Comparison Between Self-Stabilizing Clustering Algorithms in Message-Passing Model.

Mandicou BA, Olivier FLAUZAC, Rafik MAKHLOUFI, Florent NOLOT
Université de Reims Champagne-Ardenne
CReSTIC - SysCom (EA 3804)
Reims, France
{mandicou.ba, olivier.flauzac, rafik.makhloufi, florent.nolot}@univ-reims.fr

Ibrahima NIANG
Université Cheikh Anta Diop
Laboratoire d'Informatique de Dakar (LID)
Dakar, Sénégal
iniang@ucad.sn

***Abstract*—Most Ad Hoc networks use diffusion to communicate. This approach requires many messages and may cause network saturation. To optimize these communications, one solution consists in structuring networks into clusters. In this paper, we present a new self-stabilizing asynchronous distributed algorithm based on message-passing model. We compare the proposed algorithm with one of the best existing solutions based on message-passing model. Our approach does not require any initialization and builds non-overlapping k-hops clusters. It is based only on information from neighboring nodes with periodic messages exchange. Starting from an arbitrary configuration, the network converges to a stable state after a finite number of steps. A legal configuration is reached after at most $n+2$ transitions and uses at most $n * \log(2n+k+3)$ memory space, where n is the number of network nodes. Using the *OMNeT++* simulator, we performed an evaluation of the proposed algorithm to notably show that we use fewer messages and stabilizing time is better.**

***Keywords*-ad hoc networks; clustering; distributed algorithms;self-stabilizing; OMNeT++ simulator**

I. Introduction

In Ad Hoc networks, the most frequently used communication solution is diffusion. This is a simple technique that requires few calculations. But this method is expensive and may cause network saturation. In order to optimize this communication, which is an important source of resource consumption, one solution is to structure the network in *trees* [1] or *clusters* [2].

Clustering consists in organizing the network into groups of nodes called clusters, thus giving a hierarchical structure [3]. Each cluster is managed by a particular node called clusterhead. A node is elected clusterhead using a metric such as the mobility degree, node's identity, node's density, etc. or a combination of these parameters. Several solutions of clustering have been proposed. They are classified into 1-hop and k-hops algorithms. In 1-hop solutions [4], [5], [6], [7] nodes are at a distance of *1* from the clusterhead and the maximum diameter of clusters is *2*. However, in k-hops solutions [8], [9] nodes can be located at a distance of *k* from the clusterhead and the maximum diameter of clusters is *2k*. However, these approaches, generate a lot of traffic and require considerable resources.

In this paper, we propose a self-stabilizing asynchronous distributed algorithm that builds k-hops clusters. Dijkstra defined a distributed system to be self-stabilizing if, regardless of the initial state, the system is guaranteed to reach a legitimate (correct) state in a finite time [10]. Our approach builds non-overlapping k-hops clusters and does not require initialization. It is based on the criterion of maximum identity attached to the nodes for clusterhead selection and relies only on the periodic exchange of messages with the 1-hop neighborhood. The choice of the identity metric provides more stability against dynamic criteria such as mobility degree and weight of nodes.

The remainder of the paper is organized as follows. In Section II, we describe some related works of self-stabilizing clustering solutions. Section III presents our contribution. In Section IV, we describe the computational model used in the paper and give some additional concepts. In Section V, we first present a broad and intuitive explanation of the algorithm before defining it more formally. In Section VI, we present performance evaluation conducted with the *OMNeT++* simulator. Finally, we conclude and present some future work in Section VII.

II. Related work

Several clustering solutions have been done in the literature [4], [5], [6], [7], [8], [11], [9]. Approaches [6], [7], [11], [9] are based on state model at opposed of message-passing model algorithms [4], [5], [8].

Self-stabilizing algorithms presented in [4], [5], [6], [7] are 1-hop clustering solutions.

A metric called *density* is used by Mitton et *al.* in [4], in order to minimize the reconstruction of structures for low topology change. Each node calculates its density and broadcasts it to its neighbors located at 1-hop. For the maintenance of clusters, each node calculates periodically its mobility and density.

Flauzac et *al.* [5], have proposed a self-stabilizing clustering algorithm, which is based on the identity of its neighborhood to build clusters. This construction is done using the identities of each node that are assumed unique. The advantage of this algorithm is to combine in the same phase the neighbors discovering and the clusters establishing. Moreover, this deterministic algorithm constructs disjoint clusters, i.e., a node is always in only one cluster.

In [6], Johnen et *al.* have proposed a self-stabilizing protocol designed for the state model to build 1-hop clusters whose size is bounded. This algorithm guarantees that the network nodes are partitioned into clusters where each one has at most $SizeBound$ nodes. The clusterheads are chosen according to their $weight$ value. In this case, the node with the higher weight becomes clusterhead. In [7], Johnen et *al.* have extended the proposal from [6]. They have proposed a robust self-stabilizing weight-based clustering algorithm. The robustness property guarantees that, starting from an arbitrary configuration, after one asynchronous round, the network is partitioned into clusters. After that, the network stays partitioned during the convergence phase toward a legitimate configuration where clusters verify the ad hoc clustering properties.

Self-stabilizing algorithms proposed in [8], [11], [9] are k-hops clusters solutions.

In [11], using criterion of minimal identity, Datta et *al.* have proposed a self-stabilizing distributed algorithm designed for the state model that computes a subset D is a minimal *k-dominating* set of graph G. Using D as the set of *clusterheads*, a partition of G into clusters, each of radius k, follows. This algorithm converges in $O(n)$ rounds and $O(n^2)$ steps and requires $\log(n)$ memory space per process , where n is the size of the network.

Datta et *al.* [9], using an arbitrary metric, have proposed a self-stabilizing *k-clustering* algorithm base on a state model. Note that *k-clustering* of a graph is a partition of nodes into disjoints clusters in which every node is at a distance of at most k from the *clusterhead*. This algorithm executes in $O(nk)$ rounds and requires $O(log(n) + log(k))$ memory space per process, where n is the network size.

In [8], Miton et *al.* applied self-stabilization principles over a clusterization protocol proposed in [4] and presents properties of robustness. Each node calculates its density and broadcasts it to its neighbors located at k-hops. This robustness is an issue related to the dynamicity of ad hoc networks, to reduce the time stabilization and to improve network stability.

III. CONTRIBUTION

We propose a self-stabilizing asynchronous distributed algorithm that builds k-hops clusters. Our approach is based on a message-passing model as opposed to solutions proposed in [6], [7], [11], [9]. We use the criterion of maximum identity that brings more stability compared to metric variables used in [8], [4], [6], [7]. Our algorithm structures the network into non-overlapping clusters with a diameter at most equal to $2k$. This structuring does not require any initialization. It is based only on information from neighboring nodes. Contrary to other clustering algorithms, we use an unique message to discover the neighborhood of a node at distance 1 and to structure the network into *k-hops* clusters. Starting from an arbitrary configuration, the network converges to a legal configuration after a finite number of steps. A legal configuration is reached after at most $n + 2$ transitions and requires at most $n * \log(2n+k+3)$ memory space, where n is the number of network nodes. Using the *OMNeT++* simulator, we performed an evaluation of the proposed algorithm and a comparison with one of the best existing solution based on message-passing model [8]. We show that we use less messages than and stabilizing time is better.

IV. MODEL

We consider our network as a distributed system that can be modeled by an undirected graph $G = (V, E)$. $V = n$ is the set of network nodes and E represents all existing connections between nodes. An edge (u, v) exists if and only if u can communicate with v and vice-versa. This means that all links are bidirectional. In this case, the nodes u and v are neighbors. The set of neighbors $v \in V$ of node u is marked N_u. Each node u of the network has a unique identifier id_u and can communicate with N_u. We define the distance $d_{(u,v)}$ between nodes u and v in the graph G as the minimum number of edges along the path between u and v.

Our algorithm is designed for the asynchronous message-passing model following standard models for distributed systems given in [12], [13]. For this purpose, each pair of nodes is connected by a bi-directionnal link. Links are asynchronous and messages transit time is finite but not bounded. Moreover links are reliable. They do not create, corrupt or lose messages. Furthermore, each node u periodically sends to its neighbors a message that is received correctly within some finite but unpredictable time by all its 1-hop neighbors. Each node u maintains a table containing the current state of its neighbors at distance 1. Upon receiving a message, a node u executes our clustering algorithm.

V. SELF-STABILIZING K-HOPS CLUSTERING ALGORITHM

A. Preliminaries

We give some definitions used in the following.

Definition 5.1: (Cluster) We define a *k-hops* cluster as a connected graph in the network, with a diameter less than or equal to $2k$. The set of all the nodes of a cluster i is denoted V_i.

Definition 5.2: (Cluster identifier) Each cluster has an unique identifier corresponding to the greatest node identity in its cluster. The identity of a cluster that owns a node u is denoted cl_u.

In our clusters, each node u has a status noted $status_u$. Thus, a node can be clusterhead (CH), a *Simple Node* (SN) or a *Gateway Node* (GN). Moreover, each node selects a neighbor $v \in N_u$, noted gn_u, through which it passes to reach its CH.

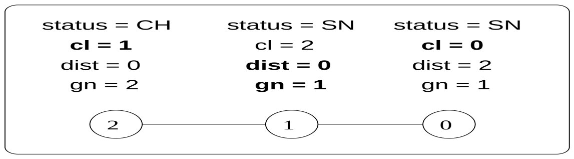
Definition 5.3: (Nodes status)

- *Clusterhead* (**CH**): a node u has CH status if it has the highest ID among all nodes of its cluster.
  - $status_u = CH \iff \forall v \in V_{cl_u}, (id_u > id_v) \wedge (dist_{(u,v)} \leq k)$.
- *Simple Node* (**SN**): a node u has SN status if u and all its neighbors are in the same cluster.
  - $status_u = SN \iff (\forall v \in N_u, cl_v = cl_u) \wedge (\exists w \in V_u/(status_w = CH) \wedge (dist_{(u,w)} \leq k))$.

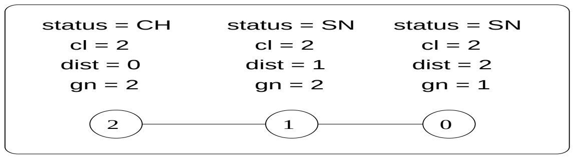
- *Gateway Node* (**GN**): a node u has GN status if exists a node v in neighborhood in a different cluster.
  - $status_u = GN \iff \exists v \in N_u, (cl_u \neq cl_v)$.

Definition 5.4: (Node Coherence)
A node u is *coherent node if and only if* it is in one of the following states:

- if $status_u = CH$ then $(cl_u = id_u) \wedge (dist_{(u,CH_u)} = 0) \wedge (gn_u = id_u)$,
- if $status_u \in \{SN, GN\}$ then $(cl_u \neq id_u) \wedge (dist_{(u,CH_u)} \neq 0) \wedge (gn_u \neq id_u)$.



(a) Incherence nodes



(b) Coherence nodes

Fig. 1. Coherent and incoherent nodes

Definition 5.5: (Node stability)
A node u is *stable node* if and only if its variables no longer change, it is coherent and satisfies the following states:

- if $status_u = CH$ then $\forall v \in N_u, (status_v \neq CH) \wedge \{((cl_v = cl_u) \wedge (id_v < id_u)) \vee ((cl_v \neq cl_u) \wedge (dist_{(v,CH_v)} = k))\}$. (Example of node 9 in cluster V_9)
- if $status_u = SN$ then $\forall v \in N_u, (cl_v = cl_u) \wedge (dist_{(u,CH_u)} \leq k) \wedge (dist_{(v,CH_v)} \leq k)$. (Example of node 0 in cluster V_{10} or node 7 in V_9).
- if $status_u = GN$ then $\exists v \in N_u, (cl_v \neq cl_u) \wedge \{((dist_{(u,CH_u)} = k) \wedge (dist_{(v,CH_v)} \leq k)) \vee ((dist_{(v,CH_v)} = k) \wedge (dist_{(u,CH_u)} \leq k))\}$. (Example of node 2 in cluster V_9 or node 8 in V_{10}).

Definition 5.6: (Network stability)
The network is *stable* if and only if all nodes are stable nodes. (see figure 2)

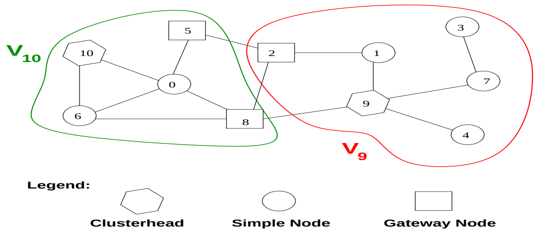


Fig. 2. Stable Nodes - Stable network

B. Basic Idea of Our Solution

Our algorithm is self-stabilizing and does not require any initialization.

Starting from any arbitrary configuration, with only one type of message exchanged, the nodes are structured in non-overlapping clusters in a finite number of steps. This message is called *hello message* and it is periodically exchanged between each neighbor nodes. It contains the following four items information: node identity (id_u), cluster identity (cl_u), node status $(status_u)$ and the distance to clusterhead $dist_{(u,CH_u)}$. Note that cluster identity is also the identity of the clusterhead. Thus, the hello message structure is $hello(id_u, cl_u, status_u, dist_{(u,CH_u)})$. Furthermore, each node maintains a neighbor table $StateNeigh_u$ that contains the set of its neighboring nodes states. Whence, $StateNeigh_u[v]$ contains the states of nodes v neighbor of u.

The solution that we propose proceeds as follows:

As soon as a node u receives a hello message, it executes Algorithm 1. During this algorithm, u executes these three steps consecutively. The first step is to update neighborhood, the next step is to manage the coherence and the last step is to build the clusters. After this three steps, u sends a hello message to its neighbors.

After updating the neighborhood, nodes check their coherency. For example, as a clusterhead has the highest identity, if a node u has CH status, its cluster identity must be equal to its identity. In Figure 1(a), node 2 is clusterhead. Its identity is 2 and its cluster identity is 1, so node 2 is not a coherent node. Similarly for nodes 1 and 0, because they do not satisfy definition 5.4. Each node detects its incoherence and corrects its during the coherence management step. Figure 1(b) shows nodes that are coherent.

During the clustering step, each node compares its identity with those of its neighbors at distance 1. A node u elects itself as a clusterhead if it has the highest identity among all nodes of its cluster. If a node u discovers a neighbor v with a highest identity then it becomes a node of the same cluster as v with SN status. If u receives again a hello message from another neighbor that is in another cluster than v, the node u becomes gateway node with GN status. As the hello message contains the distance between each node u and its clusterhead, u knows if the diameter of cluster is reached. So it can choose another cluster.

C. k-hops self-stabilizing algorithm

Each node u of the network knows the k parameter value and executes the Algorithm 1.

VI. Performance Analysis

In [14], we have proved that our network is stable after at most $n + 2$ transitions and requires at most $n * \log(2n + k + 3)$ memory space. This reflects the worst case scenario of a topology where the nodes form an ordered chain. Ad Hoc networks are often characterized by random topologies.

In order to evaluate the average performance of our solution in a random topology, we have implemented our algorithm using the *OMNeT++* environment simulation [15]. For generating random graphs, we have used *SNAP* library [16]. All simulations were carried out using *Grid'5000* platform [17].

Algorithm 1: k-hops clustering algorithm

```
/* Upon receiving message from a neighbor */
Predicates
$P_1(u) \equiv (status_u = CH)$
$P_2(u) \equiv (status_u = SN)$
$P_3(u) \equiv (status_u = GN)$
$P_{10}(u) \equiv (cl_u \neq id_u) \vee (dist_{(u,CH_u)} \neq 0) \vee (gn_u \neq id_u)$
$P_{20}(u) \equiv (cl_u = id_u) \vee (dist_{(u,CH_u)} = 0) \vee (gn_u = id_u)$
$P_{40}(u) \equiv$
$\forall v \in N_u, (id_u > id_v) \wedge (id_u \geq cl_v) \wedge (dist_{(u,v)} \leq k)$
$P_{41}(u) \equiv \exists v \in N_u, (status_v = CH) \wedge (cl_v > cl_u)$
$P_{42}(u) \equiv \exists v \in N_u, (cl_v > cl_u) \wedge (dist_{(v,CH_v)} < k)$
$P_{43}(u) \equiv \forall v \in N_u/(cl_v > cl_u), (dist_{(v,CH_v)} = k)$
$P_{44}(u) \equiv \exists v \in N_u, (cl_v \neq cl_u) \wedge \{(dist_{(u,CH_u)} =$
$k) \vee (dist_{(v,CH_v)} = k)\}$

Macros
$NeighCH_u = \{id_v/v \in N_u \wedge statut_v = CH \wedge cl_u = cl_v\}.$
$NeighMax_u =$
$(Max\{id_v/v \in N_u \wedge statut_v \neq CH \wedge cl_u = cl_v\}) \wedge$
$(dist_{(v,CH_u)} = Min\{dist_{(x,CH_u)}, x \in N_u \wedge cl_x = cl_v\}).$

Rules
 /* Update neighborhood */
$StateNeigh_u[v] := (id_v, cl_v, status_v, dist_{(v,CH_v)});$

 /* Cluster-1: Coherent management */
$R_{10}(u) :: P_1(u) \wedge P_{10}(u)$
$\longrightarrow cl_u := id_u; gn_u = id_u; dist_{(u,CH_u)} = 0;$
$R_{20}(u) :: \{P_2(u) \vee P_3(u)\} \wedge P_{20}(u) \longrightarrow$
$status_u := CH; cl_u := id_u; gn_u = id_u; dist_{(u,CH_u)} = 0;$

 /* Cluster-2: Clustering */
$R_{11}(u) :: \neg P_1(u) \wedge P_{40}(u) \longrightarrow$
$status_u := CH; cl_u := id_v; dist_{(u,CH_u)} := 0; gn_u := id_u;$
$R_{12}(u) :: \neg P_1(u) \wedge P_{41}(u) \longrightarrow status_u := SN; cl_u :=$
$id_v; dist_{(u,v)} := 1; gn_u := NeighCH_u;$
$R_{13}(u) :: \neg P_1(u) \wedge P_{42}(u) \longrightarrow status_u := SN; cl_u :=$
$cl_v; dist_{(u,CH_u)} := dist_{(v,CH_v)} + 1; gn_u := NeighMax_u;$
$R_{14}(u) :: \neg P_1(u) \wedge P_{43}(u) \longrightarrow$
$status_u := CH; cl_u := id_v; dist_{(u,CH_u)} := 0; gn_u := id_u;$
$R_{15}(u) :: P_2(u) \wedge P_{44}(u) \longrightarrow status_u := GN;$
$R_{16}(u) :: P_1(u) \wedge P_{41}(u) \longrightarrow status_u := SN; cl_v :=$
$id_v; dist_{(u,v)} := 1; gn_u := NeighCH_u;$
$R_{17}(u) :: P_1(u) \wedge P_{42}(u) \longrightarrow status_u := SN; cl_u :=$
$cl_v; dist_{(u,CH_u)} := dist_{(v,CH_v)} + 1; gn_u := NeighMax_u;$

/* Sending hello message */
$R_0(u) :: hello(id_u, cl_u, status_u, dist_{(u,CH_u)});$
```

A. *Impact of density and network size on the stabilization time*

First, we study the impact of nodes degree and network size on the stabilization time. In figure 3(a), we have fixed a hops number $k = 2$. For each node degree of 3, 5 and 7 we consider a network size from 100 to 1000 nodes. Note that we generate *d-regular graphs models* using *SNAP* library, where d represents node's degree (number of neighbors for each node). For each given network size, we compute several series of simulations. We give the average stabilization time as the average of all values corresponding to simulation results. We note that the stabilization time increases as the number of nodes in the network increases. Furthermore, we note that for arbitrary topologies, the average stabilization time is below $n + 2$, formal value proved in the worst case. Moreover, the number of transitions needed to reach a legal configuration appears stable when the network size increases (500 to 1000 nodes).

To observe the impact of the network density as illustrated in figure 3(b), we consider a network size of 100, 200 and 400 nodes and we vary the nodes degree. We observe that the stabilization time decreases as the nodes degree increases. The main reason is due to the fact that each node has more neighbors, thus during each transition, we have more nodes that fixed at the same time. With our approach, we have a better stabilization time with networks of high density.

B. *Scalability*

To examine the scalability of the proposed solution, we vary the number of nodes in the network at the same time as the density of connectivity. For $k = 2$, we consider a network size of 100 to 1000 nodes. For each value of the network size, we vary the density from 10% to 100%. Note that we generate *Erdös-Renyi random graphs models* using *SNAP* library. We obtain the 3D curve illustrated in figure 4. We note that except for low densities (10% and 20%), the stabilization time varies slightly with the increasing number of nodes. In case of a low network density, we observe a peak that is due to longer chains in the network topology. With these series of simulation, we can make two remarks. (*i*) The only determining factor with our approach is the density of connectivity and our solution is scalable. (*ii*) On average, for networks with an arbitrary topology, the stabilization time is far below that of the worst case ($n + 2$ transitions).

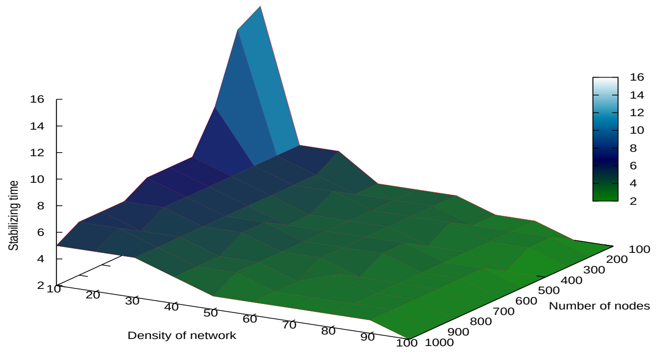
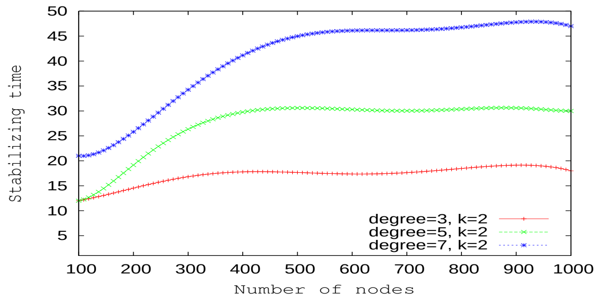


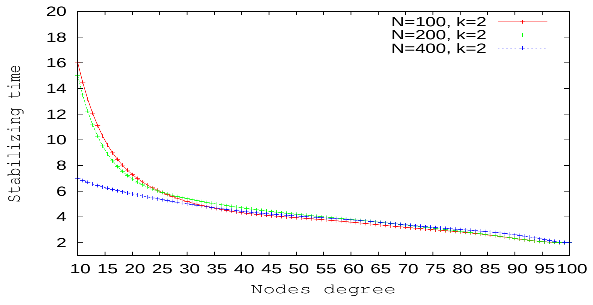
Fig. 4. Scalability

C. *Size and number of clusters*

As the density of connectivity is the determining factor for our algorithm, we evaluate the number of clusters obtained according to the network density. For $k = 2$, we consider a network size of 100, 500 and 1000 nodes. We vary the node degree from 5 to 100 neighbors. Figure 5(a) shows that regardless the number of nodes in network, we get less clusters when the number of neighbors increases. In fact, in denser

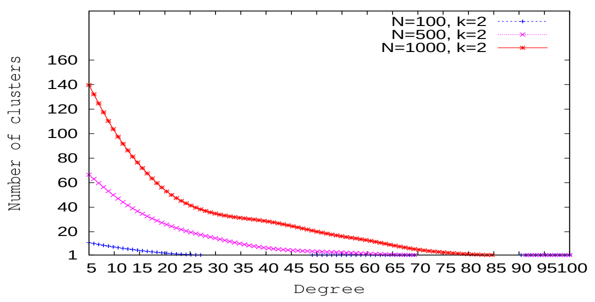


(a) Stabilization time according to the number of nodes

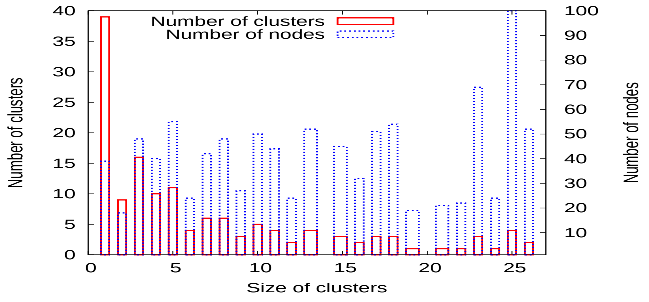


(b) Stabilization time according to nodes degree

Fig. 3. Impact of nodes degree on the stabilization time



(a) Number of clusters according nodes degree



(b) Size and number of clutsers

Fig. 5. Size and number of clusters

networks, nodes with the largest identity absorb more nodes into their clusters.

As we have more clusters with low density, we consider a network size of 1000 nodes with 5 neighbors for each node. We evaluate nodes distribution between clusters. We note that, as illustrates in figure 5(b), we have clusters of variable size. We have 39 singleton clusters, around 4% of the total number of nodes. We also note that the highest identity clusters include the more nodes its. The main reason is due to the fact that nodes choose as CH those with the highest identity.

D. Impact of the k parameter

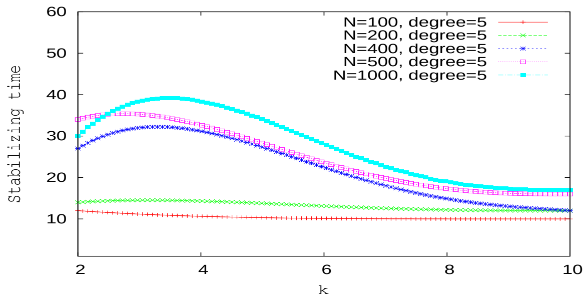


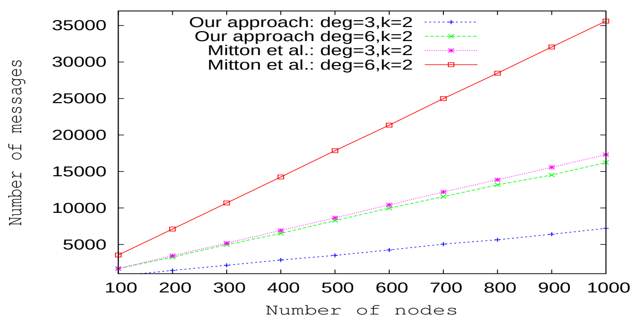
Fig. 6. Impact of k parameter

In order to observe the impact of the k parameter, we fix the node degree at 5 and we consider a network size of 100, 200, 400, 500 and 1000 nodes. For each network size, we vary the k parameter from 2 to 10. Figure 6 shows the stabilization time according to the variation of the k parameter. We observe that the stabilization time decreases as the k parameter increases. In fact, if k parameter increases and because the hello message contains the distance between each node u and its clusterhead, the sphere of influence of the largest nodes increase. Thus, nodes carrying fewer transitions to be fixed at a CH. In the end, we have fewer clusters. Nevertheless, in the case of a small value of the k parameter, we have more clusters with small diameters. Therefore, it requires more transitions to reach a stable state in all clusters. Note that regardless the value of the k parameter, the stabilization time is far below that of the worst case scenario ($n + 2$ transitions).

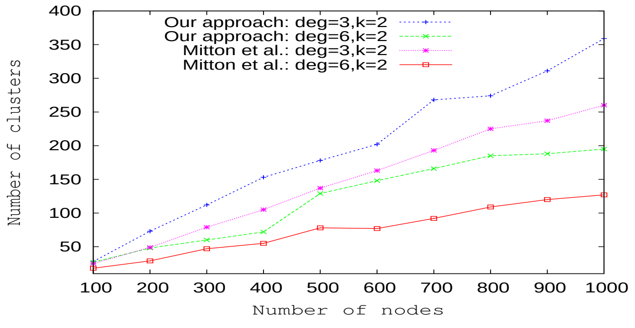
E. Comparison with a well known solution based on message-passing model

In message-passing model, their exist few solutions. We compare our approach with a well known existing solution in message-passing model [8]. We compare these two approaches in terms of number of messages exchanged (*i.e* received messages) and number of clusters. We have implemented our algorithm and [8] on OMNeT++ environment simulation. Simulations are made within the same random graph.

In order to evaluate number of exchanged messages, we fix the node degree at 3 and 6 and we consider a network size from 100 to 1000 nodes. For each given network size, we compute several series of simulations. We give the average number of



(a) Number of exchanged messages



(b) Number of clusters

Fig. 7. Comparison with Mitton et al. [8]

exchanged messages in the network to achieve the legal state. Figure 7(a), show that our approach generates less messages than [8]. The main reason is due to the fact that our approach is based only on information from neighboring at distance 1 to build *k-hops* clusters. As opposed to solution [8], each node must know $\{k+1\}$-*Neighboring*, computes its *k-density* value and locally broadcasts it to all its *k-neighbors*. This is very expensive in terms of exchanged messages.

We also evaluate the number of clusters obtained by both approaches. As illustrates in figure 7(b), our approach built more than clusters. Moreover it generates less messages. This implies that our clusters are less densely. Therefore, easy to manage by the clusterhead and consumes fewer resources. Less densely clusters leads to a decrease communications and an optimization of resource consumption.

Note that we have made comparisons for $k = 2$. In fact, if k parameter increase, the broadcast area of solution [8] increases. Whereas our approach is based only on a neighborhood of a node at distance 1 to structure the network into non-overlapping *k-hops* clusters.

VII. Conclusion

We presented a self-stabilizing asynchronous distributed algorithm based on a message-passing model. The proposed solution structures the network into non-overlapping clusters with diameter at most equal to $2k$. This structuring does not require any initialization. Furthermore, it is based only on information from neighboring nodes at distance 1. Contrary to other clustering algorithms, we have used an unique message to discover the neighborhood of a node at distance 1 and to structure the network into non-overlapping *k-hops* clusters.

Starting from an arbitrary configuration, the network converges to a legal configuration after at most $n+2$ transitions and requires at most $n * \log(2n+k+3)$ memory space. Experimental results show that for arbitrary topology networks, the stabilization time is far below the worst case scenario. A comparison with one of the best existing solution based on message-passing model show that we use fewer messages and stabilizing time is better.

As future work, we plan to implement mechanisms to balance clusters and maintaining the formed ones in case of topology change.

VIII. Acknowledgements

This work is supported in part by Regional Council of Champagne-Ardenne and European Regional Development Fund. The simulation are executed on Grid'5000 experimental testbed, hosted at the ROMEO HPC Center.

The authors wish to thank the reviewers and editors for their valuable suggestions and expert comments that help improve the contents of paper.

References

[1] L. Blin, M. G. Potop-Butucaru, and S. Rovedakis, "Self-stabilizing minimum degree spanning tree within one from the optimal degree," JPDC, 2011, pp. 438 – 449.
[2] A. Datta, L. Larmore, and P. Vemula, "Self-stabilizing leader election in optimal space," in SSS, 2008, pp. 109–123.
[3] C. Johnen and L. Nguyen, "Self-stabilizing weight-based clustering algorithm for ad hoc sensor networks," in ALGOSENSORS, 2006, pp. 83–94.
[4] N. Mitton, A. Busson, and E. Fleury, "Self-organization in large scale ad hoc networks," in MED-HOC-NET, 2004.
[5] O. Flauzac, B. S. Haggar, and F. Nolot, "Self-stabilizing clustering algorithm for ad hoc networks," ICWMC, 2009, pp. 24–29.
[6] C. Johnen and L. Nguyen, "Self-stabilizing construction of bounded size clusters," ISPA, 2008, pp. 43–50.
[7] C. Johnen and L. H. Nguyen, "Robust self-stabilizing weight-based clustering algorithm," TCS, 2009, pp. 581 – 594.
[8] N. Mitton, E. Fleury, I. Guerin Lassous, and S. Tixeuil, "Self-stabilization in self-organized multihop wireless networks," in ICDCSW, 2005, pp. 909–915.
[9] E. Caron, A. K. Datta, B. Depardon, and L. L. Larmore, "A self-stabilizing k-clustering algorithm for weighted graphs," JPDC, 2010, pp. 1159–1173.
[10] E. W. Dijkstra, "Self-stabilizing systems in spite of distributed control," Commun. ACM, 1974, pp. 643–644.
[11] A. K. Datta, S. Devismes, and L. L. Larmore, "A self-stabilizing O(n)-round k-clustering algorithm," in SRDS, 2009, pp. 147–155.
[12] H. Attiya and J. Welch, Distributed computing: fundamentals, simulations, and advanced topics. John Wiley & Sons, 2004.
[13] G. Tel, Introduction to Distributed Algorithms. Cambridge University Press, 2000.
[14] M. Ba, O. Flauzac, B. S. Haggar, F. Nolot, and I. Niang, "Self-stabilizing k-hops clustering algorithm for wireless ad hoc networks," in ACM IMCOM, 2013.
[15] A. Varga and R. Hornig, "An overview of the OMNeT++ simulation environment," in Simutool, 2008, pp. 60:1–60:10.
[16] SNAP: Stanford Network Analysis Platform. http://snap.stanford.edu
[17] F. Cappello et *al.*, "Grid'5000: A large scale and highly reconfigurable grid experimental testbed," in GRID, 2005, pp. 99–106.