Self-stabilizing Consensus Average Algorithm in Distributed Sensor Networks

Jacques M. Bahi[1], Mohammed Haddad[2],
Mourad Hakem[1], and Hamamache Kheddouci[2]

[1] DISC Laboratory, Femto-ST - UMR CNRS, Université de Franche-Comté, France
[2] LIRIS Laboratory, UMR CNRS 5205, Université de Lyon 1, F-69622, France
{Mourad.Hakem,Jacques.Bahi}@lifc.univ-fcomte.fr,
{Mohammed.Haddad,Hamamache.Kheddouci}@univ-lyon1.fr

Abstract. One important issue in sensor networks that has received renewed interest recently is average consensus, i.e., computing the average of n sensor measurements, where nodes iteratively exchange data with their neighbors and update their own data accordingly until reaching convergence to the right parameters estimate. In this paper, we introduce an efficient self-stabilizing algorithm to achieve/ensure the convergence of node states to the average of the initial measurements of the network. We prove that the convergence of the fusion process is finite and express an upper bound of the actual number of moves/iterations required by the algorithm. This means that our algorithm is guaranteed to reach a stable situation where no load will be sent from one sensor node to another. We also prove that the load difference between any two sensor nodes in the network is within $\frac{\varepsilon}{D} \times \left\lfloor \frac{D+1}{2} \right\rfloor < \varepsilon$, where ε is the prescribed global equilibrium threshold (this threshold is given by the system) and D is the diameter of the network.

1 Introduction

Recent years have witnessed significant advances in wireless sensor networks which emerge as one of the most promising technologies for the 21st century [1]. In fact, they present huge potential in several domains ranging from health care applications to military applications. Distributed in irregular patterns across remote and often hostile environments, sensor nodes will autonomously aggregate into collaborative and asynchronous communication mode. Indeed, the asynchronous mode presents the major advantages of allowing more flexible communication schemes. They are less sensitive to the communication delays and to their variations. Moreover, they also present some tolerance to the loss of data messages since that losses do not prevent the progression of the fusion process on both the sender and destination nodes.

In general, the primary objective of a wireless sensor network is to collect data from the monitored area and to transmit it to a base station (sink) for processing. During this phase, resource failures are more likely to occur and can have an adverse effect on the application. Hence, they must be robust and survivable despite individual node and link failures [2, 3, 4, 5, 6]. The advent of wireless sensor networks and its conception constraints, have posed a number of research challenges to the networking and distributed computation communities. A problem that has received renewed interest recently is **average consensus**. It computes iteratively the global average of distributed measures in a sensor network by using only local communications. Distributed average consensus, in ad hoc networks, is an important issue in distributed agreement and synchronization problems [7] and is also a central topic for load balancing (with divisible tasks) in parallel computing [8, 9]. More recently, it has also found applications in distributed coordination of mobile autonomous agents and distributed data fusion in sensor networks [10, 11, 12, 13].

In the literature, this problem has been formulated and studied in various ways. The first approaches were based on flooding. For instance, in [14], each sensor node broadcasts all its stored and received data to its neighbors. After some times, each node will hold all the data of the network and acts as a fusion center to compute the estimate of the unknown parameter. In [15, 16, 17], the authors compute the average of the sensor measurements combined with local Kalman filtering and/or mobile agents. The works developed in [14, 18, 19] consist of distributed linear iterations, where each sensor updates its current state by a weighted fusion of its current neighbors' states (which are distorted when they reach it) and these fusion weights decrease to zero in an appropriate way, as time progresses. Other authors consider some practical issues in sensor networks such as fault tolerance and asynchronism. For instance, some works compute the average while taking into account link failures [20], other works study the consensus problem into asynchronous environment [21, 22] while considering communication delays, or from the energy point of view by minimizing the number of iterations [19].

To the best of our knowledge, none of the above approaches is able to give an analytical bound of the actual number of moves/iterations required by the algorithm, nor to improve the upper bound for the load difference (upper bounded by the diameter of the topology) between any two sensor nodes in the final load balanced distribution. In this paper, we present an efficient self-stabilizing algorithm to tackle the problem of distributed data fusion in large-scale sensor networks. This study differs from previous works for the following reasons:

- We express an upper bound of the actual number of moves/iterations required by the algorithm to ensure the convergence of node states to the average of the initial measurements of the network. More precisely, we prove that there exists an upper bound of the convergence time beyond which all the sensor nodes in the network neither receive nor send any amount of load and, therefore, achieve a stable balanced state.
- We improve the load difference between any two sensor nodes in the network which is within $\frac{\varepsilon}{D} \times \left\lfloor \frac{D+1}{2} \right\rfloor$ rather than ε, where ε is the prescribed global equilibrium threshold (this threshold is given by the system) and D is the diameter of the network.
- Unlike earlier methods, we use a new concept of Self-Stabilization to achieve the convergence of the system to a final balanced load state.

In a self-stabilizing model [23, 24, 25, 26], each vertex has only a partial view of the system, called the *local state*. The vertex's local state include the state of the vertex itself and the state of its neighborhood. The union of the local states of all the vertices gives the *global state* of the system. Based on its local state, a vertex can decide to make *a move*. Then, self-stabilizing algorithms are given as a set of rules of the form [**If** $p(i)$ **Then** M], where $p(i)$ is a predicate and M is a move. $p(i)$ is true when state of the vertex i is locally illegitimate. In this case, the vertex i is called a *privileged/active* vertex. A vertex executes the algorithm as long as it is active (at least one predicate is true).

The rest of the paper is organized as follows. After some definitions and notations in Section 2.1, we present in Sections 2 the design and analysis of the proposed self-stabilizing algorithm and give the corresponding proofs. To evaluate the behavior of the proposed algorithm, we provide in Section 3 some results through simulations that we conducted on NS2 (Network Simulator 2). Finally we give some concluding remarks in Section 4 and 5.

2 Self-stabilizing Consensus Average Algorithm

In this section, we give a self-stabilizing algorithm for computing the consensus average in a wireless sensor network under a serial, or central, scheduler. Nevertheless, there exist algorithms that make any self-stabilizing algorithm using the central scheduler operate under the distributed one [27, 28, 29, 30, 31]. We also assume a composite read/write atomicity. We begin by giving fundamentals and a description of our algorithm then we focus on the legitimate state formulation as well as the local information at the nodes. After that, we present the algorithm which consists in only one rule and give the proofs.

2.1 Fundamentals

A sensor network is modeled as a connected undirected graph $G = (V, E)$. The set of nodes is denoted by V (the set of vertices), and the links between nodes by E (the set of edges). The nodes are labeled $i = 1, 2, \ldots, n$, and a link between nodes i and j is denoted by (i, j). The set of neighbors of node i is denoted by $N_i = \{j \in V \mid (i, j) \in E\}$, and the degree (number of neighbors) of node i $\eta_i = |N_i|$. Each node takes initial measurement z_i, for the sake of simplicity, let us suppose that $z_i \in \mathbb{R}$. Then, z will refer to the vector whose ith component is z_i. Each node on the network also maintains a dynamic state $x_i(t) \in \mathbb{R}$ which is initially set to $x_i(0) = z_i$. Intuitively each node's state $x_i(t)$ is its current estimate of the average value $\sum_{i=1}^{n} z_i/n$. The goal of the averaging algorithm, is to let all the states $x_i(t)$ go to the average $\sum_{i=1}^{n} z_i/n$, as $t \to \infty$. Throughout the paper, we use the terms *scalar* and *load* interchangeably.

In our framework, instead of reaching $\sum_{i=1}^{n} z_i/n$, when $t \to \infty$, we ensure reaching $\sum_{i=1}^{n} z_i/n \pm \varepsilon$ but in a finite time. Where ε is the prescribed global equilibrium threshold.

2.2 Outline of the Algorithm

In order to reach, in a fully distributed way, the global consensus average, we draw inspiration from a natural phenomenon that fits well as a model for our problem. This phenomenon is the *communicating vessels*. In fact, one can see that by considering nodes the network as similar vessels all filled with some amount of water (the sensed value), then by making all the vessels communicating we will obtain, after stabilization, the same amount of water in all vessels. This amount is actually the global average (see Figure 1).

To model the behavior of the transfer of water from a vessel to another, we also act as in the natural phenomenon; that is the vessels with low amount of water create a depression and aspirate water from more loaded neighbors until the equilibrium is reached. Hence, there will be streams of water circulating between the vessels as a vessel could aspirate and be aspirated at the same time. In our model, we transfer an atomic quantity ϵ from a highly loaded node to a less loaded node until they reach the equilibrium. This transfer is supposed to be performed by some atomic transaction mechanism that could be called by our algorithm. Thus, the atomic transaction algorithm will be composed with our algorithm [32].

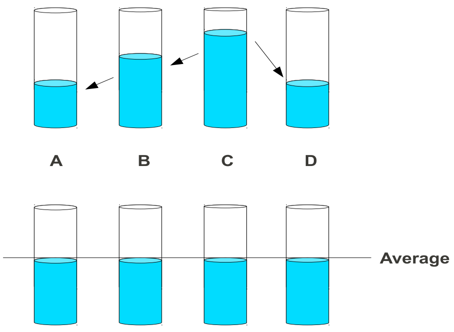

Fig. 1. Communicating vessels

2.3 Global Legitimate State

Let $G = (V, E)$ the graph modeling the sensor network. The algorithm should converge to a state where all node reach the same value representing the consensus average. However, we admit some error in the precision; that is two nodes should reach the same value according to some error ε. The legitimate state of the network is then expressed as follows:

$$\forall i, j \in V : |x_i - x_j| \leq \varepsilon \tag{1}$$

where ε is the prescribed global equilibrium threshold. This threshold is given by the system. We first prove that the Statement (1) ensures that every node in the network has reached the consensus average within a certain error e but always within the threshold ε.

Theorem 1. *Let $G = (V, E)$ be a graph such that $|V| = n$.*

$$(\forall i, j \in V : |x_i - x_j| \leq \varepsilon) \Longrightarrow (\forall i \in V : x_i = \frac{\sum_j x_j}{n} \pm e_i \wedge e_i \leq \varepsilon)$$

Proof. Since all the vertices are holding the same value according to a given error ε, we have:

$$\begin{aligned}
&(\forall i, j \in V : |x_i - x_j| \leq \varepsilon)\\
\Leftrightarrow\ &(\forall i, j \in V : -\varepsilon \leq x_i - x_j \leq \varepsilon)\\
\Leftrightarrow\ &(\forall i, j \in V : x_j - \varepsilon \leq x_i \leq x_j + \varepsilon)\\
\Leftrightarrow\ &(\forall i, j \in V : x_i = x_j \pm e_i \wedge e_i \leq \varepsilon)\\
\Rightarrow\ &(\forall i \in V : \textstyle\sum_j x_i = \sum_j x_j \pm \sum_j e_i \wedge e_i \leq \varepsilon)\\
\Rightarrow\ &(\forall i \in V : n \times x_i = \textstyle\sum_j x_j \pm n \times e_i \wedge e_i \leq \varepsilon)\\
\Rightarrow\ &(\forall i \in V : x_i = \frac{\sum_j x_j}{n} \pm e_i \wedge e_i \leq \varepsilon)
\end{aligned}$$ □

2.4 Local Information

Every node i in the network has to maintain the following data structure:

- x_i: the scalar value at node i.
- N_i: the set of neighbors of node i.
- σ: the local equilibrium threshold.

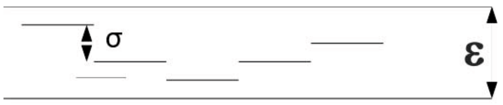

Fig. 2. The threshold σ

The threshold σ has to be chosen such that the transitive difference between nodes will never exceed the real threshold ε (see Figure 2). In fact, let's suppose three vertices a, b and c such that a is a neighbor of b which also a neighbor of c but a an c aren't neighbors. If the difference between the values x_a and x_b is less than σ and the difference between the values x_b and x_c is less than σ then what could we say about the difference between the values x_a and x_c ? Hence, the threshold σ is defined according to the diameter of the network D. Actually, by setting $\sigma \leq \varepsilon/D$, we obtain a sufficient condition on vertices to ensure the global threshold. The deployment knowledge of sensor networks is often used to get better performance. Indeed, in [33] deployment knowledge like the number of nodes and the diameter of the network is addressed.

2.5 The Algorithm

As mentioned above, the algorithm consists in only on rule that

2.1. The rule R_1 : Local equilibrium

```
$R_1$: Transfers $\sigma$ from a neighbor $j$ to $i$ if $j$ is more loaded than $i$.
If $\exists j \in N_i : x_j - x_i > \sigma$ Then
    $Transfer(x_j, x_i)$
End If
```

With

2.2. Transfer Transaction Procedure

```
$Transfer(x_j, x_i)$
$x_j = x_j - \sigma$
$x_i = x_i + \sigma$
```

2.6 Convergence Proof

Let $G = (V, E)$ the graph modeling the sensor network, with $|V| = n$ and $|E| = m$. In the following, we consider a discrete time where every move increments the time t by 1. Let $Max(t)$ be the maximum value in the network at the time t and respectively $Min(t)$ be the minimum value.

Lemma 1. $\forall t, Max(t) \geq Max(t+1)$ *(respectively,* $\forall t, Min(t) \leq Min(t+1)$*).*

Proof. the proof is straightforward since we transfer an atomic quantity σ from a highly loaded node to a less loaded node. □

Lemma 2. *If the system is unstable, that is the Statement (1) is false, then we have* $Max(t) < Max(t + \Delta t)$ *such that* Δt *is within* $O(n)$ *moves.*

Proof. The worst case is when only one vertex is not in the equilibrium (consider it to be the black vertex in Figure 3). Since all other vertices are in equilibrium, all of them are holding the maximum value. Hence, in the worst case, the transfer stream will be formed by all the vertices in the network as a Hamiltonian path. This produces that the Max (rsp. Min) value will be decremented (rsp. incremented) by at least σ within $O(n)$ moves. □

Theorem 2. *The algorithm described by the rule* R_1 *converges within* $O\left(\frac{Max(0) - Min(0)}{\sigma} \times n\right)$ *moves.*

Proof. By the previous lemmas, we have seen that Max value is decremented by at least σ within $O(n)$ moves (rsp. for Min). The worst case here is when the average is close to one of the extremal values either $Max(0)$ or $Min(0)$. Hence, $O\left(\frac{Max(0) - Min(0)}{\sigma}\right)$ transfers will be needed to reach the average. Since every transfer could cost $O(n)$ moves, we obtain that the algorithm converges within $O\left(\frac{Max(0) - Min(0)}{\sigma} \times n\right)$. □

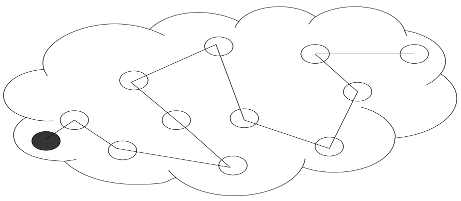

Fig. 3. Worst case of load transfer

2.7 Improvement of the Algorithm

We propose now to introduce a new rule to the algorithm. This rule aims to improve the global equilibrium of the network while proceeding only on local information. The rule is as follows:

2.3. Neighborhood Equilibrium

```
$R_2$: Transfers $\sigma$ from a neighbor $k$ to a neighbor $l$ if $k$ is more
loaded than $l$.
If $(\forall j \in N_i : |x_j - x_i| \leq \sigma) \wedge (\exists k, l \in N_i : x_k - x_l > \sigma)$ Then
  $Transfer(x_k, x_l)$
End If
```

With

2.4. Transfer procedure

```
$Transfer(x_k, x_l)$
$x_k = x_k - \sigma$
$x_l = x_l + \sigma$
```

Observe however that this second transfer procedure will require distance two knowledge.

Theorem 3. *The load difference between any two sensor nodes in the network is within* $\frac{\varepsilon}{D} \times \left\lfloor \frac{D+1}{2} \right\rfloor < \varepsilon$, *where* ε *is the prescribed global equilibrium threshold (this threshold is given by the system) and* D *is the diameter of the network.*

Proof. To prove this bound, consider a linear chain graph of n nodes ($D = n - 1$) arranged with ascending order of their loads x_i, $1 \leq i \leq n$ along a line. If all nodes are in the following state:

$$\begin{gathered} x_1 = \tfrac{\varepsilon}{n-1} < \; x_2 = 2 \times \tfrac{\varepsilon}{n-1} \\ < \cdots < \\ x_{i-1} = (i-1) \times \tfrac{\varepsilon}{n-1} < \; x_i = i \times \tfrac{\varepsilon}{n-1} \\ < \cdots < \\ x_{n-1} = \varepsilon < \; x_n = n \times \tfrac{\varepsilon}{n-1} \end{gathered}$$

Then, for this configuration, the only rule that can be executed is rule 2. For the sake of simplicity, assume that nodes, with index $i \mod 2 = 0$, will be activated for rule 2. Thus, we get

$$\begin{gathered} x_1 = x_2 = x_3 = 2 \times \tfrac{\varepsilon}{n-1} \\ < \cdots < \\ x_{i-1} = x_i = x_{i+1} = i \times \tfrac{\varepsilon}{n-1} \\ < \cdots < \\ x_{n-2} = x_{n-1} = x_n = \varepsilon \end{gathered}$$

Similarly, for this configuration, the only rule that can be executed is rule 1 and for the sake of simplicity, assume that the involved nodes $(3, 6, \ldots n-3)$ will be activated for rule 1. Thus, we get:

$$\begin{gathered} x_1 = x_2 = 2 \times \tfrac{\varepsilon}{n-1} < x_3 = x_4 = 3 \times \tfrac{\varepsilon}{n-1} \\ < \cdots < \\ x_{i-1} = x_i = i \times \tfrac{\varepsilon}{n-1} < x_{i+1} = x_{i+2} = (i+1) \times \tfrac{\varepsilon}{n-1} \\ < \cdots < \\ x_{n-3} = x_{n-2} = \varepsilon - \tfrac{\varepsilon}{n-1} < x_{n-1} = x_n = \varepsilon \end{gathered}$$

Now, by alternating the execution of the two rules, there will be streams of load circulating between the nodes as a node could aspirate and be aspirated at the same time. Hence, this process is repeated until reaching the configuration case where nodes on the line are in the ascending order by an increment of $\frac{\varepsilon}{n-1}$ with at least two adjacent nodes which have the same load value. Formally:

$$\forall i, \forall k, l \in N_i,\ k \notin N_l : x_k - x_l \leq \sigma$$

In this case, all nodes will never again execute rule 2. This means, that all nodes reach their stable state where no load will be sent from one sensor node to another.

This configuration case can be viewed as a splitting of the initial linear chain graph into a new linear chain of virtual nodes, where each virtual node contains at least two nodes with the same load value. The virtual nodes along a new chain graph are in the ascending order by an increment of $\frac{\varepsilon}{n-1}$.

Thus, it follows that for $n \geq 2$ the load difference between any two sensor nodes in the linear chain graph is within

$$\tfrac{\varepsilon}{n-1} \times \left\lfloor \tfrac{n}{2} \right\rfloor = \tfrac{\varepsilon}{D} \times \left\lfloor \tfrac{D+1}{2} \right\rfloor < \varepsilon$$

$\square$

Theorem 4. *The bound* $\frac{\varepsilon}{D} \times \left\lfloor \frac{D+1}{2} \right\rfloor$ *is attainable.*

Proof. To see that this bound is really attainable, consider a linear chain graph of $n = 6$, a non negative integer $\varepsilon = 5$ and

$$\sigma = \tfrac{\varepsilon}{D} = \tfrac{5}{5} = 1 = |x_{i+1} - x_i|,\ 1 \leq i \leq n-1$$

By alternating the execution of the two rules, we obtain the final stable situation of loads

$$x_1 = 2 < x_2 = x_3 = 3 < x_4 = x_5 = 4 < x_6 = 5$$

with the difference of

$$\frac{\varepsilon}{n-1} \times \left\lfloor \frac{n}{2} \right\rfloor = \frac{\varepsilon}{D} \times \left\lfloor \frac{D+1}{2} \right\rfloor = \frac{5}{5} \times \left\lfloor \frac{5+1}{2} \right\rfloor = 3 < \varepsilon = 5$$

□

Theorem 5. *For a non negative integer load balancing problem, the load difference between any two sensor nodes in the network is within* $\left\lfloor \frac{D+1}{2} \right\rfloor$, *where* D *is the diameter of the network.*

Proof. the proof is straightforward since the prescribed global equilibrium threshold ε is bounded by the diameter of the network D. □

We discuss the performance of introducing this rule in the next section.

3 Experimentation

In this section, we discuss some results through simulations that we conducted on NS2 (Network Simulator 2). We considered different sizes for the sensor network: 50, 100, 200, 400, 800 and 1600 nodes with an average density of 100 nodes per km^2. The radio transmission range is assumed to be 250 m. The threshold σ is set to 0.1 while $Max(0) - Min(0)$ is set to 10. The scalars of nodes and nodes positions are determined according to uniform distribution. We ran the two versions of our algorithm. The first version executes only the rule R_1 and the second executes both rules R_1 and R_2. For every size of the network, we consider 10 executions of the algorithm then we calculate the average of obtained results.

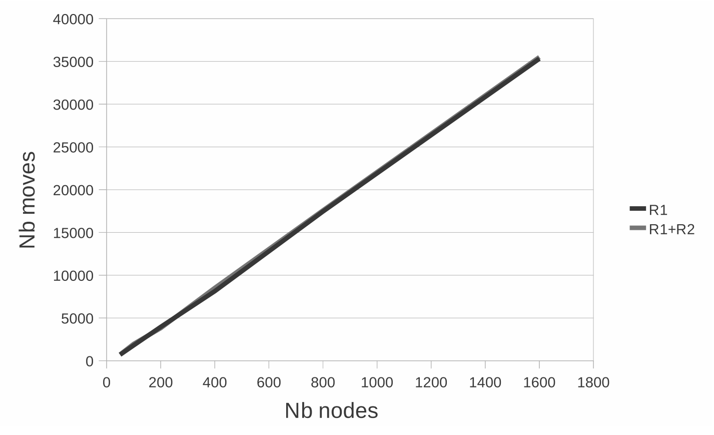

Fig. 4. Convergence time

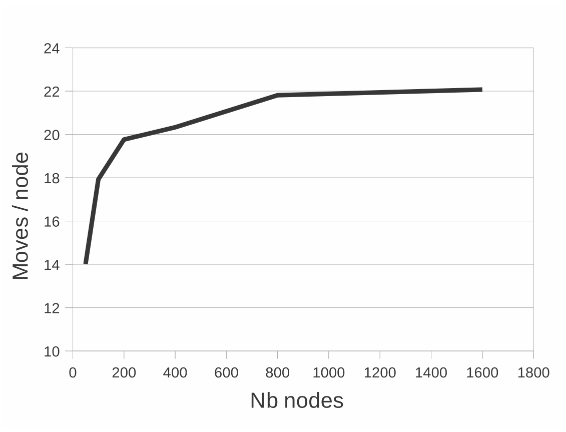

Fig. 5. Number of moves per node

We first discuss the convergence time. In our study we express this time by the number of moves performed by the set of all nodes of the network.

Consider the Figure 4. We can observe that both versions of the algorithms converge within similar amount of moves. We also observe that the number of moves is increasing linearly with the number of nodes in the network n. In fact, the slope of the line is about 22 while the one determined by the upper bound of the convergence time is exactly $(Max(0) - Min(0))\,/\sigma = 100$. Actually, the equation of the obtained line is $y = 22.33\,x - 415$. Hence, we can expect that for large values of n, the number of moves performed by a node will be about 22 moves. This is confirmed by Figure 5.

This figure gives the number of moves performed by a node according to the total number of nodes in the network. The observed value increases in a logarithmic way until reaching the value of ~ 22.

Now, in order to show the performance of the introduction of the rule R_2, we consider the number of nodes that converge outside the interval $\overline{x} \pm \sigma$ where $\overline{x}$ is the global consensus average. Before giving interest to that number of nodes, we first discuss the ratio between σ and ε. In all our simulations, we observed that for every node i, after the convergence, the value $|x_i - \overline{x}|$ is always less than $3 \times \epsilon$. Hence, if we suppose a uniform distribution of sensed values or scalars, one might have no need of a prior knowledge or estimation of the diameter of the network to set σ according to a precision ε.

The Figure 6 gives the number of nodes that converge outside the interval $\overline{x} \pm \sigma$ (but still all the nodes converge within the interval $\overline{x} \pm \varepsilon$).

We can observe that for networks with small number of nodes, the introduction of the rule R_2 has not much effect. However, with the increase of the node number, the difference between the two versions of the algorithm become more important. Moreover, after the convergence, the value $|x_i - \overline{x}|$ is always less than $2 \times \epsilon$ if we consider the algorithm using both rules R_1 and R_2.

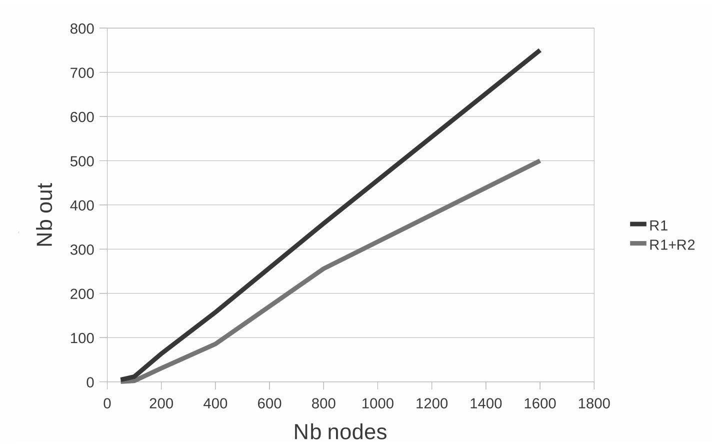

Fig. 6. Number of node out of $\overline{x} \pm \sigma$

4 Discussion and Future Work

We present in this section some generalizations of our algorithms. For the sake of simplicity, the discussion is given from the point of view of node i.

i) Improving Reliability: unexpected node failures may occur during the fusion process due to various reasons such as battery depletion/ exhaustion, software glitches, dislocation or environmental hazards and malicious attacks. To cope with this problem, when the area of interest has a significant density of sensors, we can perform redundancy/replication mechanisms, where some sensors can be in an active state: they participate in the network functioning while the others in a passive state (standby). These sensors wakeup periodically. If a working sensor node fails, it must be replaced by a passive one. However, two questions arise here: i) How the fault detection is done? and ii) how to replace the failed sensors?

These questions (i) and (ii) raise the following problems: (1) Since sensor nodes are not aware of their neighbors, especially the number of sleeping/passive nodes. How to adjust the wakeup period of these sensors? (2) During the recovery process, how to handle the case where two or more sleeping nodes, would realize at the same time that the working/active node is down? Indeed, for the same covered area, it should not contain several working nodes simultaneously, which would distort the computation of the average consensus, the *self stabilizing* algorithm should be built on the fact that only one sensor node must be in the *active* state for each covered area. To compute node's sleeping wakeup rate, we can borrow the same principle to [5]. Intuitively, nodes are initially in the sleeping mode. Each node sleeps for an exponentially distributed time generated according to a probability density function (PDF) $f(t) = \lambda e^{-\lambda t}$, where λ is the probing rate of the sensor node and t denotes its sleeping time duration.

ii) Distributed Termination: the detection of the conjunction of local terminations, which is a stable property, is a non-trivial problem. In fact it covers two issues: (i) detect whether all sensor node states converge to the average of the initial measurements of the network even when sensor nodes are subject to failures and (ii) ensure that we have achieved the desired computations. Solving this problem in a distributed manner, allows each sensor to detect that it has done and all the nodes reach this coherent state. Thus, the objective here is to overlay the *self stabilizing* iterative fusion process, a control mechanism that can detect the conditions of termination/convergence.

5 Conclusion

In this paper, we have addressed the problem of distributed data fusion in wireless sensor networks. This is a very natural and important problem, as several objectives (convergence, performance) must be considered simultaneously to fulfill the requirements of the user application. To the best of our knowledge, the proposed algorithm is the first to address the upper bound of the number of moves/iterations required to achieve/ensure the convergence of node states to the average of the initial measurements of the network. In addition, we also showed that the load difference between any two sensor nodes in the network is within $\frac{\varepsilon}{D} \times \left\lfloor \frac{D+1}{2} \right\rfloor < \varepsilon$, where ε is the prescribed global equilibrium threshold (this threshold is given by the system) and D is the diameter of the network.

Our approach should be extended to the context of safety critical applications. For instance, security threats must be addressed during the self-stabilizing fusion process. Most current approaches do not consider/include security measures, which opens an opportunity for further research in this field.

Acknowledgments. We thank the referees for all the valuable comments that helped us to improve the paper.

References

[1] Akyildiz, I., Su, W., Sankarasubramniam, Y., Cayirci, E.: A survey on sensor networks. IEEE Communications Magazine, 102–114 (2002)
[2] Paradis, L., Han, Q.: A survey of fault management in wireless sensor networks. JNSM 15(2), 171–190 (2007)
[3] Hai, L., Amiya, N., Ivan, S.: Fault-tolerant algorithms/protocols in wireless sensor networks. In: Handbook of Wireless Ad Hoc and Sensor Net., pp. 265–295 (2009)
[4] Saleh, I., Eltoweissy, M., Agbaria, A., El-Sayed, H.: A fault tolerance management framework for wireless sensor networks. JCM 2(4), 38–48 (2007)
[5] Ye, F., Zhang, H., Lu, S., Zhang, L., Hou, J.C.: A randomized energy-conservation protocol for resilient sensor networks. Wireless Networks 12(5), 637–652 (2006)
[6] de Souza, L.M.S., Vogt, H., Beigel, M.: A survey on fault tolerance in wireless sensor networks. Sap research, braunschweig, germany
[7] Lynch, N.: Distributed algorithms. Morgan Kaufmann Publishers, Inc. (1996)
[8] Cedo, F., Cortés, A., Ripoll, A., Senar, M.A., Luque, E.: The convergence of realistic distributed load-balancing algorithms. Theory Comput. Syst. 41(4), 609–618 (2007)
[9] Rabani, Y., Sinclair, A., Wanka, R.: Local divergence of markov chains and the analysis of iterative load-balancing schemes. In: Proceedings of the IEEE Symp. on Found. of Comp. Sci., Palo Alto (1998)

[10] Bahi, J., Couturier, R., Vernier, F.: Synchronous distributed load balancing on dynamic networks. Journal of Parallel and Distributed Computing 65(11), 1397–1405 (2005)

[11] Olfati-Saber, R., Murray, R.M.: Consensus problems in networks of agents with switching topology and time-delays. IEEE Transaction on Automatic Control 49(9), 1520–1533

[12] Bliman, P., Ferrari-Trecate, G.: Average consensus problems in networks of agents with delayed communications. Journal of IFAC 44(8), 1985–1995 (2008)

[13] Moallemi, C.C., Roy, B.V.: Consensus propagation. IEEE Trans. Inf. Theory 52(11), 4753–4766 (2006)

[14] Legg, J.A.: Tracking and sensor fusion issues in the tactical land environement. Technical Report TN.0605 (2005)

[15] Olfati-Saber, R., Shamma, J.S.: Consensus filters for sensor networks and distributed sensor fusion. In: 44th IEEE Conf. on Dec. and Cont. CDC-ECC (2005)

[16] Olfati-Saber, R.: Distributed kalman filter with embeded consensus filters. In: 44th IEEE Conf. on Dec. and Cont. (2005)

[17] Olfati-Saber, R., Fax, J., Murray, R.: Consensus and cooperation in networked multi-agent systems. In: Proc. of IEEE, pp. 215–233 (2007)

[18] Xiao, L., Boyd, S., Lall, S.: A space-time diffusion scheme for peer-to-peer least-squares estimation. In: Proc. of Fifth International Conf. on Information Processing in Sensor Networks (IPSN 2006), pp. 168–176 (2006)

[19] Talebi, M.S., Kefayati, M., Khalaj, B.H., Rabiee, H.R.: Adaptive consensus averaging for information fusion over sensor networks. In: IEEE International Conference on Mobile Adhoc and Sensor Systems (MASS), pp. 562–565 (2006)

[20] Kar, S., Moura, J.M.F.: Distributed consensus algorithms in sensor networks with imperfect communication: link failures and channel noise. IEEE Transactions on Signal Processing 57(1), 355–369 (2009)

[21] Bahi, J.M., Giersch, A., Makhoul, A.: A scalable fault tolerant diffusion scheme for data fusion in sensor networks. In: InfoScale 2008, pp. 1–5. ICST press (2008)

[22] Bertsekas, D.P., Tsitsiklis, J.N.: Parallel and Distributed Computation: Numerical Methods. Athena Scientific (1997)

[23] Gupta, S.K.S., Srimani, P.K.: Self-stabilizing multicast protocols for ad hoc networks. Journal of Parallel and Distributed Computing 63(1), 87–96 (2003); Wireless and Mobile Ad Hoc Networking and Computing

[24] Beauquier, J., Clement, J., Messika, S., Rosaz, L., Rozoy, B.: Self-stabilizing counting in mobile sensor networks. In: PODC 2007: Proceedings of the Twenty-Sixth Annual ACM Symposium on Principles of Distributed Computing, pp. 396–397. ACM, New York (2007)

[25] Hoepman, J.-H., Larsson, A., Schiller, E.M., Tsigas, P.: Secure and self-stabilizing clock synchronization in sensor networks. In: Masuzawa, T., Tixeuil, S. (eds.) SSS 2007. LNCS, vol. 4838, pp. 340–356. Springer, Heidelberg (2007)

[26] Dijkstra, E.W.: Self-stabilizing systems in spite of distributed control. Commun. ACM 17(11), 643–644 (1974)

[27] Gradinariu, M., Tixeuil, S.: Conflict managers for self-stabilization without fairness assumption. In: International Conference on Distributed Computing Systems, p. 46 (2007)

[28] Goddard, W., Hedetniemi, S.T., Jacobs, D.P., Srimani, P.K.: Self-stabilizing protocols for maximal matching and maximal independent sets for ad hoc networks. In: Proceedings of the 17th International Symposium on Parallel and Distributed Processing, IPDPS 2003, pp. 162.2. IEEE Computer Society, Washington, DC (2003)

[29] Goddard, W., Hedetniemi, S.T., Jacobs, D.P., Trevisan, V.: Distance- k knowledge in self-stabilizing algorithms. Theoretical Computer Science 399(1-2), 118–127 (2008); Flocchini, P., Gąsieniec, L. (eds.): SIROCCO 2006. LNCS, vol. 4056. Springer, Heidelberg (2006)

[30] Beauquier, J., Datta, A.K., Gradinariu, M., Magniette, F.: Self-stabilizing local mutual exclusion and daemon refinement. In: Herlihy, M.P. (ed.) DISC 2000. LNCS, vol. 1914, pp. 223–237. Springer, Heidelberg (2000)
[31] Afek, Y., Dolev, S.: Local stabilizer. Journal of Parallel and Distributed Computing 62(5), 745–765 (2002)
[32] Leal, W., Arora, A.: Scalable self-stabilization via composition. In: Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS 2004), pp. 12–21. IEEE Computer Society, Washington, DC (2004)
[33] Jaworski, J., Ren, M., Rybarczyk, K.: Random key predistribution for wireless sensor networks using deployment knowledge. Computing 85(1-2) (2009)