A Scalable Fault Tolerant Diffusion Scheme for Data Fusion in Sensor Networks

Jacques M. Bahi, Arnaud Giersch and Abdallah Makhoul
Computer Science Laboratory, University of Franche-Comté (LIFC)
Rue Engel-Gros, BP 527, 90016 Belfort Cedex, France
{jacques.bahi, arnaud.giersch, abdallah.makhoul}@univ-fcomte.fr

ABSTRACT

In sensor networks, sensor nodes are usually deployed randomly over an area to collect the information of interest. Data fusion is the phase of processing the collected information by sensor nodes before they are sent to the end user. This paper introduces a distributed consensus algorithm that allows the nodes of a sensor network to track the average of n sensor measurements. The study of the above mentioned algorithm showed that it is robust to asynchronism and dynamic topology changes. It is also put in evidence that the algorithm is fully distributed and does not require any global coordination. Moreover, The proposed method doesn't involve explicit point-to-point message or routing, it diffuses information across the network. Accordingly, simulation results are provided illustrating the effectiveness of the studied algorithm.

1. INTRODUCTION

Sensor networks have been considered as one of the most important technologies for the 21st century, due to their tight integration with the physical world. The main task of a sensor node is to detect events and collect data from the region of interest. The phase of processing the collected data before sending it to the final user is known as "data fusion", or “sensor fusion”. It is defined as the process of gathering data from multiple and various sources. The data fusion in sensor networks is quiet different than other networks, due to their particularity and originality in the sense that, they are autonomous, have a limited capability and are infrastructure less. Due to the above reasons, data fusion over sensor networks deals with issues as developing distributed, asynchronous and non-complex algorithms and methods to integrate collected data.

A fundamental problem in sensor networks is to solve detection and estimation problems using distributed scalable algorithms. In this paper, a specific method of sensor fusion is utilized for the estimation of the average of the collected data which is identified throughout the paper as the unknown parameter estimation. We assume that each sensor takes a measurement of the unknown parameter. The aim is to then find the average of the physical measurements over the whole network. This problem is also known as the average consensus problem [11, 5].

In order to compute the average consensus, centralized and distributed fusion schemes are distinguished. In the centralized fusion schemes, each sensor collects the information and sends it by multi-hop communications or by direct transmission to a central base station which extracts the estimation of the unknown parameter. This scenario suffers from significant setbacks such as routing and dynamic topology changes. In addition, direct transmission has also proven to be very ineffective. For example, some sensor nodes may be very far away from the base station, thus the amount of energy consumed can be extremely high. In contrast to the centralized schemes, in distributed data fusion methods, the sensor nodes only communicate their collected data to their neighbors, which eliminates the need for a computing center. In such schemes, each sensor node updates its data to get a local estimation of the average in the entire network.

Many schemes for distributed data fusion in sensor networks have been proposed. The first and the simplest method is flooding. Via the method of flooding, each sensor node broadcasts all its stored and received data to its neighbors. As a consequence, each node has all the data of the network and acts as a fusion center to compute the average consensus. Nevertheless, the above technique has several disadvantages [11]. By flooding, duplicate messages can be sent to the same node, or one node can receive the same message from several neighbors. In addition, this proposed technique requires a significant amount of transmissions and a large storage memory, which wastes the network's resources and which considerably decreases the lifetime of the network.

An iterative method for distributed data fusion in sensor networks based on the calculation of an average consensus has been proposed in [11]. The authors consider a sensor network of n sensor nodes, where every node takes a noisy measurement of the unknown parameter. Each node broadcasts its data to its neighbors and updates its estimation according to a weighted sum of the received data.

Alternatively, in [5] the authors proposed a scalable sensor fusion scenario that performs fusion of sensor measurements combined with local Kalman filtering. They developed a distributed algorithm that allows the sensor nodes to compute the average of all of their measurements. It is worthy to note that many other sensor fusion and average consensus approaches are based on Kalman filters and mobile agents [9, 4, 8, 7].

Although the above mentioned works and other existing data fusion scenarios guarantee some level of robustness to node failure and dynamic topology changes [11, 5, 10], they do not address certain practical issues such as fault tolerance and asynchronism. As sensor nodes are driven by batteries and have a small and finite source of energy, they are prone to failure. Moreover, due to external influences, the reachability of nodes is not always guaranteed, which leads to dynamic topology changes. Furthermore, naturally and due to wireless channels the messages are prone to loss. In synchronous methods, it is assumed that all nodes perform operations in a step order, which is impractical and undesirable in real sensor networks. Hence, one of the most challenging issues in sensor networks is to provide a data fusion method that does not require any node synchronization.

To the best of our knowledge, the above aspects which are extremely important, are not taken into account in previous data fusion scenarios. In this work, a fault tolerant and asynchronous data fusion scheme in sensor networks will be presented. It focuses on a distributed iterative procedure for calculating averages over asynchronous sensor networks. The sensor nodes exchange and update their data by the mean of a weighted sum in order to achieve the average consensus. The suggested algorithm does not rely on synchronization between the nodes nor does it require any knowledge of the global topology. To round up, the convergence of the proposed algorithm is proved in a general asynchronous environment.

The outline of the paper is as follows. In the next section we introduce the problem of average consensus. In section 3, we present our work, the general algorithm as well as some practical aspects. Section 4 is devoted to the experimental results. Finally we conclude the paper and we briefly discuss future works.

2. AVERAGING IN SENSOR NETWORK

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 $i = 1, 2, \dots, n$, and a link between nodes i and j is denoted by (i, j). The dynamic topology changes are represented by the time varying graph $G(t) = (V, E(t))$, where $E(t)$ is the set of active edges at time t and V is the set of nodes which stays invariable. The set of neighbors of node i at time t is denoted by $N_i(t) = \{j \in V \mid (i, j) \in E(t)\}$, and the degree (number of neighbors) of node i at time t by $\eta_i(t) = |N_i(t)|$.

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 $t \to \infty$.

In the work presented in [11], each node exchanges its instantaneous data with its neighbors and updates its state following a linear discrete time method as follows

$$x_i(t+1) = x_i(t) - \sum_{j \in N_i} \alpha_{ij}(t)(x_i(t) - x_j(t)), i = 1, \dots, n. \quad (1)$$

Here $\alpha_{ij}(t)$ is the weight on $x_j(t)$ at node i, and $\alpha_{ij}(t) = 0$ for $j \notin N_i(t)$.

This method is a specific method and it does not consider many important practical issues such as transmission errors and delays. it is however, robust to the dynamic topology changes, in other words to the link failures. Hence, the results presented in equation 1 implicitly assume synchronization. Knowing that real sensor networks constitute an inherently asynchronous environment with dynamic network delays, synchronization is impractical and undesirable. To avoid those problems, we will present in the next section our approach which is more general and takes into account delays and messages reliability.

3. ASYNCHRONOUS FUSION SCHEME

This section is devoted to the main algorithm of the averaging data scheme. First we introduce the general form of the algorithm, then some practical aspects are developed.

3.1 General Algorithm

The proposed algorithm to compute the average consensus over the network is based on information diffusion. As shown in the previous section, each node takes a measurement and then cooperates with its neighbors in a diffusion manner to estimate the average of all the collected information. To overcome the synchronization between the nodes, the algorithm is able to deal with communication delays and with messages loss. It is inspired from the load balancing algorithm presented by Bertsekas and Tsitsiklis in [3, section 7.4]. In the load-balancing field, the idea is to balance some "load" in order to fully use the processor resources. By applying the same kind of algorithm, and substituting some scalar to the load, we obtain an algorithm that leads to computing the average of the initial values. In this work, the algorithm of Bertsekas and Tsitsiklis is however extended to be able to deal with dynamic topology changes, and messages loss.

These are some common notations and assumptions required by our algorithm. As our method takes into account delays and does not rely to any node synchronization, we consider that at time t a node i gets the state of its neighbor j at time $d_j^i(t)$, where $0 \leq d_j^i(t) \leq t$, $d_j^i(t)$ represents the transmission delay between nodes i and j. Therefore, let us denote $x_j^i(t) = x_j(d_j^i(t)) \in \mathbb{R}$ the state of node j at time $d_j^i(t)$, received at time t by node i. Then, we defined the extended neighborhood of node i at time t as the set

$$\overline{N}_i(t) = \left\{ j \mid \exists\, d_j^i(t) \in \{t - B + 1, ..., t\} \text{, such that } j \in N_i(d_j^i(t)) \right\};$$

note that $N_i(t) \subset \overline{N}_i(t)$.

Finally, $s_{ij}(t)$ is the amount of the data sent by node i to node j at time t, and $r_{ji}(t) = s_{ji}((d_j^i(t))$ is the amount of data received by node i from node j at time t.

We now introduce three assumptions that ensures the convergence of our algorithm.

ASSUMPTION 1. *There exists* $B \in \mathbb{N}$ *such that* $\forall t \geqslant 0$, $t - B < d_j^i(t) \leq t$ *and the union of communication graphs* $\bigcup_{\tau=t}^{t+B-1} G(\tau)$ *is a connected graph.*

This assumption known as jointly connected condition [11, 1] implies that each node i is connected to a node j within any time interval of length B and that the delay between two nodes cannot exceed B.

ASSUMPTION 2. *There exists* $\alpha > 0, \forall t \geqslant 0$, $\forall i \in N, \forall j \in N_i(t)$, *such that* $\alpha(x_i(t) - x_j^i(t)) \leq s_{ij}(t)$. $\big(s_{ij}(t) = 0$ *if* $(x_i(t) \leq x_j^i(t))$ *for all* $j \in N_i(t)\big)$.

ASSUMPTION 3.

$$x_i(t) - \sum_{k \in N_i(t)} s_{ik}(t) \geq x_j^i(t) + s_{ij}(t) \quad (2)$$

The two last assumptions are given to manage the choice of $s_{ij}(t)$. The first one postulates when a node i detects a difference between its state and the states of its neighbors. Therefore, it computes non negligible s_{ij} to all nodes j where $(x_i(t) > x_j^i(t))$. The assumption 3 prohibits node i to compute very large s_{ij} which creates a ping-pong state. Recall that, the ping-pong state is established when two nodes keep sending data to each other back and forth, without ever reaching equilibrium. These assumptions are similar to assumption 4.2 introduced in [3, section 7.4].

The main steps of our algorithm are presented by Algorithm 1.

Algorithm 1 The General Algorithm.

1: Each node maintains an instantaneous state $x_i(t) \in \mathbb{R}$, and at $t = 0$ (after all nodes have taken the measurement), each node initializes its state as $x_i(0) = z_i$.

2: At each step t each node i:

- compares its state to the states of its neighbors;
- chooses and computes $s_{ij}(t)$. They have to be chosen carefully in order to ensure the convergence of the algorithm;
- diffuses its information;
- receive the information sent by its neighbors $r_{ji}(t)$;
- updates its state with a combination of its own state and the states at its instantaneous and extended neighbors $(\overline{N}_i(t))$ as follows

$$x_i(t+1) = x_i(t) - \sum_{j \in N_i(t)} s_{ij}(t) + \sum_{j \in \overline{N}_i(t)} r_{ji}(t). \quad (3)$$

The evaluation of the state x_i under Algorithm 1 can therefore be understood by considering equation (3).

THEOREM 1. *if the assumptions 1, 2 and 3 are satisfied, Algorithm 1 guarantees that* $\lim_{t \to \infty} x_i(t) = \frac{1}{n} \sum_{i=1}^{n} x_i(0)$ *i.e., all node states converge to the average of the initial measurements of the network.*

The proof of theorem 1 is available in [2].

3.2 Practical Aspects

This section covers practical aspects for the implementation of Algorithm 1. The main two aspects are how to choose $s_{ij}(t)$ and how to overcome the loss of messages.

Each node updates its state following equation (3). This is achieved, by updating each sensors $s_{ij}(t)$ through time. For the sake of simplicity, the value of $s_{ij}(t)$ is chosen to be computed by the weighted difference between the sates of nodes i and j as follows: $s_{ij}(t) = \alpha_{ij}(t)(x_i(t) - x_j^i(t))$ if $x_i(t) > x_j^i(t)$, $s_{ij}(t) = 0$ otherwise.

The choice of $s_{ij}(t)$ is then deduced from the proper choice of the weights $\alpha_{ij}(t)$. Hence, $\alpha_{ij}(t)$ must be chosen such that the states

Algorithm 2 Temporally updating weights of node i.

```
1: for $j \leftarrow 1$ to $n$ do
2:     if $j \neq i$ then
3:         $s_{ij} \leftarrow 0$
4:         $\alpha_{ij} \leftarrow 0$
5:     end if
6: end for
7: $k \leftarrow 0$
8: $Sum \leftarrow 0$
9: find $\ell$ such that $\Delta_i^\ell = Delta_i[k]$
10: $\alpha_{i\ell} = 1/(\eta_i + 1)$
11: $s_{i\ell} = \alpha_{i\ell} \times \Delta_i^\ell$
12: repeat
13:     $Sum \leftarrow Sum + s_{il}$
14:     $k \leftarrow k + 1$
15:     find $\ell$ such that $\Delta_i^\ell = Delta_i[k]$
16:     $\alpha_{i\ell} \leftarrow 1/(\eta_i + 1)$
17:     $s_{i\ell} \leftarrow \alpha_{i\ell} \times \Delta_i^\ell$
18: until $NOT\ ((x_i - Sum \geq x_\ell^i + s_{i\ell})\ AND\ (k < n))$
```

at all the nodes converge to the average $\sum_{i=1}^{n} z_i/n$, in other words assumptions 2 and 3 must be satisfied.

First let define the deviation $\Delta_i^j(t)$ of node i as: $\Delta_i^j(t) = x_i(t) - x_j^i(t)$ if $j \in N_i(t)$ and $x_i(t) > x_j^i(t)$, $\Delta_i^j(t) = 0$ otherwise.

Algorithm 2 presents our method for temporally updating the averaging weights. Node i computes the difference between its current state and current states of its neighbors. The positive deviations $(\Delta_i^j > 0)$ are then stored in the array $Delta_i$, in a decreasing order. Then, it sets the weight α_{ij} to $1/(\eta_i(t) + 1)$, where $\eta_i(t)$ is the current number of its neighbors, starting by its neighbors nodes j whose have the larger deviations while respecting assumption 3.

Note that, this algorithm could be used with other formulas to compute α_i^j as long as assumptions 3 and 2 are ensured.

After computing weights, the nodes exchange their data and information with their neighbors. During these exchanges, some messages can potentially be lost. It will now be shown how to deal with such message loss. The main idea is that, each time some data $s_{ij}(t)$ is sent by a node i, it must be, at a later time, taken into account by the node j. This cannot happen if the message containing $s_{ij}(t)$ disappears, due to some communication failure. In order to be able to recover such lost message, instead of sending $s_{ij}(t)$, it is the sum $\Sigma_{s_{ij}}(t) = \sum_{0 \leq \tau \leq t} s_{ij}(\tau)$ that is sent. Symmetrically the receivers maintain the sum of the received data $\Sigma_{r_{ji}}(t) = \sum_{0 \leq \tau \leq t} r_{ji}(\tau)$. Upon receiving, at a time t, a message from node i, a node j can now recover all the amount of data that was sent before time $d_i^j(t)$. It has only to calculate the difference between the received $\Sigma_{s_{ij}}(d_i^j(t))$ and the locally stored $\Sigma_{r_{ji}}(t)$.

To conclude, the state messages exchanged during the execution of the algorithm are composed of two scalar values : the current state of the node, $x_i(t)$, and the sum of the sent data $\Sigma_{s_{ij}}(t)$.

4. EXPERIMENTAL RESULTS

To evaluate our approach, we conducted multiple series of simulations using the discrete event simulator OMNET++ [6]. The objective of these simulations is to confirm that our algorithm can successfully achieve desirable results for a wide range of sensor networks with an arbitrary distribution of sensor nodes. Therefore, we performed several runs of the algorithm (an average of 100 runs). In each experimental run, the network graph is randomly generated,

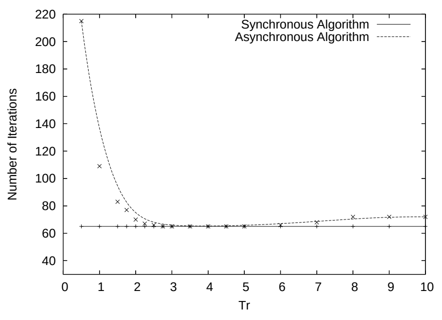

Figure 1: Number of Iterations

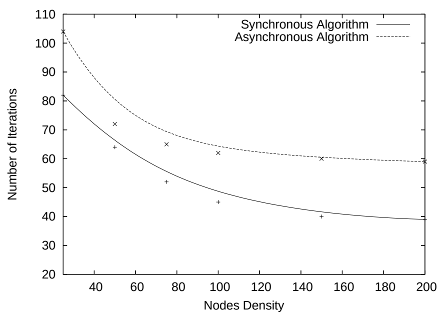

Figure 3: Number of iterations

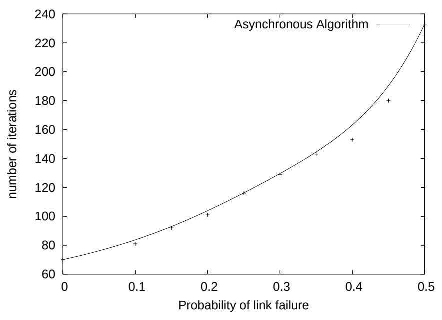

Figure 5: Number of Iterations

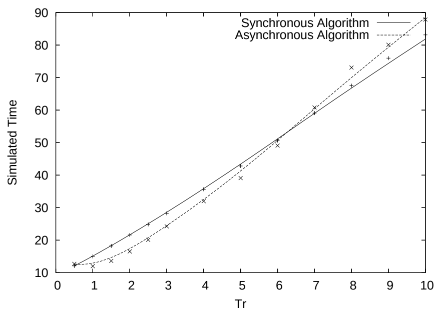

Figure 2: Simulated time

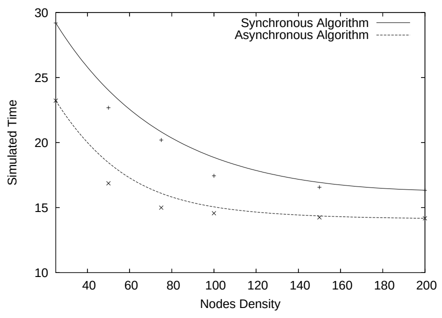

Figure 4: Simulated Time

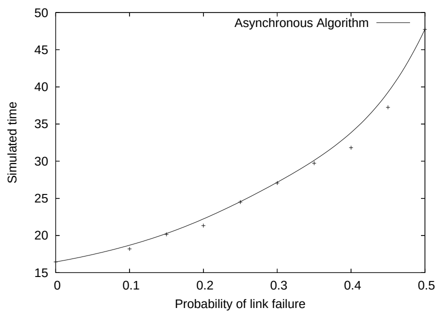

Figure 6: Simulated Time

where the nodes are distributed over a $[0, 100] \times [0, 100]$ field; then two nodes are connected by an edge if the Euclidean distance between them is less than 30. Furthermore, every node is randomly given an initial data z_i. They are normalized in such a manner that their average is set to one. After the topology-formation phase is completed, the nodes begin the iterative computation and exchange messages with their neighbors. A simulation experiment stops when the sensor network reaches global convergence. We have this condition when all the sensor nodes data x_i go to one.

In our simulations, we compared our algorithm in two modes synchronous and asynchronous. In the first one, each node has to wait until the whole neighbors update their data, before updating its data. On the other hand, in asynchronous mode, the sensor node operates independently without the need of waiting the other nodes.

We studied the performance of our algorithm with regard to the following parameters: (a) The time ratio T_r, defined as the ratio of the time taken by a node to perform an iteration over the time to transfer a message. This ratio and its impact will be studied in section 4.1. (b) The number of sensor nodes. We will show how our algorithm reacts once we increase the number of sensor nodes deployed in the same area. (c) The probability of the communication link failure p. As in sensor networks the communication links are prone to failure, we study the behavior of our approach under the dynamically changes of the communication graphs.

The main metrics we evaluated in this paper are, the mean error between the actual data x_i and the average of the initial data, the mean number of iterations and the overall time before reaching the global convergence. This time is the one simulated, as given by the discrete simulator OMNET++ [6]. For all the experiments, the global convergence state is said to be reached when $\varepsilon_i = |x_i - \sum_{i=1}^{n} y_i/n|$ becomes less than some fixed constant ε.

Note that, in the figures next sections, the points represent the obtained results and the curves are an extrapolation of these points.

4.1 Basic Behavior

First, we show simulation results for the case where we have a fixed topology with a fixed number of 50 nodes and $\varepsilon = 10^{-4}$. The essential parameter we varied in this section is the time ratio T_r.

Figures 1 and 2 show the variation of the mean of the number of iterations as well as the overall simulated time, while varying the ratio T_r in synchronous and asynchronous modes. In the synchronous algorithm, the number of iterations stays constant, which is not surprising since the iterations of the different nodes are synchronized. Furthermore, as expected in this case, the simulated time increases linearly with T_r.

In the asynchronous mode, it is noticed that the number of iterations is very high for low T_r ($T_r \leq 1$), then it decreases sharply until the value of iterations is equal to the value in the synchronous mode (at $T_r \approx 2.5$). For the higher T_r the number of iterations stays constant and is slightly greater than the number of iterations in the synchronous mode. Intuitively, this can be understood by the fact that, in the asynchronous case and for large T_r, the iterations are naturally nearly synchronized. If the time evolution in the two modes is tracked, it is observed that when $1 \leq T_r \leq 6$ the synchronous mode needs more time than the asynchronous mode to reach the global convergence. This is interpreted as follows: for such values of T_r, the cost of synchronizations is more important than the cost of the additional iterations needed by the asynchronous mode.

In a second experiment, the sensor network was fixed (50 nodes

$T_r = 2$) and the behavior of the algorithm was stimulated, in both synchronous and asynchronous modes. The mean error of the nodes $\varepsilon' = \sum_{i=1}^{n} \varepsilon_i / n$ was plotted see Figure 7. It can be seen that the convergence in the synchronous mode is faster than the convergence in the asynchronous one. It is also noticed that the two graphs have the same pace. In many scenarios an exact average is not required, and one may be willing to trade precision for simplicity. For instance, minimizing the number of iterations to reduce the energy consumption can be superior in sensor networks applications where exact averaging is not essential.

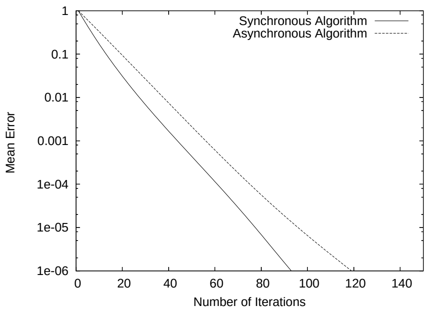

Figure 7: The Mean Error ε

4.2 Larger Sensor Network

Our scheme can be applied to sensor networks where a large number of sensor nodes are deployed, since it is fully distributed and there is no centralized control. In our simulations we varied the number of sensor nodes from 20 to 200 nodes, deployed in the region $[0, 100] \times [0, 100]$, we selected the time ratio $T_r = 2$ and for all nodes i, $\varepsilon = 10^{-4}$. However, as shown in the two Figures (Figure 3 and Figure 4), as the number of sensor nodes increases, the average of the iterations number as well as the time needed to reach global convergence decreases in the two cases synchronous and asynchronous. We notice that in the synchronous mode we obtained less number of iterations, on the other hand it takes more time to reach the global convergence than the asynchronous one. The higher the degree of a node, the more the precise would be the estimation of the average consensus and hence the lower will be the number of iterations.

4.3 Dynamic topology

In a next step, we simulated the proposed sensor fusion scheme with dynamically changing communication graphs. We generated the sequence of communication graphs as follows: at each time step, each edge in the graph is only available with a selected probability p, independent of other edges and all previous steps. To ensure the jointly connected condition of the generated graphs, we selected a period of time τ in which an edge cannot stay disconnected more than τ time. We fixed the number of sensor nodes to 50, the time ratio T_r to 2 and $\varepsilon = 10^{-4}$. In preliminary results, the period τ was chosen in a way that is equal to three times the time of a communication. We show in figure 5 and figure 6 the variation of the number of iterations and the time simulation with the probability of link failure p. We notice that the number of iterations and the overall time increase with the increase of the probability, but not in an exponential way.

Note that we also tried to run the synchronous algorithm with dynamic topology changes, but the execution times were so prohibitive, that we abandoned those experiments. These results confirm that synchronous algorithms are unfeasible for real sensor networks.

5. CONCLUSION AND FUTURE WORK

In this paper, we introduce a fault tolerant diffusion scheme for data fusion in sensor networks. This algorithm is based on data diffusion, the nodes cooperate and exchange their information only with their direct instantaneous neighbors. In contrast to existing works, our algorithm does not rely on synchronization nor on the knowledge of the global topology. We prove that under suitable assumptions, our algorithm achieves the global convergence in the sense that, after some iterations, each node has an estimation of the average consensus overall the whole network. To show the effectiveness of our algorithm, we conducted series of simulations and studied our algorithm under various metrics.

In our scenario, we have focused on developing a reliable and robust algorithm from the view points of asynchronism and fault tolerance in a dynamically changing topology. We have taken into account two points which don't have been previously addressed by other authors, namely the delays between nodes and the loss of messages. Knowing that in real sensor networks the nodes are prone to failures, one of the high-level goals of our future work is to allow nodes to be dynamically added and removed during the execution of the data fusion algorithm.

6. REFERENCES

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

[2] J. M. Bahi, A. Giersch, and A. Makhoul. A fault tolerant diffusion scheme for data fusion in sensor networks. Technical report, LIFC laboratory, 2008. http://info.iut-bm.univ-fcomte.fr/staff/makhoul/.

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

[4] R. Olfati-Saber, J. Fax, and R. Murray. Consensus and cooperation in networked multi-agent systems. *Proc. of IEEE*, pages 215–233, 2007.

[5] R. Olfati-Saber and J. S. Shamma. Consensus filters for sensor networks and distributed sensor fusion. *Proceedings of 44th IEEE Conference on Decision and Control CDC-ECC*, 2005.

[6] OMNeT++. http://www.omnetpp.org/.

[7] D. Scherber and H. Papadopoulos. Distributed computation of averages over ad hoc networks. *IEEE journal on Selected Areas in Communications*, 23(4):776–787, April 2005.

[8] D. Spanos, R. Olfati-Saber, and R. Murray. Distributed sensor fusion using dynamic consensus. *proceedings of IFAC*, 2005.

[9] A. Speranzon, C. Fischione, and K. Johansson. Distributed and collaborative estimation over wireless sensor networks. *Proceedings of 45th IEEE Conference on Decision and Control*, 2006.

[10] M. S. Talebi, M. Kefayati, B. H. Khalaj, and H. R. Rabiee. Adaptive consensus averaging for information fusion over sensor networks. *In the proceedings of The Third IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS'06)*, 2006.

[11] L. Xiao, S. Boyd, and S. lall. A scheme for robust distributed sensor fusion based on average consensus. *Proc. of the International Conference on Information processing in Sensor Networks (IPSN)*, pages 63–70, 2005.