), i \in [0,3]$ is satisfied.

9.2.1 Proof of SP at distance 0 and 1

Lemma 2 (lemma resp. 3) establishes that once a configuration of a $A0$ (resp. $A1$) is reached, each quasi-leader v (resp.every v's neighbor) have a record in its Knowledge Table (KT) about the v's cluster.

Lemma 9. *$A0 = SC1 \cap \{c \in Conf \mid \forall v \in V:\ Sp0(v)$ is satisfied $\}$ is closed under any computation step.*

Proof. Le c_1 be a configuration of $A0$. According to Lemma 7, starting from c_1, $RR_{CH}(v)$ is the only rule that leads a non quasi-leader v to the quasi-leader status. However, this rule is disabled as long as $(v,\perp,\perp) \notin KT_v$, because the guard $RF\text{-}guard(v)$ is not satisfied. On the other hand, $R0_3(v)$ is the only rule that removes the record $(v,\perp,\perp)$ from KT_v. However, this rule leads v to the non quasi-leader status, by setting HS_v to O.
Thus, starting from a configuration of $A0$, a configuration that does not belong to $A0$ can never be reached. Therefore, $A0$ is closed under any computation step. □

Theorem 2. *$A0$ is an attractor from $SC1$.*

Proof. Let c be a configuration of $SC1$, but not of $A0$. In the configuration c, there is at least a node v such that $SP0(v)$ is not satisfied, i.e., $QL(v) \wedge (v,\perp,\perp) \notin KT_v$.
In c, if $Status_v = O$, then $HS_v \neq Status_v$, because v is quasi-leader. So, $R0_3(v)$ is enabled.
Otherwise (i.e., $Status_v \neq O$), the rule $R0_1(v)$ is enabled (by assumption, $(v,\perp,\perp) \notin KT_v$). One rule among $R0_1(v)$ and $R0_3(v)$ is enabled till $QL(v) \wedge (v,\perp,\perp) \notin KT_v$. By fairness, the node v performs one of the two rules.
- After performing $R0_1(v)$, the record $(v,\perp,\perp)$ belongs to KT_v. The reached configuration belongs to $A0$.

• After performing $R0_3(v)$, HS_v is set to O. So, v becomes non quasi-leader. The reached configuration belongs also to $A0$.

Eventually, a configuration of $A0$ is reached. $A0$ is closed under any computation step (Lemma 9). Thus, $A0$ is an attractor from $SC1$. □

Lemma 10. *$A1 = A0 \cap \{c \in Conf \mid \forall v \in V :\ Sp1(v)$ is satisfied $\}$ is closed under any computation step.*

Proof. Let c_1 be a configuration of $A1$.
According to Lemma 7, starting from c_1, $RR_{CH}(v)$ is the only rule that leads a non quasi-leader v to the quasi-leader status. However, this rule is disabled while there exists a v's neighbor z such that $(v, \perp, \perp) \notin KT_z$, because the guard $RF\text{-}guard(v)$ is not satisfied.
Let z be a v's neighbor. $R1_3(z)$ is the only rule can remove the record $(v, \perp, \perp)$ from KT_z. However, in c_1, as long as v is quasi-leader this rule is disabled, because $(v, \perp, \perp) \in KT_v$ ($c_1 \in A0$). Thus, staring from c_1, all reached configurations belong to $A1$. □

Theorem 3. *$A1$ is an attractor from $A0$.*

Proof. Let c be a configuration of $A0$, but not of $A1$. In c, there is a node v such that $Sp1(v)$ is not satisfied, i.e.,$QL(v) \wedge \exists z \in N_v, (v, \perp, \perp) \notin KT_z$. As $c \in A0$ (i.e., $(v, \perp, \perp) \in KT_v$), then the rule $R1_1(z)$ is enabled in c. As $A0$ is closed, the rule $R1_1(z)$ stays enabled as long as $Sp1(v)$ is not satisfied. By fairness, the node z performs this action, and adds the record $(v, \perp, \perp)$ to KT_z. A configuration of $A1$ is reached. As $A1$ is closed under any computation step (Lemma 10); thus, $A1$ is an attractor from $A0$. □

9.2.2 Proof of SP at distance 2

Lemma 12 establishes that the hs field of the record stored by node a v in KT_v about its own cluster has always the same value as HS_v.

Lemma 11.
$A2 = A1 \cap \{c \in Conf \mid \forall v \in V :\ (v, \perp, \perp, hs, pif) \in KT_v \Rightarrow hs = HS_v \wedge HS_v \neq O\ \}$ is closed under any computation step.

Proof. Let c be a configuration of $A2$. The only rule that adds the record $(v, \perp, \perp, hs, pif)$ to KT_v is $R0_1(v)$. However, this rule ensures that $HS_v \neq O$, and it sets hs to HS_v. Furthermore, hs and HS_v are always simultaneously updated (see rules $R0_2(v)$ and $R0_4(v)$). Moreover, the only rule that sets HS_v to O is $R0_3(v)$, but this rule deletes the record $(v, \perp, \perp)$ from KT_v. Thus, starting from c, after any computation step the reached configuration belongs to $A2$. □

Lemma 12. *$A2$ is an attractor from $A1$.*

Proof. Let c be a configuration of $A1$, but not of $A2$. In c, there is at least a node v that satisfies $(v, \perp, \perp, hs, pif) \in KT_v \wedge (hs \neq HS_v \vee HS_v = O)$. In c, according to the value of $Status_v$ and HS_v, one rule among $R0_2(v)$, $R0_3(v)$, $R0_4(v)$ is enabled.
• If $Status_v = O$, the rule $R0_3(v)$ is enabled.
• If $Status_v \neq O$, then either $R0_2(v)$ or $R0_4(v)$ is enabled.

One among these rules stays enabled as long as $hs \neq HS_v$. By fairness, the node v performs a rule.

- After performing $R0_3(v)$, $(v, \perp, \perp)$ is removed from KT_v.
- After performing $R0_2(v)$ or $R0_4(v)$, hs is updated to HS_v, and $HS_v \neq O$.

In both cases, a configuration of $A2$ is reached. As $A2$ is closed (Lemma 11), thus $A2$ is an attractor from $A1$. □

To explain the following lemmas, let us define the notion of new, old and very old records.

Notation 1 (New, Old, very Old records) *A record of a knowledge table is said to be new if the value of its hs and pif fields are respectively NCH and C. Otherwise, it is called an old record (it has not recently inserted). A old record is said very old if $hs \neq NCH$ or $pif = F$.*

Lemma 14 establishes that each quasi-leader v has an old record about its own cluster in its knowledge table. In the following, we assume that v is a quasi-leader.

Lemma 13. $P3(v) \equiv QL(v) \Rightarrow (v, \perp, \perp, hs_v, pif_v) \in KT_v \wedge (hs_v \neq NCH \vee pif_v \neq C)$
$A3 = A2 \cap \{c \in Conf \mid \forall v \in V : \ P3(v)$ *is satisfied* $\}$ *is closed under any computation step.*

Proof. Let cs be a computation step $c_1 \xrightarrow{cs} c_2$, such that $c_1 \in A3$. In c_1, two cases are possible:

- v **is non quasi-leader.** The only rule can lead v to the quasi-leader status is $RR_{CH}(v)$ (Lemma 7). The execution of this rule during cs ensures that $(v, \perp, \perp, NCH, F) \in KT_v$ holds in c_2. Thus, c_2 belongs to $A3$.
- v **is quasi-leader.** By assumption, in c_1 we have $(v, \perp, \perp, hs_v, pif_v) \in KT_v \wedge$ $(hs_v \neq NCH \vee pif_v \neq C)$. In order that $c_2 \notin A3$, during cs v should remove the record $(v, \perp, \perp)$ from KT_v, or it should update the value of hs_v or pif_v.
  - **Deleting** $(v, \perp, \perp)$ **from** KT_v. As long as v is quasi-leader, it cannot delete the record $(v, \perp, \perp)$ from KT_v, because $c_1 \in A0$, and $A0$ is closed (Lemma 9).
  - **Updating of** hs_v. As $c_1 \in A2$, we have $HS_v \neq O$ and $hs_v = HS_v$; so, $R0_2(v)$ is the only rule can update hs_v. Note that $R0_2(v)$ sets pif_v to C. If during cs, hs_v is set to NO or CH, then $c_2 \in A3$.
    If during cs, hs_v is set to NCH then in c_1 $Status_v = NCH$ (i.e., v is non leader). So, in c_1, v is quasi-ordinary (Lemma 6). In addition, $R0_2(v)$ is performed during cs, only if $Status_v \neq HS_v$ in c_1. We conclude that in c_1, $HS_v \in \{O, NO\} \wedge Ready_v = RO$. In c_2, v is non quasi-leader, because $hs_v = NCH \wedge \ Ready_v = RO$. Thus, c_2 belongs to $A3$.
  - **Updating of** pif_v **only**. The configuration c_2 does not belong to $A3$ if $hs_v = NCH$ and $pif_v = C$. The rule that sets pif_v to C without changing the hs_v value, is $RC(v)$. As long as v is quasi-leader and $hs_v = NCH$ (so, $Ready_v = RCH$), this rule is disabled.

According to these cases, starting from c_1 and after any computation step, the reached configuration belongs to $A3$. Thus, $A3$ is closed under any computation step. □

Lemma 14. $A3$ *is an attractor from* $A2$.

Proof. Let c be a configuration of $A2$, but not of $A3$. In c, there is at least a node v that does not satisfy $P3(v)$, i.e., $QL(v) \wedge (v, \perp, \perp, hs_v, pif_v) \in KT_v \wedge hs_v = NCH \wedge pif_v = C$ (because $c_1 \in A0$). We prove that all computations starting from c reach a configuration of $A3$. In c, two cases are possible:

- **Case 1:** $HS_v \neq Status_v$. In c, v is enabled and it stays enabled up to the time where it performs $R0_2(v)$ or $R0_3(v)$. By fairness, the node v performs one of these actions. The reached configuration belongs to $A3$, because $hs_v \neq NCH$ or $HS_v = O$ (i.e., v is no longer quasi-leader).
- **Case 2:** $HS_v = Status_v$. While $HS_v = Status_v$ $(= NCH)$, v stays quasi-leader, because no rule can set $Ready_v$ to RO. After at most 2 rounds, a configuration of $A3$ is reached where $pif_v \neq C$. The convergence from c to a configuration of $A3$ is done as follows.

  In c, we have $(v, \perp, \perp, hs_v, pif_v) \in KT_v$, $hs_v = NCH$ and $pif_v = C$. Furthermore, in c, every node z neighbor of v has a record $(v, \perp, \perp, hs_z, pif_z)$ in KT_z, because v is quasi-leader and $c \in A1$. In c, hs_z may be different to hs_v, or pif_z may be different to C.
  In round 1: the node z either updates hs_z by performing $R1_2(z)$, or it updates pif_z by performing $IC\text{-}d1(z)$. After the z's action, we have $(v, \perp, \perp, NCH, C) \in KT_z$.
  In round 2: the rule $RB(v)$ is enabled, because $(v, \perp, \perp, NCH, C) \in KT_v$ and $(v, \perp, \perp, NCH, C) \in KT_z$. The node v stays enabled up to the time where it performs this action. By fairness, v performs this action.

After executing $RB(v)$, a configuration of $A3$ is reached where $(v, \perp, \perp, NCH, B) \in KT_v$. As $A3$ is closed under any computation step (Lemma 13). Thus, $A3$ is an attractor from $A2$. □

Lemma 16 establishes that any neighbor z of v has a record concerning the v's cluster in its knowledge table. If the record in KT_z is old then it cannot become new because the rules $R1_2$ and $IC\text{-}d1$ cannot be performed in $A4$. These rules are the only ones that may replace a old record about v in KT_z by a new one.

Lemma 15.
$P4(v) \equiv QL(v) \Rightarrow \forall z \in N_v : (v, \perp, \perp, hs_z, pif_z) \in KT_z \wedge (hs_z = HS_v \vee HS_v \neq NCH)$
$A4 = A3 \cap \{c \in Conf \mid \forall v \in V :\ P4(v)$ is satisfied $\}$ is closed under any computation step.

Proof. Let cs be a computation step $c_1 \xrightarrow{cs} c_2$, such that $c_1 \in A4$. In c_1, two general cases are possible:

- **Case 1:** v **is quasi-leader**. As $c_1 \in A3$, we have: $(v, \perp, \perp, hs_v, pif_v) \in KT_v \wedge (hs_v \neq NCH \vee pif_v \neq C)$.
  - **Deleting** $(v, \perp, \perp)$ **from** KT_z. Starting from c_1, z cannot remove the record $(v, \perp, \perp)$ from KT_z as long as $QL(v)$ is satisfied ($c_1 \in A1$, and $A1$ is closed).
  - **Updating of** hs_z. The rule $R1_2(z)$ is the only rule that changes hs_z value. This rule always sets hs_z to hs_v which has the same value as HS_v -$hs_v = HS_v$ ($c_1 \in A2$) -.
  - **Updating of** HS_v. The rules can change HS_v value are $R0_2(v)$ and $R0_3(v)$. If during cs, HS_v is set to CH, NO or O, then c_2 belongs to $A4$. Let during cs, HS_v is set to NCH; so, in c_1, $Status_v = NCH$ (i.e., v is non leader). According to Lemma 6, in c_1, v is quasi-ordinary. In addition, $R0_2(v)$ is performed during cs, only if $Status_v \neq HS_v$ in c_1. We conclude that in c_1, $HS_v \in \{O, NO\} \wedge Ready_v = RO$. In c_2, v is non quasi-leader, because $HS_v = NCH \wedge\ Ready_v = RO$. Thus, c_2 belongs to $A4$.
- **Case 2:** v **is non quasi-leader**. Starting from c_1, $RR_{CH}(v)$ is the only rule that leads v to the quasi-leader status (Lemma 7). This rule is disabled as long as there exists a v's neighbor z where $(v, \perp, \perp, NCH, F) \notin KT_z \wedge HS_v = NCH$. So, c_2 is not reached from c_1.

During cs, all reached configurations belong to $A4$. Therefore, $A4$ is closed under any computation step. □

Lemma 16. *$A4$ is an attractor from $A3$.*

Proof. Let c be a configuration of $A3$, but not of $A4$. So in c, there is at least a node v that does not satisfy $P4(v)$, i.e., $QL(v) \wedge \exists z \in N_v : (v, \perp, \perp, hs_z, pif_z) \in KT_z \wedge$
$hs_z \neq HS_v \wedge HS_v = NCH$ (because $c_1 \in A1$). In c, two cases are possible:

- **Case 1:** $HS_v \neq Status_v$. In c, v is enabled and it stays enabled up to the time where it performs $R0_2(v)$ or $R0_3(v)$. By fairness, the node v performs one of these actions. After that we have, $HS_v \neq NCH$.
- **Case 2:** $HS_v = Status_v$. While $HS_v = Status_v$ $(= NCH)$, v stays quasi-leader because no rule can set $Ready_v$ to RO. As c belongs to $A2$ and $A3$, the rule $R1_2(z)$ is enabled, and it stays enabled up to the time where it is performed. By fairness, the node z performs $R2_1(z)$. After this action, we have $hs_z = HS_v$.

In both cases, a configuration of $A4$ is reached. Thus, $A4$ is an attractor from $A3$. □

Lemma 18 establishes that any neighbor z of a quasi-leader v has an old record concerning the v's cluster in its knowledge table.

Lemma 17. $P5(v) \equiv QL(v) \Rightarrow \forall z \in N_v, (v, \perp, \perp, hs_z, pif_z) \in KT_z \wedge (hs_z \neq NCH \vee pif_z \neq C)$
$A5 = A4 \cap \{c \in Conf \mid \forall v \in V :\ P5(v)$ is satisfied $\}$ is closed under any computation step.

Proof. Let c_1 be a configuration of $A5$. Assume that a computation step cs exists: $c_1 \xrightarrow{cs} c_2$, such that c_2 does not belong to $A5$. In c_2, there is at least a node v such that $P5(v)$ is not satisfied, i.e., $QL(v) \wedge \exists z \in N_v, (v, \perp, \perp, hs_z, pif_z) \in KT_z \wedge hs_z = NCH \wedge pif_z = C$ (because $c_1 \in A1$). In c_1, two general cases are possible:

- **Case 1: v is quasi-leader**. As $c_1 \in A3$, we have: $(v, \perp, \perp, hs_v, pif_v) \in KT_v \wedge$
$(hs_v \neq NCH \vee pif_v \neq C)$. To reach c_2, z should remove the record $(v, \perp, \perp)$ from KT_z, or it should update hs_z to NCH, or pif_z to C.
  - **Deleting $(v, \perp, \perp)$ from KT_z**. Starting from c_1, z cannot remove the record $(v, \perp, \perp)$ from KT_z as long as v is quasi-leader ($c_1 \in A1$, and $A1$ is closed).
  - **Updating of hs_z to NCH**. The rule $R1_2(z)$ is the only rule that changes the hs_z value. This rule always sets hs_z to hs_v if they are different. In c_1, $hs_v = HS_v$ because $c_1 \in A2$. Furthermore, in c_1 if $hs_v = NCH$ then $hs_z = hs_v$ because c_1 belongs to $A4$. Therefore, hs_z cannot be updated to NCH.
  - **Updating of pif_z to C**. The only rule can set pif_z to C without changing hs_z is $IC\text{-}d1(z)$. If in c_1, $hs_z \in \{CH, NO\}$, then the reached configuration belongs to $A5$. Let in c_1, $hs_z = NCH$; so $pif_z \neq C$. In c_1, if $pif_v = C$, then $hs_v \neq NCH$ ($c_1 \in A3$); so, $hs_z \neq hs_v$. In this configuration, $IC\text{-}d1(z)$ is disabled because $disabled(z)$ is not satisfied (rule $R1_2(z)$ is enabled).
- **Case 2: v is non quasi-leader**. Starting from c_1, $RR_{CH}(v)$ is the only rule giving v the quasi-leader status (Lemma 7). This rule is disabled as long as there exists a v's neighbor z where $(v, \perp, \perp, NCH, F) \notin KT_z$.

The simultaneous execution of $RR_{CH}(v)$ with a z's action during cs does not falsify the predicate $P5(v)$. $RR_{CH}(v)$ is performed during cs only if in c_1, $(v, \perp, \perp, NCH, B) \in KT_v$ and
$(v, \perp, \perp, NCH, F) \in KT_z$. In this configuration, the rules $R1_2(z)$, $R1_3(z)$ and $IC\text{-}d1(z)$ are disabled.

There is a contradiction; cs does not exist. Therefore, $A5$ is closed under any computation step. □

Lemma 18. *$A5$ is an attractor from $A4$.*

Proof. Let c be a configuration of $A4$, but not of $A5$. In c, there is a node v such that $P5(v)$ is not satisfied, i.e., $QL(v) \wedge \exists z \in N_v : (v, \perp, \perp, hs_z, pif_z) \in KT_z \wedge hs_z = NCH \wedge pif_z = C$.

We prove that all computations starting from c reach a configuration of $A5$. In c, two cases are possible:

- **Case 1:** $HS_v \neq NCH$. In c, z is enabled and it stays enabled up to the time where it performs $R1_2(z)$. By fairness, the node z performs this action. After z's action, a configuration of $A5$ is reached, because $(v, \perp, \perp, hs_z, C) \in KT_z \wedge hs_z \neq NCH$.
- **Case 2:** $HS_v = NCH$. If every z's neighbor expect v, named w has the record $(v, z, \perp, hs_w, pif_w)$ in its knowledge table where $hs_w = hs_z \wedge pif_w = C$, then $IB(z)$ is enabled. Otherwise, the node w is enabled. The maximal computation that reach a configuration of $A5$ starting from c is as follows.
  In round 1: according to whether $(v, z, \perp, hs_w, pif_w) \notin KT_w$, or $hs_w \neq hs_z$ or $pif_w \neq C$, one rule among $R2_1(w)$, $R2_2(w)$ and $IC\text{-}d2(w)$ is enabled. This rule stays enabled as long as $(v, \perp, \perp, hs_z, pif_z) \in KT_z \wedge pif_z = C$. By fairness, the node w performs one of these actions. After a w's action, we have $(v, z, \perp, hs_w, C) \in KT_w \wedge hs_w = hs_z$.
  In round 2: $IB(z)$ is enabled, and it stays enabled until $pif_z \neq C$. By fairness, z performs this action. After that, a configuration of $A5$ is reached where $(v, \perp, \perp, hs_z, B) \in KT_z$.

A configuration of $A5$ is eventually reached. Thus, $A5$ is an attractor from $A4$. □

In the following, we assume that z is a neighbor of v that is quasi-ordinary. Let w be a neighbor of z. If z has an old record concerning v's cluster then w has also a record concerning the v's cluster.

Lemma 19. $P6(z) \equiv QO(z) \wedge \exists (v, \perp, \perp, hs_z, pif_z) \in KT_z \;\wedge (hs_z \neq NCH \vee pif_z \neq C)$
$\Rightarrow \forall w \in N_z/\{v\} : (v, z, \perp) \in KT_w$
$A6 = A5 \cap \{c \in Conf \mid \forall z \in V :\; P6(z)$ is satisfied $\}$ is closed under any computation step.

Proof. Let c_1 be a configuration of $A6$. Assume that a computation step cs exists: $c_1 \xrightarrow{cs} c_2$, such that $c_2 \notin A6$. In c_2, there is at least a node z such that $P6(z)$ is not satisfied; i.e., $QO(z) \wedge \exists (v, \perp, \perp, hs_z, pif_z) \in KT_z \wedge (hs_z \neq NCH \vee pif_z \neq C) \wedge \exists w \in N_z/\{v\} : (v, z, \perp) \notin KT_w$ is satisfied. In c_1, four cases are possible:

- **Case 1: In c_1, $\neg QO(z)$.** $RR_O(z)$ is the only rule that leads z to the quasi-ordinary status (Lemma 8). In c_1, if there exists $(v, \perp, \perp) \in KT_z$ and a z's neighbor w, such that $(v, z, \perp) \notin KT_w$, the predicate $Constraint1(z)$ is not satisfied, and $RR_O(z)$ is disabled.
- **Case 2: In c_1, $QO(z) \wedge (v, \perp, \perp) \notin KT_z$.** In c_1, v is non quasi-leader ($HS_v \in \{O, NCH\}$), because $c_1 \in A1$. To reach c_2, z should add the record $(v, \perp, \perp)$ to KT_z. $R1_1(z)$ is the only rule can do this action. However, the execution of this rule during cs requires that $(v, \perp, \perp, hs_v, pif_v)$ belongs to KT_v in c_1. We conclude that in c_1, $hs_v = NCH$ because $HS_v \neq O$ and $hs_v = HS_v$ ($c_1 \in A2$). If $R1_1(z)$ is performed during cs, the reached configuration belongs to $A6$, because $(v, \perp, \perp, NCH, C) \in KT_z$.
- **Case 3:** In c_1, $QO(z) \wedge (v, \perp, \perp, hs_z, pif_z) \in KT_z \;\wedge hs_z = NCH \wedge pif_z = C$.
  To reach c_2, either z updates pif_z to B or F, or it updates hs_z to NO or CH.
  - **Updating of** pif_z. Starting from c_1 (i.e. $pif_z = C$), $IB(z)$ is the only rule can update pif_z. However, this rule is disabled as long as $QO(z)$ and $\exists w \in N_z/\{v\}, (v, z, \perp) \notin KT_w$.

- **Updating of hs_z.** In c_1, v is non quasi-leader, i.e., $HS_v \in \{O, NCH\}$; because $(v, \perp, \perp, NCH, C) \in KT_z$ and $c_1 \in A5$. As $c_1 \in A2$, if $(v, \perp, \perp, hs_v, pif_v) \in KT_v$ then $hs_v = NCH$; so, $hs_v = hs_z$. $R1_2(z)$ is the only rule can update hs_z, but it can be performed during cs only if in c_1 $hs_z \neq hs_v$. Thus, in c_1, this rule is disabled.

• **Case 4: In c_1,** $QO(z) \wedge \exists(v, \perp, \perp, hs_z, pif_z) \in KT_z \wedge (hs_z \neq NCH \vee pif_z \neq C)$. To reach c_2, a z's neighbor w should delete the record $(v, z, \perp)$ from KT_w. $R2_3(w)$ is the only rule can do this action. Nevertheless, this rule is disabled in c_1, and it stays disabled as long as $QO(z)$ and $(v, \perp, \perp) \in KT_z$ are satisfied.

There is a contradiction; the computation step cs does not exist. Therefore, $A6$ is closed under any computation step. □

Lemma 20. *$A6$ is an attractor from $A5$.*

Proof. Let c be a configuration of $A5$, but not of $A6$. So, in c, there is at least a node z that does not satisfy $P6(z)$, i.e., $QO(z) \wedge \exists(v, \perp, \perp, hs_z, pif_z) \in KT_z \wedge (hs_z \neq NCH \vee pif_z \neq C) \wedge \exists w \in N_z/\{v\}, (v, z, \perp) \notin KT_w$. In c, the rule $R2_1(w)$ is enabled, and it stays enabled as long as $(v, \perp, \perp, hs_z, pif_z) \in KT_z$, $QO(z)$, and $(v, z, \perp) \notin KT_w$. By fairness, the node w performs this action, and it adds the record $(v, z, \perp)$ to KT_w. As $A6$ is closed (Lemma 19); thus, $A6$ is an attractor from $A5$. □

Once a configuration of $A6$ is reached, all nodes in v's 2-QR-neighborhood have a record concerning v's cluster. Because, all v's neighbors have an old record concerning v's cluster in their knowledge table.

Theorem 4. *In a configuration of $A6$, every node v satisfies $Sp2(v)$.*

Proof. Let v, z and w be nodes, such that $(v, z, w) \in Path$. Let c be a configuration of $A6$.
As c belongs to $A5$, $P5(v)$ is satisfied in c, and it stays satisfied after any computation step ($A5$ is an attractor, Lemma 18). Thus, we have

$$QL(v) \Rightarrow (v, \perp, \perp, hs_z, pif_z) \in KT_z \wedge (hs_z \neq NCH \vee pif_z \neq C) \qquad (1)$$

On the other hand, $P6(z)$ is satisfied in c ($c \in A6$) and after any computation step ($A6$ is an attractor, Lemma 20). Thus, we have,

$$QO(z) \wedge (v, \perp, \perp, hs_z, pif_z) \in KT_z \wedge (hs_z \neq NCH \vee pif_z \neq C) \Rightarrow (v, z, \perp) \in KT_w \qquad (2)$$

According to equations (1) and (2), we conclude that in c and along any computation, the predicate $Sp2(v)$ is satisfied: $QL(v) \wedge QO(z) \Rightarrow (v, z, \perp) \in KT_w$. □

9.2.3 Proof of SP at distance 3

Notation 2 *In follows we use the notation B/F as a value of a pif field to denote that the pif field has either the value B or F (ex. $(z, \perp, \perp, ., B/F) \in KT_z$).*

Definition 15. *Let $PQO(z, w)$ be a predicate defined as follows.*
$PQO(z, w) \equiv QO(z) \vee \big((z, \perp, \perp, NO, B/F) \in KT_z \wedge (z, \perp, \perp, NO, B/F) \in KT_w\big)$.

The predicate $PQO(z,w)$ is introduced to describe configurations where either (1) z is quasi-ordinary or (2) z has the nearly ordinary status, it has already initiated a PIF, and the z's neighbor w has participated to this PIF.

Lemma 21. *Starting from a configuration c where the predicate $PQO(z,w)$ is not satisfied, only the execution of the rule $IB(w)$ reaches a configuration where $PQO(z,w)$ is satisfied.*

Proof. Let us study the w's and z's actions that reach a configuration where $PQO(z,w)$ is satisfied.
$RR_O(z)$ is the only rule that leads z to the quasi-ordinary status (Lemma 8). $RR_O(z)$ is performed only if $(z,\perp,\perp,NO,B) \in KT_z$ and $(z,\perp,\perp,NO,F) \in KT_w$. Thus, before the execution of this rule, the predicate $PQO(z,w)$ is verified.
The rule reaching a configuration where $(z,\perp,\perp,NO,B) \in KT_z$ is $RB(z)$. The execution of this rule requires that $(z,\perp,\perp,NO,C) \in KT_z$ and $(z,\perp,\perp,NO,C) \in KT_w$. Thus, after execution of $RB(z)$, the predicate $PQO(z,w)$ stays non satisfied.
On the other hand, the rule reaching a configuration where $(z,\perp,\perp,NO,F) \in KT_z$ is $RR_O(z)$. As mentionned above, before the execution of this rule, the predicate $PQO(z,w)$ is satisfied.
The only rule reaching a configuration where $(z,\perp,\perp,NO,F) \in KT_w$ is $IF\text{-}d1(w)$. $IF\text{-}d1(w)$ is performed only if $(z,\perp,\perp,NO,B/F) \in KT_z$ and $(z,\perp,\perp,NO,B) \in KT_w$. Thus, before the execution of this rule, the predicate $PQO(z,w)$ is verified.
As consequence, only the execution of $IB(w)$ may reach a configuration satisfying $PQO(z,w)$. □

Observation 3

- *In any configuration c of $A6$, if $PQO(z,w)$ is satisfied, then $HS_z \neq CH$, because either $QO(z)$ or $HS_z = NO$ (in c, $(z,\perp,\perp,NO,B/F) \in KT_z$ and $c \in A2$).*
- *In any configuration c of $A6$, if $PQO(z,w)$ is not satisfied but the rule $IB(w)$ is enabled, then $HS_z = NO$ because $(z,\perp,\perp,NO,B/F) \in KT_z$ and $c \in A2$.*

In the following, we assume that w is a neighbor of z and $PQO(z,w)$ is verified. Lemma 22 establishes that if z has a record concerning v's cluster with the pif field value different of C then w cannot perform a rule $R2_i$ where $i \in [1,3]$ on the record $(v,z,\perp)$.

Lemma 22. $P7(z) \equiv \forall(v,z,w) \in Path, PQO(z,w) \wedge (v,\perp,\perp,hs_z,B/F) \in KT_z$
$\Rightarrow (v,z,\perp,hs_w,pif_w) \in KT_w \wedge hs_w = hs_z$
$A7 = A6 \cap \{c \in Conf \mid \forall z \in V:\ P7(z)$ is satisfied $\}$ is closed under any computation step.

Proof. Let cs be a computation step $c_1 \xrightarrow{cs} c_2$, such that $c_1 \in A7$. Assume that c_2 does not belong to $A7$. In c_1, three cases are possible:

- **Case 1: In c_1, $(v,\perp,\perp,hs_z,B/F) \notin KT_z$.** Starting from c_1, only the rules $IB(z)$ and $IF\text{-}d1(z)$ reach a configuration where $(v,\perp,\perp,hs_z,B/F) \in KT_z$.
In c_1, if the predicate $PQO(z,w)$ is not satisfied, and the rule $IB(w)$ is disabled, then the reached configuration belongs to $A7$ (because $PQO(z,w)$ stays not satisfied). Otherwise (i.e., $PQO(z,w)$ is satisfied or $IB(w)$ is enabled), we have $HS_z \neq CH$ (see Observation 3). $IB(z)$ and $IF\text{-}d1(z)$ are disabled as long as $(v,z,\perp,hs_w,pif_w) \notin KT_w \vee hs_w \neq hs_z$.
- **Case 2: In c_1, $\neg Pzw \wedge (v,\perp,\perp,hs_z,B/F) \in KT_z$.** According to lemma 21, only the rule $IB(w)$ reaches a configuration where the predicate $PQO(z,w)$ is satisfied. According to Observation 3, in c_1 if this rule is enabled, then $HS_z = NO$. Starting from c_1, as long as $(v,\perp,\perp,hs_z,B/F) \in KT_z$, and $(v,z,\perp,hs_w,pif_w) \notin KT_w$ or $hs_w \neq hs_z$, the predicate $Disabled(w)$ is not satisfied (rule $R2_1(w)$ or $R2_2(w)$ is enabled). In this configuration, the rule $IB(w)$ is disabled.

• **Case 3:** $PQO(z,w) \wedge (v,\bot,\bot,hs_z,B/F) \in KT_z$.

To reach c_2, either the node w deletes $(v,z,\bot,hs_w,pif_w)$ from KT_w, or it updates hs_w.

- **Deleting of** $(v,z,\bot)$ **from** KT_w. In c_1, we have $HS_z \neq CH$ (see Observation 3). As long as $HS_z \neq CH$ and $(v,\bot,\bot) \in KT_z$, w cannot delete the record $(v,z,\bot)$ from KT_w, because $R2_3(w)$ is disabled.
- **Updating of** hs_w. hs_w is always sets to hs_z by the rule $R2_2(w)$.

There is a contradiction; cs does not exist. Therefore, $A7$ is closed under any computation step. □

Lemma 23. *$A7$ is an attractor from $A6$.*

Proof. Let c be a configuration of $A6$, but not of $A7$. In c, there is at least a node z such that $P7(z)$ is not satisfied, i.e., $PQO(z,w) \wedge (v,\bot,\bot,hs_z,B/F) \in KT_z \wedge \exists w \in N_z/\{v\}, (v,z,\bot,hs_w,pif_w) \notin KT_w \vee hs_w \neq hs_z$.

In c, we have $HS_z \neq CH$, because $PQO(z,w)$ is satisfied (see Observation 3).

In c, according to whether $(v,z,\bot,hs_w,pif_w) \notin KT_w$ or $hs_w \neq hs_z$, one of the rules $R2_1(w)$ and $R2_2(w)$ is enabled. This rule stays enabled as long as $HS_z \neq CH$ and $(v,\bot,\bot,hs_z,pif_z) \in KT_z$. By fairness, the node w performs one of these rules, and get $(v,z,\bot,hs_w,pif_w) \in KT_w \wedge hs_w = hs_z$. A configuration of $A7$ is reached. As $A7$ is closed (Lemma 22); thus, $A7$ is an attractor from $A6$. □

Lemma 25 establishes that any neighbor z of a quasi-leader v has a very old record concerning the v's cluster in its knowledge table.

Lemma 24. $P8(v) \equiv QL(v) \Rightarrow \forall z \in N_v, (v,\bot,\bot,hs_z,pif_z) \in KT_z \wedge (hs_z \neq NCH \vee pif_z = F)$
$A8 = A7 \cap \{c \in Conf \mid \forall v \in V:\ P8(v)$ is satisfied $\}$ is closed under any computation step.

Proof. Let c_1 be a configuration of $A8$. Assume that a computation step cs exists: $c_1 \xrightarrow{cs} c_2$, such that c_2 does not belong to $A8$. In c_1, two general cases are possible:

• **Case 1: v is quasi-leader**. As $c_1 \in A3$, we have: $(v,\bot,\bot,hs_v,pif_v) \in KT_v \wedge (hs_v \neq NCH \vee pif_v \neq C)$. To reach c_2, z should remove the record $(v,\bot,\bot)$ from KT_z, or it should update hs_z to NCH, or pif_z to B or C.

- **Deleting** $(v,\bot,\bot)$ **from** KT_z. Starting from c_1, z cannot remove the record $(v,\bot,\bot)$ from KT_z as long as $QL(v)$ is satisfied ($c_1 \in A1$, and $A1$ is closed).
- **Updating of** hs_z **to** NCH. $R1_2(z)$ is the only rule that changes the hs_z value. This rule always sets hs_z to hs_v. In c_1, $hs_v = HS_v$ because $c_1 \in A2$. Furthermore, in c_1 if $hs_v = NCH$ then $hs_z = hs_v$ because $c_1 \in A4$. In this configuration, $R1_2(z)$ is disabled.
- **Updating of** pif_z **to** B **or** C. The only rules can set pif_z to B or C without changing hs_z are $IC\text{-}d1(z)$ and $IB(z)$. If in c_1, $hs_z \in \{CH, NO\}$, then the reached configuration belongs to $A8$. Let in c_1, $hs_z = NCH$; so $pif_z = F$. As in c_1 $pif_z = F$, then $IB(z)$ is disabled. $IC\text{-}d1(z)$ is enabled in c_1 only if $pif_v = C$, i.e., $hs_v \neq NCH$ ($c_1 \in A3$); so, $hs_z \neq hs_v$. In this configuration, $disabled(z)$ is not satisfied (rule $R1_2(z)$ is enabled). Thus, $IC\text{-}d1(z)$ is also disabled.

• **Case 2: v is non quasi-leader**. Starting from c_1, $RR_{CH}(v)$ is the only rule giving v the quasi-leader status (Lemma 7). This rule is disabled as long as there exists a v's neighbor z where $(v,\bot,\bot,NCH,F) \notin KT_z$.

Let us study the simultaneous execution of $RR_{CH}(v)$ with a z's action during cs. $RR_{CH}(v)$ is performed during cs only if in c_1, $(v,\bot,\bot,NCH,B) \in KT_v$ and $(v,\bot,\bot,NCH,F) \in KT_z$. At that time, $P8(v)$ is verified, and no z's action can falsify the predicate $P8(v)$, because the rules $R1_1(z)$, $R1_2(z)$, $R1_3(z)$, $IC\text{-}d1(z)$, $IB(z)$ and $IF\text{-}d1(z)$ are disabled.

There is a contradiction; cs does not exist. Therefore, $A8$ is closed under any computation step. □

Lemma 25. *$A8$ is an attractor from $A7$.*

Proof. Let c be a configuration of $A7$, but not of $A8$. In c, there is at least a node v such that $P8(v)$ is not satisfied, i.e., $QL(v) \wedge \exists z \in N_v : (v, \perp, \perp, hs_z, pif_z) \in KT_z \wedge hs_z = NCH \wedge pif_z \neq F$.

We prove that all computations starting from c reach a configuration of $A8$. In c, two cases are possible:

• **Case 1:** $HS_v \neq NCH$. In c, z is enabled and it stays enabled up to the time where it performs $R1_2(z)$. By fairness, the node z performs this action. After z's action, a configuration of $A8$ is reached, because $(v, \perp, \perp, hs_z, C) \in KT_z \wedge hs_z \neq NCH$.

• **Case 2:** $HS_v = NCH$. As $c_1 \in A3$, we have $(v, \perp, \perp, hs_v, pif_v) \in KT_v \wedge hs_v = NCH \wedge pif_v \neq C$. As $c_1 \in A5$, we have $pif_z \neq C$; so, $pif_z = B$. Since $c \in A7$, every node w neighbor of z $(w \neq v \wedge HS_z \neq CH)$ has the record $(v, z, \perp, hs_w, pif_w)$ in KT_w, and $hs_w = NCH$.

The maximal computation that reaches a configuration of $A8$ is as follows.
In round 1: every non ordinary node u neighbor of w $(u \neq z \wedge u \neq v \wedge HS_w \neq CH \wedge HS_u \neq O)$ that does not have the record (v, w, z, hs_u, pif_u) in KT_u, or $hs_u \neq NCH$ is enabled. One rule among $R3_1(u)$, $R3_2(u)$ is enabled. This rule stays enabled as long as $(v, z, \perp, hs_w, pif_w) \in KT_w \wedge hs_w = NCH$. By fairness, the node u performs one of these actions. After the u's action, we have $(v, w, z, hs_u, C) \in KT_u \wedge hs_u = NCH$.
In round 2: the rule $IF\text{-}d2(w)$ is enabled, and it stays enabled as long as $pif_w = C$, $pif_z = B$ and $(v, w, z, NCH, C) \in KT_u$. By fairness, w performs this rule. After that, we have $(v, z, \perp, NCH, F) \in KT_w$.
In round 3: the rule $IF\text{-}d1(z)$ becomes enabled because $pif_v \neq C$, $pif_z = B$ and $(v, z, \perp, NCH, F) \in KT_w$. By fairness, the node z performs this action. Once z performs this action, we have $(v, \perp, \perp, NCH, F) \in KT_z$.

A configuration of $A8$ is eventually reached. Thus, $A8$ is an attractor from $A7$. □

In $A9$, if z has a very old record concerning v's cluster then w has a old record concerning the v's cluster.

Lemma 26.
$$P9(z) \equiv \forall (v, z, w) \in Path, PQO(z, w) \wedge (v, \perp, \perp, hs_z, pif_z) \in KT_z \wedge (hs_z \neq NCH \vee pif_z = F)$$
$$\Rightarrow (v, z, \perp, hs_w, pif_w) \in KT_w \wedge (hs_w \neq NCH \vee pif_w \neq C)$$
$A9 = A8 \cap \{c \in Conf \mid \forall z \in V :\ P9(z)$ is satisfied $\}$ is closed under any computation step.

Proof. Let cs be a computation step $c_1 \xrightarrow{cs} c_2$, such that $c_1 \in A9$. Assume that c_2 does not belong to $A9$. In c_1, four cases are possible:

• **Case 1: In c_1, $(v, \perp, \perp) \notin KT_z$.** In c_1, v is non quasi-leader $(HS_v \in \{O, NCH\})$, because $c_1 \in A1$. To reach c_2, z should add the record $(v, \perp, \perp)$ to KT_z. $R1_1(z)$ is the only rule can do this action. However, the execution of this rule during cs requires that $(v, \perp, \perp, hs_v, pif_v)$ belongs to KT_v in c_1. We conclude that in c_1, $hs_v = NCH$ because $HS_v \neq O$ and $hs_v = HS_v$ $(c_1 \in A2)$. If $R1_1(z)$ is performed during cs, the reached configuration belongs to $A9$, because $(v, \perp, \perp, NCH, C) \in KT_z$.

• **Case 2: In c_1, $(v, \perp, \perp, hs_z, pif_z) \in KT_z \wedge hs_z = NCH \wedge pif_z \neq F$.**
To reach c_2, either z updates pif_z to F, or it updates hs_z to NO or CH.

- **Updating of pif_z to F.** Starting from c_1, $IF\text{-}d1(z)$ is the only rule can update pif_z to F.
  In c_1, if $PQO(z,w)$ is not satisfied and the rule $IB(w)$ is disabled, then after z's action the configuration still belongs to $A9$, because $PQO(z,w)$ stays not satisfied.
  Otherwise (i.e., $PQO(z,w)$ is satisfied or $IB(w)$ is enabled), in c_1 we have $HS_z \neq CH$ (Observation 3). $IF\text{-}d1(z)$ is disabled as long as $HS_z \neq CH$ and $(v,z,\perp,hs_w,pif_w) \notin KT_w \vee$
  $hs_w \neq hs_z \vee pif_w = C$.
- **Updating of hs_z to NO or CH.** In c_1, v is non quasi-leader, i.e., $HS_v \in \{O, NCH\}$; because $hs_z = NCH, pif_z \neq F$ and $c_1 \in A7$. In c_1, if $(v,\perp,\perp,hs_v,pif_v) \in KT_v$ then $hs_v = NCH$ ($c_1 \in A2$). $R1_2(z)$ is the only rule can update hs_z, and it always sets hs_z to hs_v. In c_1, $hs_z = hs_v$; thus $R1_2(z)$ is disabled.

• **Case 3:** $\neg PQO(z,w) \wedge (v,\perp,\perp,hs_z,pif_z) \in KT_z \wedge (hs_z \neq NCH \vee pif_z = F)$. According to lemma 21, only the rule $IB(w)$ reaches a configuration where the predicate $PQO(z,w)$ is satisfied. According to Observation 3, in c_1, if this rule is enabled, then $HS_z = NO$. In c_1, as long as $(v,\perp,\perp,hs_z,pif_z) \in KT_z \wedge (hs_z \neq NCH \vee pif = F)$, but $(v,z,\perp,hs_w,pif_w) \notin KT_w$ or
$hs_w = NCH \wedge pif_w = C$, then the predicate $Constraint3(w)$ is not satisfied; and $IB(w)$ is disabled.

• **Case 4:** $PQO(z,w) \wedge (v,\perp,\perp,hs_z,pif_z) \in KT_z \wedge (hs_z \neq NCH \vee pif_z = F)$. To reach c_2, either w deletes the record $(v,z,\perp,hs_w,pif_w)$ from KT_w, or it updates hs_w to NCH, or it updates pif_w to C.

- **Deleting of $(v,z,\perp)$ from KT_w.** According to Observation 3, in c_1, $HS_z \neq CH$. Starting from c_1, as long as $HS_z \neq CH$ and $(v,\perp,\perp) \in KT_z$, the node w cannot delete the record $(v,z,\perp)$ from KT_w, because $R2_3(w)$ is disabled.
- **Updating of hs_w to NCH.** The rule $R2_2(w)$ is the only rule that changes the hs_w value. This rule always sets hs_w to hs_z. However, in c_1 if $hs_z = NCH$, then $pif_z = F$; so $hs_z = hs_w$ (because $c_1 \in A7$). In this configuration, $R2_2(w)$ is disabled.
- **Updating of pif_w to C.** $IC\text{-}d2(w)$ is the only rule can update pif_w to C without updating hs_w. Thus, if in c_1, $hs_w \in \{CH, NO\}$ then the reached configuration belongs to $A9$. Let in c_1, $hs_w = NCH \wedge pif_w \neq C$. During cs, $IC\text{-}d2(w)$ is performed only if in c_1, $pif_z = C$ (i.e., $hs_z \neq NCH$). In this configuration, $disabled(w)$ is not satisfied, because $R2_2(w)$ is enabled ($hs_w \neq hs_z$). Thus, $IC\text{-}d2(w)$ is disabled in c_1.

There is a contradiction; cs does not exist. Therefore, $A9$ is closed under any computation step. □

Lemma 27. *$A9$ is an attractor from $A8$.*

Proof. Let c be a configuration of $A8$, but not of $A9$. In c, there is at least a node z that does not satisfy $P9(z)$, i.e., $PQO(z,w) \wedge (v,\perp,\perp,hs_z,pif_z) \in KT_z \wedge (hs_z \neq NCH \vee pif_z = F) \wedge$
$\exists w \in N_z/\{v\}, (v,z,\perp,hs_w,pif_w) \in KT_w \wedge hs_w = NCH \wedge pif_w = C$ (because $c \in A7$).
In c, we have $HS_z \neq CH$, because $PQO(z,w)$ is satisfied (see Observation 3). We prove that all computations starting from c reach a configuration of $A8$. In c, two cases are possible:

• **Case 1:** $hs_z \neq NCH$. In c, w is enabled and it stays enabled up to the time where it performs $R2_2(w)$. By fairness, the node w performs this action. After the w's action, a configuration of $A9$ is reached where $(v,z,\perp,hs_w,C) \in KT_z \wedge hs_w \neq NCH$.

• **Case 2:** $hs_z = NCH$. Since $pif_z = F$ and $c \in A7$, every node w neighbor of z ($w \neq v$) has the record $(v,z,\perp,hs_w,pif_w)$ in KT_w, and $hs_w = hs_z$. Moreover, in c_1, $HS_z \neq CH$ (Observation 3).

The maximal computation that reaches a configuration of $A9$ is as follows.

In round 1: every non-ordinary node u neighbor of w $(u \neq z \wedge u \neq v \wedge HS_w \neq CH \wedge HS_u \neq O)$ that does not have the record (v, w, z, hs_u, pif_u) in KT_u, or $hs_u \neq NCH$ is enabled. One rule among $R3_1(u)$, $R3_2(u)$ is enabled. This rule stays enabled as long as $(v, z, \perp, hs_w, pif_w) \in KT_w \wedge hs_w = NCH$. By fairness, the node u performs one of these actions. After u's action, we have $(v, w, z, hs_u, C) \in KT_u \wedge hs_u = hs_w$.
In round 2: the rule $IF\text{-}d2(w)$ is enabled, and it stays enabled as long as $pif_z = F$. By fairness, the node w performs this action. After this action, we have $(v, z, \perp, NCH, F) \in KT_w$.

A configuration of $A9$ is reached. As $A9$ is closed (Lemma 26); thus, $A9$ is an attractor from $A8$. □

In the following, we assume that u is a neighbor of w that is a quasi-ordinary, and u is a quasi-leader. If w has an old record concerning v's cluster then u has also a record concerning the v's cluster.

Lemma 28. $P10(w) \equiv \forall(v, z, w, u) \in Path, PQO(z, w) \wedge QO(w) \wedge (v, z, \perp, hs_w, pif_w) \in KT_w \wedge$
$(hs_w \neq NCH \vee pif_w \neq C) \Rightarrow \neg QL(u) \vee (v, w, z) \in KT_u$
$SC2 = A9 \cap \{c \in Conf \mid \forall w \in V :\ P10(w)$ is satisfied $\}$ is closed under any computation step.

Proof. Let cs be a computation step $c_1 \xrightarrow{cs} c_2$, such that $c_1 \in SC2$. Starting from c_1, only the execution of w's rules can reach a configuration where $PQO(z, w) \wedge QO(w) \wedge (v, z, \perp, hs_w, pif_w) \in KT_w \wedge (hs_w \neq NCH \vee pif_w \neq C)$ is satisfied (if it is not satisfied in c_1). As the same, only the execution of u's rules can reach a configuration where $\neg QL(u) \vee (v, w, z) \in KT_u$ is not satisfied. As in an atomic step, w and u execute only one rule. Thus, only the execution of one rule among $IB(w)$, $RR_O(w)$, $R2_1(w)$, $R2_2(w)$, $IF\text{-}d2(w)$, $RR_{CH}(u)$, $R3_3(u)$ and $R3_4(u)$ during cs, may falsify the predicate $P10(w)$.

According to lemma 21, only the execution of $IB(w)$ may reach a configuration where the predicate $PQO(z, w)$ is satisfied. In c_1, as long as $QO(w) \wedge (v, z, \perp, hs_w, pif_w) \in KT_w \wedge (hs_w \neq NCH \vee pif_w \neq C)$, and there is quasi-leader u where $(v, w, z) \notin KT_u$, the predicate $Constraint2(w)$ is not satisfied; and so, the rule $IB(w)$ is disabled.

$RR_O(w)$ is the only rule that leads w to the quasi-ordinary status (Lemma 8). In c_1, as long as $(v, z, \perp, hs_w, pif_w) \in KT_w \wedge (hs_w \neq NCH \wedge pif_w \neq C)$ and there exists a quasi-leader u where $(v, w, z) \notin KT_u$, the predicate $Constraint2(w)$ is not satisfied; and so, $RR_O(w)$ is disabled.

$R2_1(w)$ is the only rule can add the record $(v, z, \perp)$ to KT_w. As long as in c_1 $PQO(z, w)$ is satisfied, then $HS_z \neq CH$ (Observation 3). Furthermore, as $c_1 \in A9$, in c_1 we have $(v, \perp, \perp, hs_z, pif_z) \notin KT_z \vee (hs_z = NCH \wedge pif_z \neq F)$. The execution of $R2_1(w)$ during cs requires that $(v, \perp, \perp, hs_z, pif_z)$ belongs to KT_z in c_1. We conclude that in c_1, $hs_z = NCH$. If $R2_1(w)$ is performed during cs, the reached configuration belongs to $SC2$, because $(v, z, \perp, NCH, C) \in KT_w$.

Let in c_1, $PQO(z, w) \wedge QO(w) \wedge (v, z, \perp, hs_w, pif_w) \in KT_w$ but $hs_w = NCH \wedge pif_w = C$. Let us study the updating of hs_w and pif_w values.

- **Updating of** pif_w. Starting from c_1 (i.e. $pif_w = C$), $IF\text{-}d2(w)$ is the only rule can update pif_w. The rule $IF\text{-}d2(w)$ is disabled as long as $QO(w)$ and there exists a quasi-leader u where $(v, w, z) \notin KT_u$.
- **Updating of** hs_w. $R2_2(w)$ is the only rule updating hs_w. According to Observation 3, in c_1 we have $HS_z \neq CH$ because Pzw is satisfied. Furthermore, as $c_1 \in A9$ and $hs_w = NCH \wedge pif_w = C$, we conclude that in c_1 $(v, \perp, \perp, hs_z, pif_z) \notin KT_z \vee (hs_z = NCH \wedge pif_z \neq F)$. The execution of $R2_2(w)$

during cs requires that in c_1 $(v, \perp, \perp, hs_z, pif_z) \in KT_z$. In this configuration, $hs_z = hs_w = NCH$. Thus, $R2_2(w)$ is disabled.

Let in c_1, $PQO(z, w) \wedge QO(w) \wedge (v, z, \perp, hs_w, pif_w) \in KT_w \wedge (hs_w \neq NCH \vee pif_w \neq C)$. Let us study the u's actions that may falsify the predicate $P10(w)$.

• **Deleting of** (v, w, z) **from** KT_u. Starting from c_1, a quasi-leader u cannot remove the record (v, w, z) from KT_u as long as $(v, z, \perp) \in KT_w$ and $QO(w)$, because the rules $R3_3(w)$ and $R3_4(w)$ are disabled.

• **Becoming quasi-leader**. The rule $RR_{CH}(u)$ is the rule that leads u to the quasi-leader status. However, as long as $(v, z, \perp, hs_w, pif_w) \in KT_w \wedge (hs_w \neq NCH \vee pif_w \neq C)$, $QO(w)$ and $(v, w, z) \notin KT_u$, the predicate *disabled*(u) is not satisfied (rule $R3_1(u)$ is enabled). In this configuration, $RR_{CH}(u)$ is disabled.

Starting from c_1, all reached configurations belong to $SC2$. Therefore, $SC2$ is closed under any computation step. □

Lemma 29. *$SC2$ is an attractor from $A9$.*

Proof. Let c be a configuration of $A9$, but not of $SC2$. In c, there is at least a node w that does not satisfy $P10(w)$, i.e., $\exists(v, z, w, u) \in Path, PQO(z, w) \wedge QO(w) \wedge (v, z, \perp, hs_w, pif_w) \in KT_w \wedge (hs_w \neq NCH \vee pif_w \neq C) \wedge QL(u) \wedge (v, w, z) \notin KT_u$.
In c, we have $HS_w \neq CH$, and $HS_u \neq O$ (by definitions of quasi-ordinary and quasi-leader). In c, the rule $R3_1(u)$ is enabled, and it stays enabled as long as $QO(w)$, $(v, z, \perp, hs_w, pif_w) \in KT_w$, $QL(u)$ and $(v, w, z) \notin KT_u$. By fairness, the node u performs this action, and it adds the record (v, w, z) to KT_u. As $SC2$ is closed (Lemma 28); thus, $SC2$ is an attractor from $A9$. □

Once a configuration of $SC2$ is reached, all nodes in v's 3-QR-neighborhood that are quasi-leaders have a record concerning v's cluster. Because, all nodes in v's 2-QR-neighborhood have an old record concerning v's cluster in their knowledge table.

Theorem 5. *In a configuration of $SC2$, every node v satisfies $Sp3(v)$.*

Proof. Let v, z, w and u be nodes, such that $(v, z, w, u) \in Path$. Let c be a configuration of $SC2$. By definition of the predicate $PQO(z, w)$, if z is quasi-ordinary in c, then $PQO(z, w)$ is satisfied. In c, $P8(v)$ and $P9(v)$ are satisfied, and they stay satisfied after any computation step. Thus, we have

$$QO(z) \wedge QL(v) \Rightarrow (v, z, \perp, hs_w, pif_w) \in KT_w \wedge (hs_w \neq NCH \vee pif_w \neq C) \tag{3}$$

$P10(z)$ is satisfied in c ($c \in SC2$) and after any computation step ($SC2$ is an attractor, Lemma 29). Thus, we have,

$$\begin{aligned} QO(z) \wedge QO(w) \wedge (v, z, \perp, hs_w, pif_w) \in KT_w \wedge (hs_w \neq NCH \vee pif_w \neq C) \\ \Rightarrow \neg QL(u) \vee (v, w, z) \in KT_u \end{aligned} \tag{4}$$

According to equations (3) and (4), we conclude that in c and along any computation, the predicate $Sp3(v)$ is satisfied: $QL(v) \wedge QO(z) \wedge QO(w) \Rightarrow \neg QL(u) \vee (v, w, z) \in KT_u$. □

The following theorem is a consequence of the lemmas 2, and 3 and the theorems 5 and 4.

Theorem 6. *In any configuration of $SC2$, SP is satisfied.*

9.3 Completeness property

Theorem 7. *Any computation of clustering and CNK protocol has a suffix where the completeness property is verified by all reached configurations.*

Proof. Let e be a computation of clustering and CNK protocols. We name e_1 the suffix of e starting from a configuration of $SC2$ (all reached configurations of e_1 belong to $SC2$). The suffix e_1 exists, because $SC2$ is an attractor of the clustering and CNK protocols (lemma 29).
In any configuration of $SC2$, SP is satisfied (theorem 6). We conclude that along e_1, the completeness property is verified (theorem 1) because all reached configurations belong to $SC1$ and verify the Strong-completeness predicate. □

10 Concluding remarks

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. After the first round, each quasi-leader v has an accurate record about its own cluster in KT_v (by performing the rules $R0_j$ where $j \in [1, 2]$). During the $i + 1^{th}$ round ($i \in [1, 3]$), every quasi-leader v knows the paths to quasi-leaders within its iQR-neighborhood (by performing the rules Ri_j where $i \in [1, 3], j \in [1, 2]$). In a legitimate configuration, in addition to the completeness property, the correctness property is satisfied. So, each record of KT_v on every node v, is correct: the destination is a leader, the gateways are not leader, the paths are valid, the list of members is exact.
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. Only Propagation of Information with Feedback (PIF) within the v's 3-neighborhood guarantee that each cluster-head always knows the paths to its neighbor clusters.

After the last modification of hierarchical status, 4 rounds are enough to reach a terminal configuration (that is also a legitimate configuration).

Communication. CNK protocol is designed for the state model. Nevertheless, it can be easily transformed for the message-passing model. For instance, by the following basic mechanism; each node v broadcasts periodically in its neighbors its state (i.e. identity of v, HS_v, and KT_v). Based on this message, v's neighbors update their states if necessary.
CNK protocol provides a highly available useful service. Furthermore, storing all paths to the neighbor clusters limits the losses of connectivity in case of failure (not admissible fault). But, it is costly in term of memory space on each node, and in term of message size. Nevertheless, storing one or several paths to a given cluster-head does not impact on the number of exchanged messages: each head has to explore all its 3R-neighborhood to find at least a path to all neighbor clusters.

References

1. Abbasi, A.A., Younis, M.: A survey on clustering algorithms for wireless sensor networks. Computer Communications **30** (2007) 2826–2841
2. Johnen, C., Nguyen, L.H.: Robust self-stabilizing weight-based clustering algorithm. Theoretical Computer Science **410**(6-7) (2009) 581–594
3. Johnen, C., Mekhaldi, F.: Robust self-stabilizing construction of bounded size weight-based clusters. In: Euro-Par'10, Springer LNCS 6271. (2010) 535–546

4. Dasgupta, A., Ghosh, S., Xiao, X.: Fault-containment in weakly-stabilizing systems. In: SSS'09, springer LNCS 5873. (2009) 209–223
5. Joffroy Beauquier, S.D., Haddad, S.: A 1-strong self-stabilizing transformer. In: SSS'06, springer LNCS 4280. (2006) 95–109
6. Dolev, S., Herman, T.: Superstabilizing protocols for dynamic distributed systems. Chicago J. Theor. Comput. Sci. **1997** (1997)
7. Johnen, C., Tixeuil, S.: Route preserving stabilization. In: SSS'03, Springer LNCS 2704. (2003) 184–198
8. Kakugawa, H., Masuzawa, T.: A self-stabilizing minimal dominating set algorithm with safe convergence. In: APDCM'06. (2006)
9. Kamei, S., Kakugawa, H.: A self-stabilizing approximation for the minimum connected dominating set with safe convergence. In: OPODIS'08, Springer LNCS 5401. (2008) 496–511
10. Lin, C.R., Gerla, M.: Adaptive clustering for mobile wireless networks. IEEE Journal on Selected Areas in Communications **15** (1997) 1265–1275
11. Chatterjee, M., Das, S.K., Turgut, D.: WCA: A weighted clustering algorithm for mobile ad hoc networks. Journal of Cluster Computing **5**(2) (2002) 193–204
12. Bein, D., Datta, A.K., Jagganagari, C.R., Villain, V.: A self-stabilizing link-cluster algorithm in mobile ad hoc networks. In: ISPAN'05. (2005) 436–441
13. Demirbas, M., Arora, A., Mittal, V., Kulathumani, V.: A fault-local self-stabilizing clustering service for wireless ad hoc networks. IEEE Transactions on Parallel and Distributed Systems **17** (2006) 912–922
14. Drabkin, V., Friedman, R., Gradinariu, M.: Self-stabilizing wireless connected overlays. In: OPODIS'06, Springer LNCS 4305. (2006) 425–439
15. Datta, A., Devismes, S., Larmore, L.: A self-stabilizing $o(n)$-round k-clustering algorithm. In: SRDS'09. (2009)
16. Dolev, S., Tzachar, N.: Empire of colonies: Self-stabilizing and self-organizing distributed algorithm. Theoretical Computer Science **410** (2009) 514–532
17. Calinescu, G.: Computing 2-hop neighborhoods in ad hoc wireless networks. In: ADHOC-NOW'03, springer LNCS 2865. (2003) 175–186
18. Gairing, M., Goddard, W., Hedetniemi, S.T., Kristiansen, P., McRae, A.A.: Distance-two information in self-stabilizing algorithms. Parallel Processing Letters **14**(3-4) (2004) 387–398
19. Goddard, W., Hedetniemi, S.T., Jacobs, D.P., Trevisan, V.: Distance-k knowledge in self-stabilizing algorithms. Theoretical Computer Science **399** (2008) 118–127
20. Belding-Royer, E.M.: Multi-level hierarchies for scalable ad hoc routing. Wireless Networks **9**(5) (2003) 461–478
21. Segall, A.: Distributed network protocols. IEEE Transactions on Information Theory **29**(1) (1983) 23–34
22. Bui, A., Datta, A.K., Petit, F., Villain, V.: Snap-stabilization and PIF in tree networks. Distributed Computing **20** (2007) 3–19