Self-stabilization versus Robust Self-stabilization for clustering in Ad-hoc network

Colette Johnen[1] and Fouzi Mekhaldi[2]

[1] LaBRI, Univ. Bordeaux, CNRS. F-33405 Talence Cedex, France
[2] LRI, Univ. Paris-Sud XI, CNRS. F-91405 Orsay Cedex, France

Abstract. In this paper, we compare the two fault tolerant approaches: self-stabilization and robust self-stabilization, and we investigate their performances in dynamic networks. We study the behavior of four clustering protocols; two self-stabilizing `GDMAC` and `BSC`, and their robust self-stabilizing version `R-GDMAC` and `R-BSC`. The performances of protocols are compared in terms of their cluster-heads number, availability of both minimal and optimum services and the stabilization time.

Key words. Ad-hoc networks, clustering, self-Stabilization, Robust self-stabilization.

1 Introduction

A mobile ad-hoc network is a multi-hop wireless communication network, supporting mobile users, realised without any existing infrastructure. In a flat architecture of ad-hoc network, all nodes are considered equal and they take the same part in the network management, like routing and forwarding tasks. To achieve the routing in flat architecture, each node maintains a routing table with entries for all nodes in the network. Moreover, owing to the lack of infrastructure, each node must relay data packets of all its neighbors. Hence, flat routing protocols in ad-hoc networks are not scalable, due to the communication cost, size of routing tables and, energy consumption. Therefore, clustering was introduced in ad-hoc networks to improve the scalability by allowing hierarchical routing.
Clustering is a hierarchical network organization which consists in partitioning the network into clusters, such that nodes within a closed proximity form a cluster. Each cluster is composed of a single cluster-head and some ordinary nodes. As nodes are mobile, the clustering protocol must maintain the clustering structure in spite of topological changes like nodes arrival/departure and links creation/failure.

Self-stabilization and Robust self-stabilization. One of the most wanted properties of distributed systems is the fault tolerance and adaptivity to topological changes, which consist of the system's ability to react to faults and perturbations in a well-defined manner. Self-stabilization is an approach to design fault-tolerant and adaptive to topological changes distributed systems.
A self-stabilizing protocol, regardless of its initial state, converges in finite time (called stabilization period) to a legitimate state where the intended behavior is

exhibited, without any external intervention. Thus, self-stabilizing protocols are attractive because they do not require any correct initialization (as any state can be the initial one), they can recover from any transient failure, and they are adaptive to dynamic topology reconfigurations. Whatever is the current configuration, the system converges to a legitimate configuration according to the current network topology.
Despite such advantages, self-stabilization has a major drawback. During all stabilization periods, a self-stabilizing protocol does not guarantee any property. Thus, self-stabilization is suited for distributed systems with intermittent disruptions, where the delay between two successive disruptions is so large that the system can reach a legitimate state and provide the full (optimum) service for some time. Whereas in large scale mobile ad-hoc networks where the network topology changes very often, the paradigm of self-stabilization is no more satisfying. Indeed, as the delay between two successive disruptions is very small, the system is continuously disrupted and it may never provide its optimum service. As a consequence, the availability and reliability of self-stabilizing systems is compromised. To overcome these drawbacks, the robust self-stabilization approach has been developed [1,2].

A protocol is robust self-stabilizing if (1) it is self-stabilizing; (2) it quickly reaches a safe configuration where a minimal useful service is provided; (3) the minimal useful service holds during progress of the protocol toward the optimum service (i.e., during convergence to a legitimate configuration); (4) and it is also maintained despite multiple occurrences of some specific disruptions, called *highly tolerated disruptions*. Whatever the occurrence of highly tolerated disruptions, the useful minimal service still provided. Whereas the occurrence of other disruptions is handled by the self-stabilization mechanism, i.e., after their occurrence, the system may behave arbitrarily, but it will quickly provide the minimal useful service. Therefore, the robustness as defined in [1,2] may be seen as a service guarantee, which is provided by both: the fast recovering to a desired system characteristic (minimal useful service), and its preservation in spite of highly tolerated disruptions.

Contribution. Self-stabilizing protocols are almost evaluated only in terms of worst-case time and space complexities. In this context, theoretical studies of the robust self-stabilization approach has been done in [1,2]. However, clustering protocols presented in these two papers are written in the shared memory state model (a non realistic model). Furthermore, no experimental study has been made to compare the two approaches (i.e., self-stabilization and robust self-stabilization), and to investigate their performances in dynamic networks. In this paper, we compare the two approaches, through an experimental study. We study the behavior of four clustering protocols; two self-stabilizing protocols `GDMAC` [3] (Generalized Distributed Mobility-Adaptive Clustering) and `BSC` [4] (Bounded Size Clustering), and their robust self-stabilizing version `R-GDMAC` [1] and `R-BSC` [2]. The performances of protocols are compared in terms of their number of cluster-heads, availability of minimal and optimum services, and the stabilization time.

For our study, we use the *standard* simulation environment in the research community: Network Simulator 2 (`NS2`) [5]. Obviously to achieve this study, protocols were adapted to the message passing model.

Related Works. 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. A large number of them are self-stabilizing [6,7,8,9,10,11,12,13]. However, only [1,2] are robust self-stabilizing. A survey on clustering protocols can be found in [14].
`GDMAC` protocol is evaluated in [15] with respect to its convergence time and message complexity. Whereas, according to our knowledge, the three other protocols (`R-GDMAC`, `BSC` and `R-BSC`) have never been evaluated.

The remainder of the paper is organized as follows. In Section 2, an overview of the studied clustering protocols is given. In Section 3, the simulation model and some important remarks are discussed. The observed metrics are described in section 4, and the performance evaluation results with the analysis remarks are detailed in Section 5. Finally, we conclude our study in Section 6.

2 Overview of the studied clustering protocols

A clustering protocol consists of partitioning the network into non-overlapping groups of nodes called clusters. Each cluster has a single head (called cluster-head), and eventually a set of ordinary nodes. Each cluster-head acts as local coordinator of its cluster, and may participate to the management of the global network. So, cluster-heads have more tasks to perform than ordinary nodes. As consequence, cluster-heads must be more suitable than ordinary nodes.
Protocols `GDMAC`, `R-GDMAC`, `BSC` and `R-BSC` consider weight-based networks, i.e., a weight W_v is assigned to each node v of the network. In ad-hoc or sensor networks, amount of bandwidth, memory space, processing capacity or battery power of a node could be used to determine weight values. The choice of cluster-heads is based on the weight associated to each node: the higher the weight of a node is, the better this node is appropriate for the role of cluster-head.
The studied protocols build 1-hop clusters, where the ordinary nodes are neighbor of their cluster-head, i.e., they can directly communicate with it. Note that both `GDMAC` and `R-GDMAC` (resp. `BSC` and `R-BSC`) provide the same final clustering structure.

• `GDMAC` [3] is a self-stabilizing protocol building clusters having the following *ad-hoc clustering properties*:

- The cluster-head has the highest weight in its cluster.
- A cluster-head cannot have more than k neighbor cluster-heads.
- For every ordinary node v, there is no a v's neighbor cluster-head Y such that $W_Y > W_X + h$ where X is the current cluster-head of v. Otherwise, v changes the affiliation and it chooses Y as new cluster-head.

k and h are protocol parameters, and their value may be different from a node to another one. The parameter k allows to bound the number of cluster-heads

that can be neighbors. Whereas h is used to reduce the switching overhead of an ordinary node (i.e., the number of moves from its current cluster to a new neighbor one due to cluster-head's weight change).

• `R-GDMAC` [1] is a robust self-stabilizing version of the `GDMAC` protocol.

• `BSC` [4] is a self-stabilizing protocol building bounded size clusters. The built clusters respect the following *well balanced clustering properties*:

- The cluster-head has the highest weight in its cluster.
- The leader of a cluster is not overburden by the management workload of its cluster. Thus, a cluster can have at most $SizeBound$ ordinary nodes ($SizeBound$ is a parameter of the protocol).
- A node stays cluster-head only if it cannot join a neighbor cluster: all neighbor clusters are full. This property limits the number of cluster-heads locally. Therefore, if a leader v has a neighbor leader u such that $W_v > W_u$, then the cluster of v is full, i.e., it contains exactly $SizeBound$ members.

• `R-BSC` [2] is a robust self-stabilizing version of `BSC` protocol.

The main idea of `GDMAC` and `R-GDMAC` protocols is that an ordinary node v becomes cluster-head if in its neighborhood there is not a cluster-head having a weight greater than v's weight.
Each cluster-head should have less than k neighbor cluster-heads. Hence, if several (more than k) cluster-heads become neighbor, at least a cluster-head has to resign its status. To implement this property, each cluster-head v checks the number of its neighbors that are cluster-heads. If they exceed k, then it determines the $(k+1)^{th}$ highest weight among neighbor cluster-heads. All v's neighbor cluster-heads having a weight less than this value have to become ordinary.

Similarly, in `BSC` and `R-BSC` protocols, a node v becomes cluster-head if there is not cluster-head in v's neighborhood. Furthermore, a cluster-head v stays in this status only when it cannot join one of its neighbor clusters without violating the *well-balanced* clustering properties.
`BSC` and `R-BSC` build bounded size clusters. So, in order to prevent the violation of the size condition, a node u cannot freely join a cluster: u needs the permission of its potential new cluster-head. Therefore, each cluster-head v maintains a list of nodes who are authorized to join its cluster.

Robustness property. The robustness in `R-GDMAC` and `R-BSC` ensures that a minimal useful service is quickly provided. Once the minimal service is available, each node belongs to a cluster, and each cluster has a cluster-head. Furthermore, for `R-BSC` protocol, no cluster should have more than $SizeBound$ ordinary nodes. Preserving the minimal useful service ensures that the hierarchical structure is continuously provided throughout the network even during the its reorganization. In order to maintain the hierarchical structure over the network during reconstruction of clusters, `R-GDMAC` and `R-BSC` protocols use the following resignation process.
Resignation process. A cluster-head v that wants to become ordinary, does not take the ordinary status: v becomes a nearly ordinary node (i.e., it takes the

nearly ordinary status). In this state, v performs correctly its task of cluster-head, but no node can join v's cluster. The members of v's cluster has to quit their cluster. Moreover, v can become ordinary only once its cluster is empty. These conditions guarantee that during construction/maintenance of clusters, no cluster-head abandons its leadership.

The robustness property in clustering protocols is very useful. It ensure a high availability of hierarchical organization; and it allows the continuity functioning of upper-layer hierarchical protocols, as hierarchical routing protocols.

The set of highly tolerated disruptions handled by robust protocols are:

- the change of node's weight,
- the crash of ordinary nodes,
- the creation of new communication links without the emergence of new nodes,
- the failure of communication links between (1) two ordinary nodes, (2) two nodes behaving as cluster-heads (i.e., cluster-head or nearly ordinary node)
- the emergence of networks correctly partitionned (i.e., where the minimal service is already provided).

3 Model and simulation remarks

The simulation experiments are carried out thanks to the `NS2.34` simulator [5]. Our network is composed of mobile nodes with a propagation radio range of 250m randomly placed within a 1200m*1200m area. The density of a node (i.e., the number of neighbors per node) is at most 15. The parameters value used during simulation are presented in Figure 1.

Mobility model. Each node moves randomly according to the *Random Waypoint model* [16]. Initially, network nodes are randomly placed in the network area. At the beginning of the simulation, each node selects a random destination and moves toward it with a randomly chosen speed (uniformly distributed between 0 and `Speed` m/s). Upon reaching this destination, another random speed and destination are targeted after a pause time. The process is repeated until the simulation ends.

Parameter	**Value**
Simulation time	100s
Number of nodes	70
Transmission range	250m
Network area	1200m*1200m
Density	15
Speed	0m/s - 12m/s
Pause time	0.5s
Wmin	50
Wmax	80
Δ	2
SizeBound	10
k	2
h	3
freq	2

Fig. 1. Parameters value.

Weight variation model. The four studied protocols assume that each node has a weight, that can change during time. Initialy, each node randomly chooses its weight w between two values $Wmin$ and $Wmax$. The weight of a node changes according to a frequency $freq$, which is

the number of changes per second. For example, if $freq = 0.2c/s$, then the node's weight changes once every 5 seconds. According to the frequency value, the time when a node undergoes the weight change is chosen randomly. The new weight of a node is chosen randomly between $W - \Delta$ and $W + \Delta$.

In order to study the influence of network size, mobility of nodes, and node's weight variation, 3 different types of simulations have been conducted:

1. *Network size variation*: nodes are not mobile, and their number varies between 10 and 70. The frequency of weight variation is set to $2c/s$.
2. *Weight variation*: in order to see how protocols behave when reconstruction of clusters is high, we increase the frequency of the weight variation in a static network of 70 nodes. The values of frequency considered are: 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 1, 2, 3, 4 and 5.
3. *Mobility variation*: the speed of nodes is varied between $0m/s$ and $12m/s$ to see how protocols behave in presence of mobile nodes. The network contains 70 nodes and the weight changes twice per second.

For all protocols, identical mobility and weight variation scenarios are used in order to gather fair results. Furthermore, to get accurate results, each simulation is driven with ten different runs. The presented metrics are then averaged on these different runs. Furthermore, in order to show how these average values are confident, a confidence interval is computed using the confidence level 95%.

4 Observed metrics

To analyze the performance of clustering protocols and to compare robust self-stabilization with self-stabilization, the following metrics are studied:

• *The average number of cluster-heads*: as small as it is the number of cluster-heads, the protocol is far from being trivial; because in a trivial solution, all nodes are cluster-head. Thus, one goal of clustering protocols is to provide a hierarchical structure with a small number of cluster-heads.

• *The availability of minimum service*: it represents the percentage of time where the minimum service is available. A configuration where the minimum service is available is a configuration where the hierarchical structure is provided. More specifically, it is a configuration where:

- Each ordinary node belongs to a cluster.
- Each ordinary node is a neighbor of its leader (within its transmission range).
- Moreover, for protocols `BSC` and `R-BSC`, each cluster must have at most $SizeBound$ members.

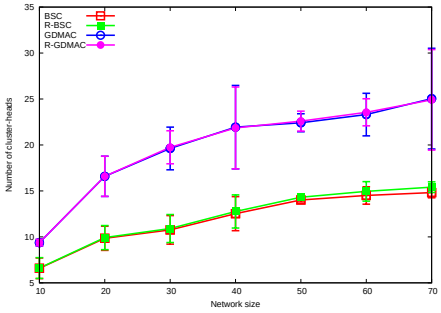
• *The availability of optimum service*: it represents the percentage of time where the optimum service is available. The optimum service is available in a configuration (called legitimate) if the built clusters verify the *ad-hoc* or *well balanced clustering properties* defined in Section 2.

As these metrics vary over time according to weight change and nodes mobility, measurements are collected every 0.02 seconds to obtain the average values.

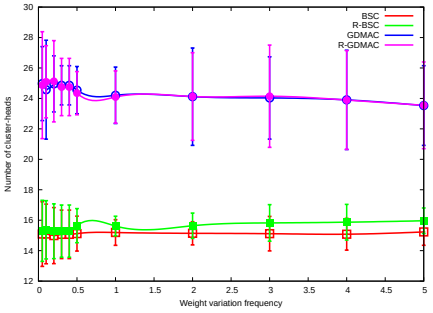
5 Simulation results and Performances analysis

5.1 Average number of cluster-heads

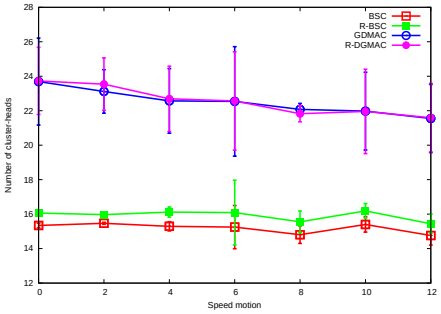
The variation of cluster-heads number in function of the network size, weight variation frequency and nodes speed are presented respectively in Figures 2(a), 2(b) and 2(c).



(a) varying the network size



(b) varying the frequency of weight changes



(c) varying the speed of nodes

Fig. 2. Average number of cluster-heads

Protocols `GDMAC` and `R-GDMAC` (resp. `BSC` and `R-BSC`) have the same behavior, because they use the same cluster-heads selection policy: weight based criteria.

For clustering protocols based on dominating sets, the number of cluster-heads is intrinsically related to the variant of dominating set computed. In fact, we distinguish a classification in two groups. Protocols `BSC` and `R-BSC` generate a smaller number of cluster-heads than protocols `GDMAC` and `R-GDMAC`. The structure used by `BSC` and `R-BSC` is the capacited dominating sets, where a cluster-head can have a neighbor cluster-head only if its cluster is full. So, two cluster-heads are rarely neighbor. While, the structure used by `GDMAC` and `R-GDMAC` is a k-fold dominating set where at most $k+1$ cluster-heads can be neighbor. If due to the mobility or weight change, $k+1$ cluster-heads become neighbor, no one needs resign its status. This feature leads to a higher total number of cluster-heads.

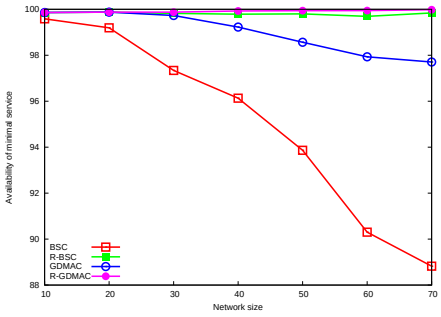
In a large scale network, a robust self-stabilizing clustering protocol generates a slightly higher number of cluster-heads than its self-stabilizing version (see BSC and R-BSC). Recall that during resignation process, robust self-stabilizing protocols use an intermediate hierarchical status, called nearly ordinary. A cluster-head wanting to resign, it takes the nearly ordinary status. A nearly ordinary node may become ordinary only once its cluster is empty; and during all this period it behaves and it is considered as a cluster-head. This is why the average number of cluster-heads is higher in a robust self-stabilizing protocol compared to its self-stabilizing version. However, the difference in cluster-heads number depends on the resignation overhead: how many cluster-heads resign their status to be ordinary?
In BSC and R-BSC protocols, as soon as two cluster-heads become neighbors, one of them must resign expect if one of clusters is full. Whereas in GDMAC and R-GDMAC protocols, if the number of neighbor cluster-heads does not exceed $k + 1$, no resignation is required. As the resignation process is more frequent in protocols BSC and R-BSC than GDMAC and R-GDMAC. Thus, the difference in cluster-heads number between R-BSC and BSC is significant, but not between GDMAC and R-GDMAC.

5.2 Availability of minimum service

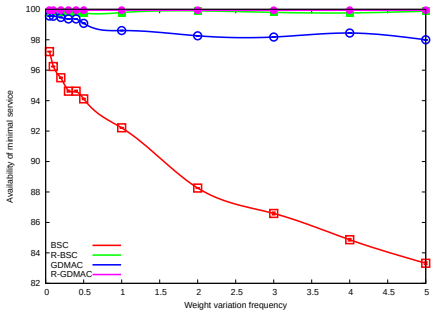
The availability of minimal service in a static network according to the network size and the frequency of weight variation are illustrated respectively in Figures 3(a) and 3(b).

Robust self-stabilizing protocols R-GDMAC and R-BSC scale well to large networks, and they are more resistant to weight change. In fact, R-GDMAC and R-BSC maintain the minimal service, once provided, during almost all their execution time whatever the network size and the weight variation frequency.
This is not the case for self-stabilizing protocols. In GDMAC protocol, the minimum service is broken by increasing the network size and the weight variation frequency. Nevertheless, the rupture rate of the minimal service stays very small. Indeed, in a network of 70 nodes where the weight changes twice per second, the rupture rate is at most 3% (thus, 97% of time, the minimal service is available). Whereas, BSC protocol has less guarantee of service than other protocols in a large scale network. Indeed, in a network of 70 nodes, the minimal service is unavailable during 12% of time. BSC is also the protocol which really suffers the most from unavailability of minimal service when the frequency of weight variation increases. Indeed, by changing the weight five time per second, the minimal service is unavailable almost 20% of time.
The poor performance of BSC protocol reflects the interest of the robustness property in large scale networks, because R-BSC protocol maintains the minimal service without any rupture whatever the network size and the weight variation frequency.

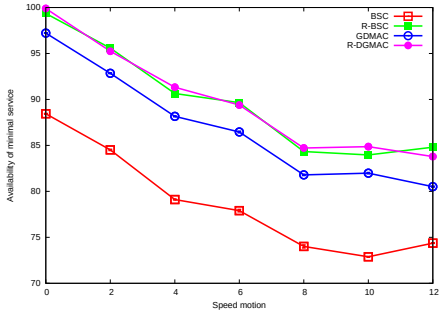
Robust self-stabilizing protocols prevent the violation of the minimal service, by using the resignation process discussed in Section 2. This mechanism guarantees



(a) varying the network size



(b) varying the frequency of weight changes



(c) varying the speed of nodes

Fig. 3. Availability of minimal service

that during construction/maintenance of clusters, no clusterhead abandons its leadership, so the minimal service is continuously provided.

The rupture of minimal service in self-stabilizing protocols happens during reconstruction of clusters due to weight change. Nevertheless, it is more frequent in `BSC` than in `GDMAC`. In `BSC`, when two cluster-heads become neighbors, in most cases one of them must defer to the other. This feature can trigger cluster-head election/resignation that may propagate throughout the network, and generates a continuous disruption of minimal service. Such an effect is called *chain reaction*. In `GDMAC`, this chain reaction effect is minimized, and the minimal service is not dramatically damaged. Furthermore, using the robust self-stabilization property improves the availability of minimal service even in the presence of chain reaction (`R-BSC`).

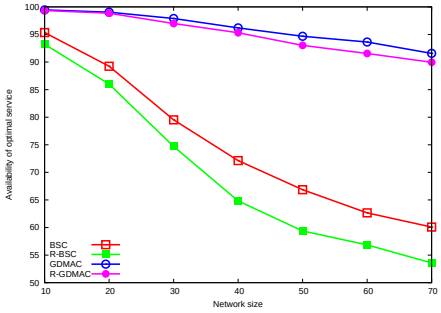
Robust protocols (`R-GDMAC` and `R-BSC`) are expected (theoretically proved in the sharded memory model) to guarantee the minimum service whatever the network size and frequency of weight change. The rupture observed (less than 0.3%) in large scale network or when the frequency is very high, is due to the following. In these protocols, when a node undergoes a change weight, it broadcasts a message to its neighbors indicating its new state (so, its new weight). When the

weight variation is very high, the number of exchanged messages is important. So, the message loss and the unordered message reception happen more frequently. Owing to these disruptions, an ordinary node can affiliate with another node (by considering it as cluster-head), but which is not a cluster-head anymore (it already resigned).

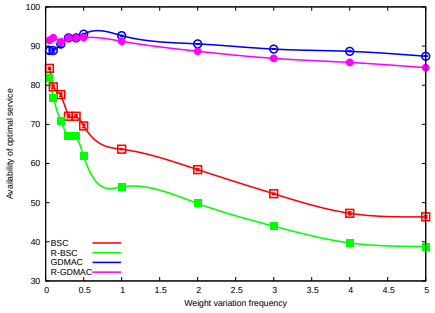
Increasing the speed of nodes has a negative impact on the availability of minimal service (see Figure 3(c)). In a dynamic network, due to nodes motion, an ordinary node and its cluster-head may be outside the transmission range of each other, i.e., they are no longer neighbors. This situation breaks the minimal service. However, even in a dynamic network, the minimal service is preserved by robust self-stabilizing protocols better than self-stabilizing ones.

5.3 Availability of optimum service

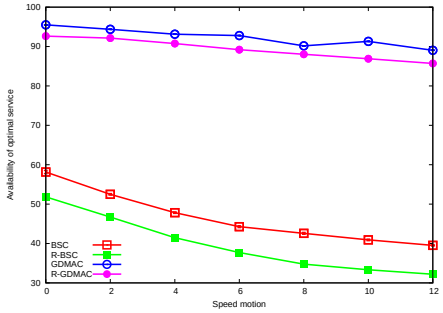
The availability of optimum service as a function of the network size, weight change frequency and nodes speed are presented in Figures 4(a), 4(b) and 4(c).



(a) varying the network size



(b) varying the frequency of weight changes



(c) varying the speed of nodes

Fig. 4. Availability of optimum service

By increasing the network size, the weight variation frequency or the speed of nodes, the optimum service is less available in `BSC` and `R-BSC` protocols than `GDMAC` and `R-GDMAC` protocols.

In `BSC` and `R-BSC` protocols, due to weight changes or nodes mobility, the hierarchical structure is continuously reconstructed in order to achieve the *well-balanced clustering properties*. As a result, the optimum service is often broken, so not highly available.
On the other hand, in `GDMAC` and `R-GDMAC` protocols, once the hierarchical structure respecting the *ad-hoc clustering properties* is built, it will rarely be modified due to the weight change. Furthermore, the mobility of nodes (especially of cluster-heads) rarely generates selection or resignation of cluster-heads. So, the optimum service is not affected.

The optimum service is slightly less available in the case of robust self-stabilizing protocols compared to their self-stabilizing versions. This is caused by the convergence (i.e., stabilization) time towards the optimum service. In fact, Figure 5 shows that the time required by a robust self-stabilizing protocol to reach the optimum service is larger than the one required by its self-stabilizing version.

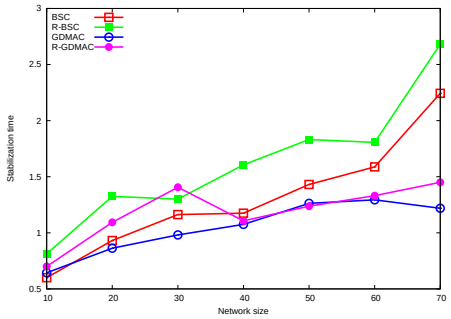


Fig. 5. Stabilization time in function of the network size

6 Concluding remarks

This article presents the first experimental study results comparing robust self-stabilization approach with self-stabilization for the clustering problem. Thanks to this study, we extract the following remarks.
The property of robustness within clustering protocols induces an increase in the average number of cluster-heads, however really negligible.
Since the robustness property consists to slow-down the convergence process, in order to maintain the minimum service. This property leads to a slight increase in the stabilization time. However, this growth in the stabilization time depends on the size of the network, but not on the nodes speed nor the frequency of weight change.
The availability of optimum service is lower in a robust self-stabilizing protocol than its self-stabilizing version. Nevertheless, the minimum service is highly available in robust self-stabilizing protocols than self-stabilizing ones. As consequence, thanks to the robustness, when the optimum service is not provided,

the minimum service is available and it will be preserved. Once the minimum service is provided, the network is completely partitionned, and each cluster has an effectual leader.
The minimum service is sufficient for the continuity of operation of upper-layer hierarchical protocols, as hierarchical routing protocols, since the hierarchical organization is available throughout the network. Therefore, robust self-stabilizing protocols are desirable, because they avoid disrupting upper-layer hierarchical protocols by maintaining the minimal service.

References

1. Johnen, C., Nguyen, L.H.: Robust self-stabilizing weight-based clustering algorithm. Theoretical Computer Science **410**(6-7) (2009) 581–594
2. Johnen, C., Mekhaldi, F.: Robust self-stabilizing construction of bounded size weight-based clusters. In: Euro-Par'10, Springer LNCS 6271. (2010) 535–546
3. Basagni, S.: Distributed and mobility-adaptive clustering for multimedia support in multi-hop wireless networks. In: VTC'99. (1999) 889–893
4. Johnen, C., Nguyen, L.H.: Self-stabilizing construction of bounded size clusters. In: ISPA'08. (2008) 43–50
5. : The Network Simulator NS-2. http://www.isi.edu/nsnam/ns/
6. 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
7. Lin, C.R., Gerla, M.: Adaptive clustering for mobile wireless networks. IEEE Journal on Selected Areas in Communications **15** (1997) 1265–1275
8. 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
9. 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
10. Drabkin, V., Friedman, R., Gradinariu, M.: Self-stabilizing wireless connected overlays. In: OPODIS'06, Springer LNCS 4305. (2006) 425–439
11. Datta, A., Devismes, S., Larmore, L.: A self-stabilizing $o(n)$-round k-clustering algorithm. In: SRDS'09. (2009)
12. Dolev, S., Tzachar, N.: Empire of colonies: Self-stabilizing and self-organizing distributed algorithm. Theoretical Computer Science **410** (2009) 514–532
13. Mitton, N., Fleury, E., Guérin-Lassous, I., Tixeuil, S.: Self-stabilization in self-organized multihop wireless networks. In: WWAN'05. (2005) 909–915
14. Abbasi, A.A., Younis, M.: A survey on clustering algorithms for wireless sensor networks. Computer Communications **30** (2007) 2826–2841
15. Bettstetter, C., Friedrich, B.: Time and message complexities of the generalized distributed mobility-adaptive clustering (GDMAC) algorithm in wireless multihop networks. In: VTC'03-Spring., IEEE (April 2003) 176–180
16. Camp, T., Boleng, J., Davies, V.: A survey of mobility models for ad hoc network research. Wireless communications & Mobile computing (WCMC): Special issue on mobile ad hoc networking: Research, trends and applications **2** (2002) 483–502