



**HAL**  
open science

# Memory Efficient Self-Stabilizing k-Independent Dominating Set Construction

Colette Johnen

► **To cite this version:**

Colette Johnen. Memory Efficient Self-Stabilizing k-Independent Dominating Set Construction. 2013.  
hal-00843995v1

**HAL Id: hal-00843995**

**<https://hal.science/hal-00843995v1>**

Submitted on 12 Jul 2013 (v1), last revised 1 Oct 2013 (v2)

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Memory efficient Self-Stabilizing $k$ -Independent Dominating Set Construction\*

## Labri Technical Report RR-1473-13

Colette Johnen

Univ. Bordeaux, LaBRI, UMR 5800, F-33400 Talence, France

**Abstract.** We propose a memory efficient self-stabilizing protocol building  $k$ -independent dominating sets. A  $k$ -independent dominating set is a  $k$ -independent set and a  $k$ -dominating set. A set of nodes,  $I$ , is  $k$ -independent if the distance between any pair of nodes in  $I$  is at least  $k + 1$ . A set of nodes,  $D$ , is a  $k$ -dominating if every node is within distance  $k$  of a node of  $D$ .

Our algorithm, named *STD*, is silent; it converges under the unfair distributed scheduler (the weakest scheduling assumption). The built  $k$ -independent dominating sets contain at most  $\lfloor 2n/k + 2 \rfloor$  nodes,  $n$  being the network size.

The protocol *STD* is memory efficient : it requires only  $2\log((k+1)n+1)$  bits per node.

The correctness and the termination of the protocol *STD* is proven.

**keywords** distributed computing, fault tolerance, self-stabilization,  $k$ -dominating set,  $k$ -independent set,  $k$ -independent dominating set

## 1 Introduction

The clustering of networks consists of partitioning network nodes into non-overlapping groups called clusters. Each cluster has a single head, called leader, that acts as local coordinator of the cluster, and eventually a set of standard nodes. In 1-hop clusters, the standard nodes are neighbor (at distance 1) of their leader. Clustering is found very attractive in infrastructure-less networks, like ad-hoc networks, since it limits the responsibility of network management only to leaders, and it allows the use of hierarchical routing. This is why numerous self-stabilizing 1-hop clustering protocols were proposed in the literature [9,10,8,11,13,14,15,16,19,20].

Silent self-stabilizing protocols building  $k$ -hops clustering set are proposed for  $k > 1$ . In  $k$ -hop clusters, the distance between a standard node and its leader is at most  $k$  [1,2,3,7]. The sets of cluster heads built by these

---

\* This work was partially supported by the ANR project *Displexity*.

protocols are not  $k$ -independant. The protocol of [1] is designed for  $k = 2$ . Routing tables are maintained by the cluster heads to store routing information to nodes both within and outside the cluster. The goal of the protocol in [2] is to build bounded size clusters (each cluster has at most *Cluster\_Max* nodes). bits per node. The protocol of [3] is designed for weighted edges networks; it requires at least  $O(k.log(n))$  bits per node. The protocol of [7] requires at least  $2(k + 1)log(n)$

In [17,18], Larsson and Tsigas propose self-stabilizing  $(l,k)$ -clustering protocols under various assumptions. These protocols ensure, if possible, that each node has  $l$  cluster-heads at distance at most  $k$ .

**Related Works.** In [4], a silent self-stabilizing protocol extracting a minimal  $k$ -dominating set from any  $k$ -dominating set is proposed. A minimal  $k$ -dominating set has no proper subset which also a  $k$ -dominating set. The protocol requires at least  $O(k.log(n))$  bits per node.

The paper [6] presents a silent self-stabilizing protocol building a small  $k$ -dominating set : the obtained dominating set contains at most  $\lceil n/k + 1 \rceil$ . The protocol of [6] requires  $O(log(n) + k.log(n/k))$  bits per node. The protocol of [5] builds competitive  $k$ -dominating sets : the obtained dominating set contains at most  $1 + \lfloor n - 1/k + 1 \rfloor$  nodes. The protocol of [5] requires  $O(log(n))$  bits per node. These both protocols use the hierachical collateral composition of several silent self-stabilizing protocols whose a leader election protocol and a spanning tree construction rooted to the elected leader. So they requires more memory space than our protocol.

In [12], A fast silent self-stabilizing protocol building a  $k$ -independant dominating set is proposed the protocol requires  $(k + 1)log(n + 1)$  bits per node.

**Contribution.** In this paper, we consider the problem of computing a  $k$ -independant dominating set in a self-stabilizing manner in case where  $k > 1$ . A nodes set is  $k$ -independant dominating set (also called maximal  $k$ -independant set) if and only if this set is a  $k$ -independant set and a  $k$ -dominating set. A set of nodes,  $I$  is  $k$ -independent if the distance between any pair of  $I$ 's nodes is at least  $k + 1$ . A set of nodes  $D$  is  $k$ -dominating if every node is within distance  $k$  of a node of  $D$ .

The presented protocol, named *SID*, is simple : no used of the the hierachical collateral composition, no need of leader election process, neither the building of spanning tree. The protocol *SID* converges under the unfair distributed scheduler (the weakest scheduling assumption). The

algorithm *STD* is silent. The protocol *STD* is memory efficient : it requires only  $2\log((k + 1)n + 1)$  bits per node.

In section 2, we establish that any  $k$ -independent sets contain at most  $\lfloor 2n/k + 2 \rfloor$  nodes,  $n$  being the network size. So the protocol of [12] and the protocol *STD* have the same upper bound on the size of built  $k$  independent dominating sets :  $\lfloor 2n/k + 2 \rfloor$  nodes.

**Paper outline.** The rest of the paper is organised as follows. In section 2, communication and computation models are defined. The protocol *STD* is presented in section 3. In the section 4, the correctness of the *STD* protocol is proven. The termination of the *STD* protocol is established in the section 5.

## 2 Model and Concepts

A distributed system  $S$  is an undirected graph  $G = (V, E)$  where vertex set  $V$  is the set of nodes and edge set  $E$  is the set of communication links. A link  $(u, v) \in E$  if and only if  $u$  and  $v$  can directly communicate (links are bidirectional); so,  $u$  and  $v$  are neighbors. We note by  $N_v$  the set of  $v$ 's neighbors:  $N_v = \{u \in V \mid (u, v) \in E\}$ . The distance between the nodes  $u$  and  $v$  is denoted  $dist(u, v)$ .  $k$ -neighborhood( $v$ ) =  $\{u \in V \mid dist(u, v) \in [1, k]\}$ .

Furthermore, at every node  $v$  in the network is assigned an identifier, denoted  $id_v$ . Two distinct nodes have different identifier. It is possible to order the identifier values. The symbol  $\perp$  denotes a value smaller than any identifier value in the network.

Each node  $v$  maintains a set of shared variables such that  $v$  can read its own variables and those of its neighbors, but it can modify only its variables. The *state* of a node is defined by the values of its local variables. The union of states of all nodes determines the *configuration* of the system. Let  $\mathit{var}$  be a shared variable,  $\mathit{var}(v)_c$  is the value of  $\mathit{var}$  for the node  $u$  in the configuration  $c$ . The *program* of each node is a set of *rules*. Each rule has the form:  $Rule_i : \langle Guard_i \rangle \longrightarrow \langle Action_i \rangle$ . The *guard* of a  $v$ 's rule is a boolean expression involving the state of the node  $v$ , and those of its neighbors. The *action* of a  $v$ 's rule updates  $v$ 's state. A rule can be executed only if it is *enabled*, i.e., its guard evaluates to true. A node is said to be enabled if at least one of its rules is enabled. In a *terminal configuration*, no node is enabled.

During a *computation step*  $c_i \rightarrow c_{i+1}$ , one or several enabled nodes perform an enabled action and the system reaches the configuration  $c_{i+1}$  from

$c_i$ . A *computation*  $e$  is a sequence of configurations  $e = c_0, c_1, \dots, c_i, \dots$ , where  $c_{i+1}$  is reached from  $c_i$  by one computation step:  $\forall i \geq 0, c_i \rightarrow c_{i+1}$ . We say that a computation  $e$  is *maximal* if it is infinite, or if it reaches a terminal configuration. We note by  $\mathcal{C}$  the set of all possible configurations, and by  $\mathcal{E}$  the set of all maximal computations. The set of maximal computations starting from a particular configuration  $c \in \mathcal{C}$  is denoted  $\mathcal{E}_c$ .  $\mathcal{E}_A$  denotes the set of all maximal computations where the initial configuration belongs to the set of configurations  $A \subset \mathcal{C}$ .

**Definition 1 (Attractor).** Let  $B_1$  and  $B_2$  be subsets of  $\mathcal{C}$ .  $B_2$  is an attractor from  $B_1$ , if and only if the following conditions hold:

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

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

- $\mathcal{L}$  is an attractor from  $\mathcal{C}$ ;
- Configurations of  $\mathcal{L}$  match the specification problem.

A self-stabilizing protocol is *silent* if all maximal computations are finite.

**Stabilization time.** We use the *round* notion to measure the time complexity. The first round of a computation  $e = c_1, \dots, c_j, \dots$  is the minimal prefix  $e_1 = c_1, \dots, c_j$ , such that every enabled node in  $c_1$  either executes a rule or it is neutralized during a computation step of  $e_1$ . A node  $v$  is *neutralized* during a computation step  $c_i \rightarrow c_{i+1}$ , if  $v$  is enabled in  $c_i$  and disabled in  $c_{i+1}$ , but it did not execute any action during  $c_i$ .

Let  $e_2$  be the suffix of  $e$  such that  $e = e_1 e_2$ . The second round of  $e$  is the first round of  $e_2$ , and so on.

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

**Lemma 1.** The size of a  $k$ -independent set is at most  $\max(\lfloor 2n/k + 2 \rfloor, 1)$ .

*Proof.* Let  $I$  be a  $k$ -independent set such that  $|I| > 1$ . Let  $v$  be a node of  $I$ . We denote by  $\text{closer}(v)$  the set of nodes closer to  $v$  than any other node of  $I$ .

Notice that  $\bigcup_{v \in I} \text{closer}(v) \subset V$  and  $\text{closer}(v) \cap \text{closer}(u) = \emptyset, \forall (u, v) \in I^2$ . Let  $u$  be the closest node of  $v$  that belongs to  $I$ . The first  $\lfloor k + 2/2 \rfloor$  nodes in the path from  $v$  to  $u$  are closer to  $v$  than any other nodes of  $I$ . So,  $\text{closer}(v)$  contains at least  $\lfloor k + 2/2 \rfloor$  nodes. We conclude that  $|I| \leq \lfloor 2n/k + 2 \rfloor$ . ■

The protocol *SID* build  $k$ -independant sets. So, the obtained  $k$ -independant dominating set contains at most  $\lfloor 2n/k + 2 \rfloor$  nodes.

### 3 The protocol *SID*

In the following subsection, we gives the notation used by the protocol *SID*.

#### 3.1 $k$ -augmentedID type

**Definition 3.**  *$k$ -augmentedID type* An  $k$ -augmentedID value,  $a$ , is an uplet  $(d, x)$  such that  $d$  is integer with  $0 \leq d \leq k$ , and  $x$  is a node identifiant. Let  $a = (d, x)$  be  $k$ -augmentedID value. We note  $a.\text{dist} = d$  and  $a.\text{id} = x$

Let  $v$  be a node of  $V$ ,  $\text{id}_v^+$  it the  $k$ -augmentedID value associated to  $v$  :  $(0, \text{id}_v)$ .

**Definition 4.** *Two total order relations on  $k$ -augmentedID  $\cup \perp$*

- **dom** is defined as follow:  $\text{dom}(a, b) = a$  if  
 $b = \perp, a.\text{id} < b.\text{id}$  or  $a.\text{id} = b.\text{id} \wedge a.\text{dist} < b.\text{dist}$
- **min** is defined as follow:  $\text{min}(a, b) = a$  if  
 $b = \perp, a.\text{dist} < b.\text{dist}$  or  $a.\text{dist} = b.\text{dist} \wedge a.\text{id} < b.\text{id}$

The  $k$ -augmented value  $a1$  dominates the  $k$ -augmented value  $a2$  iff  $\text{dom}(a1, a2) = a1$ . The  $k$ -augmented value  $a1$  is larger than the  $k$ -augmented value  $a2$  iff  $\text{min}(a1, a2) = a2$ .

**Definition 5.** *The operation  $+1$  on  $k$ -augmentedID  $\cup \perp$  is defined as follow :  $a+1 = a$  if  $a = \perp$  or if  $a.\text{dist} = k$  otherwise  $a+1 = (a.\text{dist}+1, a.\text{id})$*

#### 3.2 Code of the protocol *SID*

A node  $v$  is said to be a head if  $\text{firstHead}(v) = \text{id}_v^+$ ; otherwise it is an ordinary node. The heads set built by the protocol *SID* (defined in protocol 1) is a  $k$ -independant dominating set.

---

**Protocol 1** : the Protocol *STD* on the node  $v$ 

---

**Shared variables**

- $\text{firstHead}(v) \in k\text{-augmentedID}$
- $\text{secondHead}(v) \in k\text{-augmentedID} \cup \perp$

**Internal variables**

- $k\text{-augmentedIdSet}(v) = \{a + 1 \in k\text{-augmentedID} \mid a = \text{firstHead}(u) \vee a = \text{secondHead}(u) \text{ with } u \in N_v \wedge a.\text{dist} < k \wedge a.\text{id} \neq \text{id}_v\}$
- $\text{SecondAugmentedIdSet}(v) = \{a \in \text{augmentedIdSet} \mid a.\text{id} \neq \text{firstHead}(v).\text{id}\}$

**Macro**

- $\text{headUpdate}(v) : \text{firstHead}(v) := \text{id}_v^+; \text{secondHead}(v) := \perp;$
- $\text{ordinaryUpdate}(v) :$   
     $\text{firstHead}(v) := \min(\text{augmentedIdSet}(v));$   
     $\text{secondHead}(v) := \min(\text{SecondAugmentedIdSet}(v) \cup \{\perp\});$

**Predicates**

- $\text{Head}(v) \equiv \text{firstHead}(v) = \text{id}_v^+$
- $\text{toResign}(v) \equiv \text{id}_v^+ \neq \text{dom}(\text{augmentedIdSet}(v) \cup \{\text{id}_v^+\})$
- $\text{toElect}(v) \equiv \text{augmentedIdSet}(v) = \emptyset$
- $\text{headToUpdate}(v) \equiv \text{secondHead}(v) \neq \perp$
- $\text{ordinaryToUpdate}(v) \equiv$   
     $\text{firstHead}(v) \neq \min(\text{augmentedIdSet}(v)) \vee$   
     $\text{secondHead}(v) \neq \min(\text{SecondAugmentedIdSet}(v) \cup \{\perp\})$

**Rules**

- $\text{RE}(v) : \neg \text{Head}(v) \wedge \text{toElect}(v) \longrightarrow \text{headUpdate}(v);$
  - $\text{RU}(v) : \neg \text{Head}(v) \wedge \neg \text{toElect}(v) \wedge \text{ordinaryToUpdate}(v) \longrightarrow \text{ordinaryUpdate}(v);$
  - $\text{RR}(v) : \text{Head}(v) \wedge \text{toResign}(v) \longrightarrow \text{ordinaryUpdate}(v);$
  - $\text{RC}(v) : \text{Head}(v) \wedge \neg \text{toResign}(v) \wedge \text{headToUpdate}(v) \longrightarrow \text{headUpdate}(v);$
-

The variable  $\mathbf{firstHead}(v)$  contains the identifier of the closest head to  $v$  (with its distance to  $v$ ).

The variable  $\mathbf{secondHead}(v)$  contains the identifier of the second closest head to  $v$  (with its distance to  $v$ ) inside its  $k$ -neighborhood. If a node  $v$  does not have two heads in its  $k$ -neighborhood then  $\mathbf{secondHead}(v)$  is set to  $\perp$ . The execution of the rules  $\mathbf{RU}$  or the rules  $\mathbf{RC}$  updates the two variables  $\mathbf{firstHead}(v)$ , and  $\mathbf{secondHead}(v)$  without changing the status of  $v$  (i.e.  $v$  stays ordinary or head).

The variable  $\mathbf{augmentedIdSet}(v)$  contains some heads in the  $k$ -neighborhood of  $v$ . (i.e.  $(d, id_u) \in \mathbf{augmentedIdSet}(v)$  indicates that  $u$  is a head at distance  $d$  of  $v$ ). If the  $k$ 's neighborhood of  $v$  does not contain any head then  $\mathbf{augmentedIdSet}(v)$  is empty. Notice that in this case,  $\mathbf{toElect}(v)$  is verified. So, if  $v$  is an ordinary node then  $v$  will become a head (the rule  $\mathbf{RE}$  is enabled). Therefore, the heads set is a  $k$ -dominating set, in a terminal configuration.

If  $\mathbf{toResign}(v)$  is verified then  $v$  has in its  $k$ -neighborhood a head  $u$  having a smaller identifier than  $v$ 's identifier (i.e.  $id_v > id_u$ ). In this case, if  $v$  is a head, it is enabled. So, the set of heads is a  $k$ -independent set, in any terminal configuration.

The proof of the protocol  $\mathbf{SID}$  has two parts.

- In the section 4, we prove that a terminal configuration of  $\mathbf{SID}$  protocol is legitimate : the set of heads is a  $k$ -independent dominating set.
- In the section 5, we prove that all maximal computations under any unfair distributed scheduler are finite by *reductio ad absurdam* arguments.

## 4 Correctness of the protocol $\mathbf{SID}$

In this section, we prove that all terminal configuration of  $\mathbf{SID}$  protocol are legitimate : the set of heads is a  $k$ -independent dominating set.

**Definition 6.** *The property  $\mathbf{OrdinaryPr}(i)$  defined for all  $i \in [1, k]$  has two parts :*

- *for all ordinary node  $v$ ,  $\mathbf{firstHead}(v) = (i, id_u)$  if and only if  $u$  is the closest head to  $v$  and  $i$  is the distance between  $u$  and  $v$ .*



- for all node  $v$ ,  $\text{secondHead}(v) = (i, id_u)$  if and only if  $u$  is the second closest head to  $v$  and  $i$  is the distance between  $u$  and  $v$ .

**Observation 1** In a terminal configuration,

- An ordinary node  $v$  does not verify  $\text{OrdinaryToUpdate}(v)$ ;
- A head  $v$  does not verify  $\text{HeadToUpdate}(v)$ ;
- if  $v$  is an ordinary node then  $\text{firstHead}(v).dist > 0$  ;
- if  $\text{secondHead}(v) \neq \perp$  then  $\text{secondHead}(v).dist > 0$  ;
- if  $\text{secondHead}(v) \neq \perp$  then  $\text{secondHead}(v).dist \geq \text{firstHead}(v).dist$ .

**Lemma 2.** In a terminal configuration of protocol  $STD$ , the property  $\text{OrdinaryPr}(1)$  is verified.

*Proof.*

Let  $v$  be an ordinary node, in a terminal configuration of protocol  $STD$ .  $(1, x) \in \text{augmentedIdSet}(v)$  if and only if  $v$  has a neighbor  $u$  such that  $\text{firstHead}(u) = (0, x)$  in  $c$ . According to observation 1,  $u$  is a head; so  $x = id_u$ .

Notice that  $\forall a \in \text{augmentedIdSet}(v)$ , we have  $a.dist > 0$ , in  $c$ .

As  $\text{firstHead}(v) = \min(\text{augmentedIdSet}(v))$ , if  $v$  has a head at distance 1 then  $\text{firstHead}(v) = (1, id_u)$  with  $u$  being the head in  $v$ 's neighborhood having the smallest identifiant. otherwise  $\text{firstHead}(v).dist \neq 1$ .

According to the predicate  $\text{OrdinaryToUpdate}(v)$ , if  $v$  has several heads at distance 1 then  $\text{secondHead}(v) = (1, id_u)$  with  $u$  being the head in  $v$ 's neighborhood having the second smallest identifiant.

Otherwise,  $\text{secondHead}(v).dist$  is larger than 1. ■

**Lemma 3.** Let  $i$  be a positive integer smaller than  $k$ . In a terminal configuration of protocol  $STD$ , if the properties  $\text{OrdinaryPr}(j)$  are verified for all  $j \in [1, i]$  then the property  $\text{OrdinaryPr}(i + 1)$  is verified.

*Proof.* Let us assume that the properties  $\text{OrdinaryPr}(j)$  are verified for all  $j \in [1, i]$  in any terminal configuration of protocol  $STD$ . In a terminal configuration  $c$ ,  $(j, x) \in \text{augmentedIdSet}(v)$  iff  $v$  has a neighbor  $u$  such that  $\text{firstHead}(u) = (j - 1, x)$ , or  $\text{secondHead}(u) = (j - 1, x)$ . If  $j = 1$  then  $u$  is a head in  $c$ , according to Observation 1. If  $1 < j \leq i + 1$  then  $x$  is the identifiant of a head in  $c$  at distance  $j - 1$  of  $u$ , according to the properties  $\text{OrdinaryPr}(j - 1)$ . So  $x$  is the identifiant of a head at distance at most  $j$  of  $v$ , in  $c$ .

Let  $v'$  be the closest head to  $v$  and  $d'$  the distance from  $v'$  to  $v$  in the terminal configuration  $c$ . Assume that  $0 < d' \leq i + 1$ . In  $c$ , we have the two following properties.

- if  $(l, id) \in \mathbf{augmentedIdSet}(v)$  then  $l \geq d'$ ; and
- if  $(d', id) \in \mathbf{augmentedIdSet}(v)$  then  $id \geq id_{v'}$ . In  $c$ ,  $v$  has a neighbor  $u$  at distance  $d' - 1$  to  $v'$ . In  $c$ , the node  $v'$  is the closest head of  $u$ ; so  $\mathbf{firstHead}(u) = (d' - 1, id_{v'})$ .

We conclude that  $\mathbf{firstHead}(v) = (d', id_{v'})$ , in  $c$ .

Assume that the network has several heads. Let  $v''$  be the second closest head to  $v$  and  $d''$  the distance from  $v''$  to  $v$ , in a terminal configuration  $c$ .  $v$  has a neighbor  $u$  at distance  $d'' - 1$  to  $v''$  in  $c$ . (we have  $d'' > 0$ ).  $v''$  is the first or second closest head to  $u$ , in  $c$ . Assume that  $d'' \leq i + 1$ . According to the properties  $\mathbf{OrdinaryPr}(i)$ ,  $\mathbf{firstHead}(u) = (d'' - 1, id_{v''}) \vee \mathbf{secondHead}(u) = (d'' - 1, id_{v''})$ , in  $c$ . So  $(d'', id_{v''}) \in \mathbf{augmentedIdSet}(v)$ , in  $c$ . In  $c$ , we have the two following properties.

- if  $(l, id) \in \mathbf{augmentedIdSet}(v)$  then  $l \geq d''$  or  $(l, id) = \mathbf{firstHead}(v)$ .
- if  $(d'', id) \in \mathbf{augmentedIdSet}(v)$  then  $id \geq id_{v''}$  or  $id = \mathbf{firstHead}(v).id$ .

We conclude that  $\mathbf{secondHead}(v) = (d'', id_{v''})$ . ■

The following corollary is a direct result of lemmas 2 and 3. It establishes that the set of heads is a  $k$ -dominating set.

**Corollary 1.** *Let  $v$  be a ordinary node, in a terminal configuration of protocol  $STD$ , named  $c$ .  $\mathbf{firstHead}(v).id$  is the closest head to  $v$ ; their distance is  $\mathbf{firstHead}(v).dist \leq k$ . If  $\mathbf{secondHead}(v) = \perp$  then  $v$  has a single head in its  $k$ -neighborhood otherwise  $\mathbf{secondHead}(v).id$  is the second closest head to  $v$ ; their distance is  $\mathbf{secondHead}(v).dist$ .*

The following theorem establishes that the set of heads is a  $k$ -independent set.

**Theorem 1.** *Let  $v$  be a head, in a terminal configuration of protocol  $STD$ , named  $c$ .  $v$  has not head in its  $k$ -neighborhood.*

*Proof.* We will prove that if a head has another head in its  $k$ -neighborhood then the configuration  $c$  is not terminal.

Let  $wrongHeadSet$  the set of heads having one or several heads in their  $k$ -neighborhood. Assume that  $wrongHeadSet$  is not empty. We denoted by

$v1$  the node of *wrongHeadSet* having the largest identifier. We denote  $v2$ , the closest head to  $v1$  and  $d$  the distance between  $v1$  and  $v2$ . We have  $0 < d \leq k$ . The node  $v1$  has a neighbor  $u$  at distance  $d-1$  of  $v2$ . The node  $v2$  is the first or the second closest head to  $u$ . According to corollary 1, we have  $(d, id_{v2}) \in \text{augmentedIdSet}(v1)$ .  $v1$  is enabled because  $v1$  satisfied the predicate  $\text{toResign}(v1)$ . ■

## 5 Termination of the protocol *STD*

In this section, we prove that all maximal computations of protocol *STD* under any unfair distributed scheduler are finite by *reductio ad absurdum* arguments.

Let  $e$  be a maximal computation starting from a configuration, named  $c0$ . In a configuration  $c$  reached by  $e$ , for all node  $v$ ,  $\text{firstHead}(v)_c.id$  is either the identifier of a node or this value appears in the initial configuration (i.e. there is a node  $u$ , such that  $\text{firstHead}(v)_c.id = \text{firstHead}(u)_{c0}.id \vee \text{firstHead}(v)_c.id = \text{secondHead}(u)_{c0}.id$ ). So, the value taken by a variable  $\text{firstHead}$  in  $e$  belongs to a bounded set. Similarly we prove also that the value taken by a variable  $\text{secondHead}$  in  $e$  belongs to a bounded set.

### 5.1 RR and RE rules

Along any computation, a node performs at most one time the rule **RC**.

Assume that a or several nodes perform infinitely often the action **RE** or the action **RR**. Between two consecutive actions **RE** by a node  $u$ , this node has performed on time the action **RR**. So a node  $u$  that infinitely often perform the action **RE** or the action **RR** changes its status infinitely often. We name  $u^+$  the node the smallest identifier among nodes that changes their status infinitely often.  $e$  has a suffix  $e1$  where only nodes having a identifier larger than  $id_{u^+}$  changes their status (i.e. to perform the action **RE** or the action **RR**).

As the set of value taken by  $\text{firstHead}(u^+)$  is bounded; along  $e1$ , infinitely often after the action **RR**( $u^+$ ),  $\text{firstHead}(u^+)$  has the same value, denoted  $(l+1, id)$ . Notice that  $id < id_{u^+}$  and  $0 < l < k$ . So  $u^+$  has a neighbor  $u_l$  such that, infinitely often before the action **RR**( $u^+$ ),  $u_l$  verifies  $\text{firstHead}(u_l) = (l, id)$  or  $\text{secondHead}(u_l) = (l, id)$ .

At time, where  $u^+$  becomes head, we have  $\text{augmentedIdSet}(u^+) = \emptyset$ . So, the values of  $u_l$  variables are infinitely often larger than  $(l, id)$  : So  $u_l$  gives infinitely often to one of its variables the value  $(l, id)$ , but also gives a larger value to the same variable.

Assume that  $l > 0$ . At time where  $u_l$  gives the value  $(l, id)$  to one of its variable :  $u_l$  has a neighbor  $u_{l-1}$ , having the value  $(l-1, id)$ . At time where  $u_l$  gives a larger value than  $(l, id)$  to the same variable :  $u_{l-1}$  has a larger value than  $(l-1, id)$ . We conclude that there is a series of  $l+1$  nodes :  $u_l, u_{l-1}, ..u_0$  such that  $u_i$  has infinitely often has the value  $(i, id)$  and infinitely often does not have this value along  $e1$ .

Along  $e1$ ,  $u_0$  performs infinitely often the action **RR** and the action **RE**. We have  $id = id_{u_0} < id_{u^+}$ : there is a contradiction.

So  $e$  has a suffix, named  $e2$ , in which the only rule performed is **RU**.

## 5.2 RU rule

Assume that a node or several nodes changing infinitely often their value **firstHead** or their value **secondHead** along  $e2$ . We named  $min^+$  the smallest value infinitely often allocates to the variable **firstHead** or to the variable **secondHead** of one of these nodes. Let  $e3$  be the suffix of  $e2$  in which no variable **firstHead** and no variable **secondHead** gets a value smaller than  $min^+$ . Along  $e3$ , infinitely often, a node, named  $u^+$ , performs **RU** action to set the value  $min^+$  to its variable **firstHead** or its variable **secondHead**; and infinitely often,  $u^+$  performs **RU** action to set to the same variable a value larger than  $min^+$ .

Let  $c \rightarrow c'$  be a computation step of  $e3$  where  $u^+$  performs **RU** action to set a value larger than  $min^+$  to its variable **firstHead** or to its variable **secondHead**. In  $c$ ,  $min^+$  is smaller than  $min(\text{augmentedIdSet}(u^+))$  or  $min^+$  is smaller than  $min(\text{SecondAugmentedIdSet}(u^+))$ . This property stays verified along  $e3$  :  $u^+$  never sets the value  $min^+$  to its variable **firstHead** (resp. to its variable **secondHead**). There is a contradiction.

We have established that  $e2$  has a suffix  $e4$  where no rule is executed. A terminal configuration is reached.

## 6 Conclusion

A simple and silent self-stabilizing protocols building  $k$ -independant dominating sets is presented. The obtained  $k$ -independant dominating set contains at most  $\lfloor 2n/k + 2 \rfloor$  nodes. The protocol converges under the unfair distributed scheduler (the weakest scheduling assumption). The protocol is memory efficient : it requires only  $2\log(n + 1)(k + 1)$  bits per node.

The computation of the convergence time of the protocol *STD* is opened question.

## References

1. D. Bein, A. K. Datta, C. R. Jagganagari, and V. Villain. A self-stabilizing link-cluster algorithm in mobile ad hoc networks. In *International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN'05)*, pages 436–441, 2005.
2. A. Bui, S. Clavière, A. K. Datta, L. L. Larmore, and D. Sohier. Self-stabilizing hierarchical construction of bounded size clusters. In *18th International Colloquium Structural Information and Communication Complexity (SIROCCO'11)*, Springer LNCS 6796, pages 54–65, 2011.
3. E. Caron, A. K. Datta, B. Depardon, and L. L. Larmore. self-stabilizing  $k$ -clustering algorithm for weighted graphs. *Journal of Parallel and Distributed Computing*, 70:1159–1173, 2010.
4. A. Datta, S. Devismes, and L. Larmore. A self-stabilizing  $o(n)$ -round  $k$ -clustering algorithm. In *28th IEEE Symposium on Reliable Distributed Systems (SRDS'09)*, pages 147–155, 2009.
5. A. K. Datta, L. L. Larmore, S. Devismes, K. Heurtefeux, and Y. Rivierre. Competitive self-stabilizing  $k$ -clustering. In *IEEE 32th International Conference on Distributed Computing (ICDCS'12)*, pages 476–485, 2012.
6. A. K. Datta, L. L. Larmore, S. Devismes, K. Heurtefeux, and Y. Rivierre. Self-stabilizing small  $k$ -dominating sets. *International Journal of Networking and Computing*, 3(1):116–136, 2013.
7. A. K. Datta, L. L. Larmore, and P. Vemula. A self-stabilizing  $o(k)$ -time  $k$ -clustering algorithm. *The Computer Journal*, 53(3):342–350, 2010.
8. M. Demirbas, A. Arora, V. Mittal, and V. Kulathumani. A fault-local self-stabilizing clustering service for wireless ad hoc networks. *IEEE Trans. Parallel Distrib. Syst.*, 17(9):912–922, 2006.
9. S. Dolev and N. Tzachar. Empire of colonies self-stabilizing and self-organizing distributed algorithms. In *the 10th International Conference On Principles Of Distributed Systems (OPODIS'06)*, Springer LNCS 4305, pages 230–243, 2006.
10. V. Drabkin, R. Friedman, and M. Gradinariu. Self-stabilizing wireless connected overlays. In *Conference On Principles Of Distributed Systems (OPODIS'06)*, Springer LNCS 4305, pages 425–439, 2006.
11. W. Goddard, S. T. Hedetniemi, D. P. Jacobs, and P. K. Srimani. Self-stabilizing protocols for maximal matching and maximal independent sets for ad hoc networks. In *the 5th IPDPS Workshop on Advances in Parallel and Distributed Computational Models (WAPDCM'03)*, 2003.

12. C. Johnen. Fast self-stabilizing  $k$ -independent dominating set construction. Technical Report RR-1472-13, Univ. Bordeaux, LaBRI, UMR 3800, F-33400 Talence, France, June 2013.
13. C. Johnen and F. Mekhaldi. Robust self-stabilizing construction of bounded size weight-based clusters. In *the 16th International Euro-Par Conference (Euro-Par'10)*, Springer LNCS 6271, pages 535–546, 2010.
14. C. Johnen and L. H. Nguyen. Self-stabilizing construction of bounded size clusters. In *the IEEE 8th International Symposium on Parallel and Distributed Processing and Applications (ISPA'08)*, pages 43–50, 2008.
15. C. Johnen and L. H. Nguyen. Robust self-stabilizing weight-based clustering algorithm. *Theoretical Computer Science*, 410(6-7):581–594, 2009.
16. S. Kamei and H. Kakugawa. A self-stabilizing approximation for the minimum connected dominating set with safe convergence. In *the 12th Conference On Principles Of Distributed Systems (OPODIS'08)*, Springer, LNCS 5401, pages 496–511, 2008.
17. A. Larsson and P. Tsigas. A self-stabilizing  $(k,r)$ -clustering algorithm with multiple paths for wireless ad-hoc networks. In *IEEE 31th International Conference on Distributed Computing Systems, (ICDCS'11)*, pages 353–362. IEEE Computer Society, 2011.
18. A. Larsson and P. Tsigas. Self-stabilizing  $(k,r)$ -clustering in clock rate-limited systems. In *19th International Colloquium Structural Information and Communication Complexity, (SIROCCO'12)*, Springer, LNCS 7355, pages 219–230, 2012.
19. N. Mitton, E. Fleury, I. Guérin-Lassous, and S. Tixeuil. Self-stabilization in self-organized multihop wireless networks. In *International Conference on Distributed Computing Systems Workshops (WWAN'05)*, pages 909–915, 2005.
20. Z. Xu, S. T. Hedetniemi, W. Goddard, and P. K. Srimani. A synchronous self-stabilizing minimal domination protocol in an arbitrary network graph. In *Proceedings of the 5th International Workshop on Distributed Computing (IWDC'03)*, Springer LNCS 2918, pages 26–32, 2003.