Experimental Node Failure Analysis in WSNs

Jozef Kenyeres[1,2], Martin Kenyeres[2], Markus Rupp[1]
1) Vienna University of Technology, Institute of Communications, Vienna, Austria
2) Slovak University of Technology, Department of Telecommunications, Bratislava, Slovakia
Email: kenyeres@ktl.elf.stuba.sk, markus.rupp@tuwien.ac.at

***Abstract*—The main motivation of this paper is to bring detail analysis of an important aspect in wireless sensor networks (WSNs)- the node's failure. We focus on three scenarios of the node's failure and their effect on the average consensus algorithm implemented on a WSN. Gained results show clear connection between the network connectivity and the effect of the node's failure. The paper contains also references, how to deal with or at least reduce the effect of the node's failure.**

***Index Terms*—WSN; average consensus; node's failure; experiment**

I. Problem description

Wireless Sensor Networks (WSN) are in a general considered as fault tolerant systems. Due to their various applications in miscellaneous environments, the fault tolerance is an important design issue of every WSN application [1]. Besides effects from the surrounding environment that can greatly hamper the quality of communication, other significant sources of errors are WSNs devices itself. In ad-hoc networks they are assumed as simple and cheap devices with limited energy sources. The probability of failure of such a device is larger when compared to much more expensive devices from the traditional IT systems (IP networks or others). In [2] we presented implementation experience of the average consensus algorithm in WSNs. In our experiments we could exclude node failure scenarios. However, due to the importance of this issue, in this paper we expand the analysis and experiments to include such scenarios. The main contribution of this paper is to analyse, how such scenarios affect the algorithm performance and how to eliminate or at least decrease the effect of these events. In our analysis, we are targeting the scenarios of the node failure presented in [3]. The paper is organised as follows: in Section II we briefly describe the average consensus algorithm implementation and analysis inputs. In Sections III, IV and V we analyse the node's failure scenarios and the paper is concluded in Section VI.

II. Analysis setup

In this section we define the setup of our tested environment. We apply the average consensus algorithm implementation as presented in [1]. The implementation presented there was optimised for the fully connected topology. But the scope of this paper is not limited to the fully connected topology. Thus, we generalize the average consensus implementation and we optimize it also for different topologies. In this section, we provide the description of the generalized average consensus implementation and we describe the test topologies.

This work has been funded by the NFN SISE project S10609 (National Research Network "Signal and Information Processing in Science and Engineering")

A. The generalized average consensus implementation

The implementation presented in [1] was intended for a fully connected network, while the proposed TDM scheme created the global synchronisation model, which assigned unique time slots to every node in the network. Also the update of node's state was performed in a synchronous way after receiving messages from all neighbouring nodes- in this case all nodes in the network. For other than a fully connected topology this approach is mismatched as a node does not neighbour with all other nodes in the network and it is difficult and inefficient to establish and manage this method for a global synchronisation.

The average consensus algorithm in the discrete time stated version from [4] and implemented in [1] is stated as follows:

$$x_i(k+1) = x_i(k) + \varepsilon \sum_{j \in N_i} a_{ij}(x_j(k) - x_i(k)). \qquad (1)$$

Here, $x_j(k)$ denotes the state of node *i* at time instant *k*, N_i defines the neighbourhood, that is which nodes are being received by node *i*, the element a_{ij} of the adjacency matrix *A* is defining if the node *i* and the node *j* are neighbours and ε is the so-called mixing parameter. In a practical implementation it is not possible to interpret *k* as an instant time. Thus, instead of a single temporal point, the iteration is represented by an interval of length *T* and (1) should be stated as follows:

$$x_i(T_{k+1}) = x_i(T_k) + \varepsilon \sum_{j \in N_i} a_{ij}(x_j(T_k) - x_i(T_k)). \qquad (2)$$

Value $x_i(T_k)$ represents the state of node $n_(i)$ after performing the *k*-th iteration. The value of *T* is constant for all iterations. In practice, it should be dynamically adapted to optimize the algorithm performance and reduce the energy consumption of nodes, but this is beyond the scope of this paper. To avoid collisions we created a simple back-off mechanism that allows the nodes transmitting in pseudo-random random time inside the interval T_k. The receiving node is always checking if the message has the correct iteration number. If the number is not corresponding, the received value is denied. By this way it is guaranteed, that value $x_i(T_{k+1})$ is counted only from $x_j(T_k)$.

Other important aspects remain unchanged from the implementation presented in [1]. The consensus is reached locally

in every node, when the value of $x_i(T_k)$ is changing below the defined treshold value (in our analysis is the treshold value set to 0.0005). The consensus in the network is reached, when every node in the network reaches the consensus. More details are included in [1]. The proposed generalized average consensus was implemented in *Matlab* for simulations and also in *nesC* for experiments. *NesC* is the programming language suitable for embedded systems such as nodes and it is an extension of the well-known *C* language. *NesC* is running under *TinyOS*, an event-driven operating system, specifically developed for wireless sensor networks. Please find more details in [5,6]. For the practical realisation, we used the Memsic's school kit hardware (more details in [1,7]). Results gained from simulations and practical realisation are consistent, thus in the paper we present more accurate simulation results gained from much larger number of repeats.

B. The test topologies

The important aspect in the analysis of a node's failure is the network topology. The impact of a node's failure is very different in the case of a strongly connected network, where every node has a large neighbourhood and in the case of a weakly connected network with a limited neighbourhood of single nodes. To analyse the relation between these effects and network connectivity, we use five different topologies shown in Fig. 1. In our presented results, we limit the network size to ten nodes. In ad-hoc networks the average consensus algorithm shows a linear behaviour; the gained results are also useful in general. The gained results for larger networks match with this statement, but due to the space limitations are not presented in the paper.

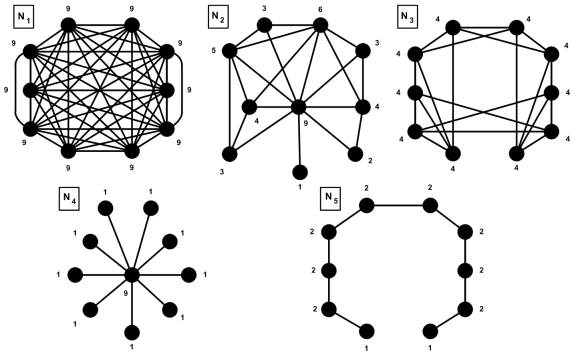

Fig. 1. Test networks

The presented topologies, stated as $N_1, N_2 N_5$, greatly differs in the connectivity. N_1 is fully connected, so the number of connections ($\|N_1\|$) equals to $|N| * (|N| - 1)$, where $|N|$ is the number of the nodes (in all presented topologies $|N| = 10$). Weakly connected networks are N_2 and N_3, with $\|N_2\| = \|N_3\| \cong 1/2 * \|N_1\|$. The main difference between these topologies is the number of neighbours of single nodes. While in N_3 every node is neighbouring with four other nodes ($\forall d(n_i) = 4$), in N_2 the number of nodes is different for single nodes and it presents the central node with $d(n_j) = |N| - 1$. Minimally connected networks are N_4 and N_5 with $\|N_4\| = \|N_5\| \cong |N| - 1$.

In our analysis, we focus on two main characteristics of the algorithm performance. The first is the number of iterations needed to reach the consensus. We use the average number of iterations among all nodes in the network and in all graphs this parameter is labelled as *Iterations*. This parameter determines how rapidly the algorithm reaches the consensus. The second parameter is the computation precision. We use a simple metric to quantify the quality of the gained results- Mean absolute percentage error (*MAPE*) stated as follows:

$$MAPE = \left[\frac{1}{N} \sum_{i=0}^{N} \left| \frac{y_i - y_{exp}}{y_i} \right| \right] * 100[\%]. \qquad (3)$$

Here, y_i is the result of the average consensus algorithm gained on the *i*-th node and y_{exp} is the expected result counted as $y_{exp} = 1/N \sum_{i=1}^{N} x_i(0)$. These parameters allow the determination of how much the algorithmic performance is affected by the node's failure. Before we start with the analysis, in Fig. 3 and Fig. 4 results are presented for all objective topologies gained in the ideal state without any node's failure. The mixing parameter was changed in the interval $< 0,0025; 0,1975 >$, what corresponds to the convergence bounds defined in [1].

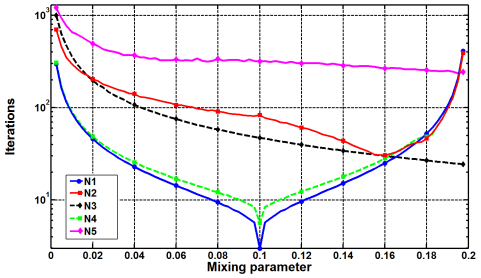

Fig. 2. Iterations vs. mixing parameter

From Fig. 2 it is clear, that the number of iterations required for reaching the consensus greatly differs for different topologies. The gained results are conform to our intuitive expectations- in the network topology with smaller connectivity, more iterations are needed than in a strongly connected network, because the information from nodes is spread slower through the network than in a strongly connected network. The only exemption is the topology N_5, but the error characteristics shown in Fig. 3 are explaining this situation.

Results gained for the error rate of computed results are also conforming to the intuitive expectations. In strongly connected networks results are almost perfect, but with a decrease of the connectivity, the error is increasing, especially for the mixing parameter value lower than $1/N$ (0.1). The topology N_5 is clearly a special case. The error rate measured here is much higher than in the other cases with the peak for the mixing parameter being $1/N$. This is the result of limited information spread in the network- for all nodes expect itself this central node is the only source of information and if this

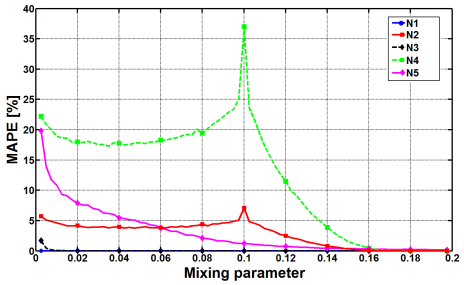

Fig. 3. Error rate vs. mixing parameter

node converges too early, than the other nodes are not able to finish their computation.

In general, it should be concluded that with the decrease of connectivity the number of iterations is increased (with the exception of some special cases). The conclusion for error rate is difficult, but it is clear that with decreasing connectivity or by the presence of the central node the error rate is increasing. As a result, even in the ideal state without any node failure, the algorithmic behaviour in different topologies is very complex and depends on a large scale of conditions. Thus, for a thorough analysis of a node's failure it is important to include the aspect of the network connectivity.

III. Node's death

In the first scenario of interest, we are assuming that a specific node nd is dead, thus it is not transmitting anymore [3]. The network N adapts to a new solution and it is expected that$y_{exp} = 1/(N-1)\sum_{i=1}^{N-1} x_i(0)$ for all $n_i \in N/\{n_d\}$. This statement is true until it does not violate the robustness condition defined in next subsection.

A. Robustness condition

The network (or graph in a Graph theory terminology) *N* adapts to a new solution y_{new} only if after the removal of the node n_d the graph N_{new} remains connected, i.e. if there is a path connecting any of two nodes (vertices) in N_{new} [8]. This condition holds, until the vertex n_d is not a cutvertex of the network N_{new}. The cutvetex of a graph is vertex, which removal leads to a disconnection of the graph. If node n_d is a cutvertex or it is not depends on the topology. From [8], every graph should be described by the parameter vertex connectivity $(K(N))$, i.e the minimal number of vertices whose removal leads to disconnecting the graph. Thus, the robustness condition should be stated as follows:

$$|D| < K(N) \wedge n_d \notin C(N); \{\forall n_d \in D\}. \quad (4)$$

Here, *D* is set of removed vertices and *C(N)* is set of cutvertices of the graph *N*. In our following analysis of the dead node scenario we are expecting that the robustness condition is fulfilled. In the case if it does not, the consensus is not reached in the network. Then, only a partial consensus is obtained for the group or groups of nodes, which remain connected. But these cases are out of our scope.

B. Analysis

As first, we analyse how effectively the network *N* adapts to a new solution y_{new} after one of $n_i \in N$ dies, so $|D| = 1$. For our analysis we choose networks N_1 and N_3, in which all nodes have the same $d(n_i)$. Gained results are shown on Figures 4 and 5.

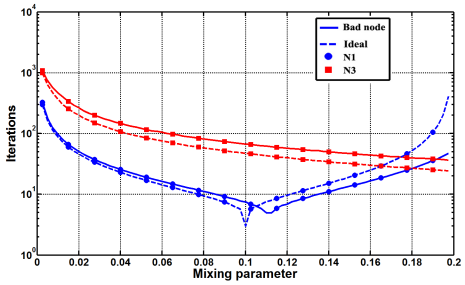

Fig. 4. Error rate vs. mixing parameter

From our gained results it is clear, that in terms of iterations both networks differ very slightly from the ideal case without a node's failure. The difference is caused by the relation between the mixing parameter and the network size. The result gained for the error rate is very similar and the error rate is affected only very slightly after one of the nodes dies.

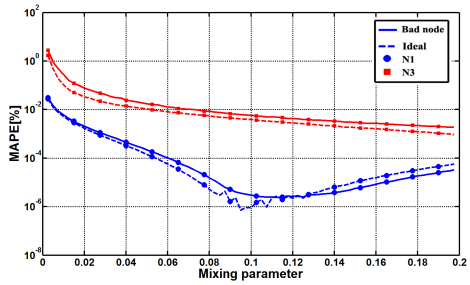

Fig. 5. Error rate vs. mixing parameter

Another point of interest is the analysis of the efficiency of the adaptation to the new solution y_{new} when $|D| > 1$. The result is shown in Fig. 6. We focus on the same topologies, N_1 and N_3. In the graph results are included for three different values of the mixing parameter. The gained results for the network N_1 are corresponding with the expectations- the relation between the adapting efficiency and the number of removed nodes is almost a linear function. The gained results for the network N_3 are far away from some kind of function. The difference is caused by the fact that removing the node in N_1 creates again a fully connected network of $N-1$ nodes. In the network N_3 the situation is not so clear and after removing the node, the topology is changed and the behaviour is hardly predictable.

Another important aspect in this node's failure scenario is the number of neighbours $(d(n_i))$ of the dead node. Theoretically, we expect that the removal of the node with larger $(d(n_i))$ should affect the network performance more than the removal of the node with a smaller *d*. Gained results from

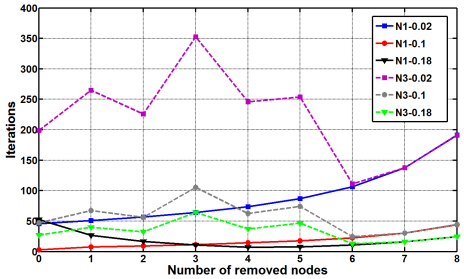

Fig. 6. Error rate vs. mixing parameter

the network (N_2) are shown in Fig. 7- because this network shows the greater variety of ($d(n_i)$) from all tested networks. In the graph results are included for three different values of the mixing parameter. From the results it is clear, that it is not possible to find some clear relation between ($d(n_i)$)) of the removed node and the network performance.

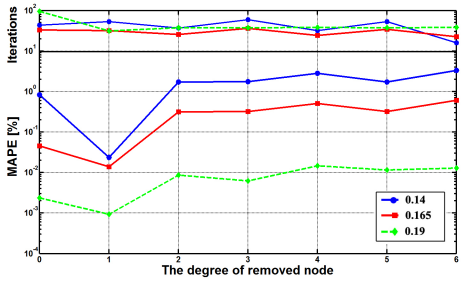

Fig. 7. Error rate vs. mixing parameter

IV. Fixed update

Another node's failure scenario from [3] is the fixed update node's failure. In this scenario, the node (n_{fixed}) is instead of $x_i(k)$ transmitting the fixed value y_{fixed} in all iterations. Such error has devastating effect on the whole network. Even the consensus is reached, instead of $y_i, \forall n_i \in N$ converge to y_{fixed}. Also the number of iterations is greatly increased, what is shown in Fig. 8. These statements holds, until in the network is present one n_{fixed}. In other case the consensus is not reached anymore.

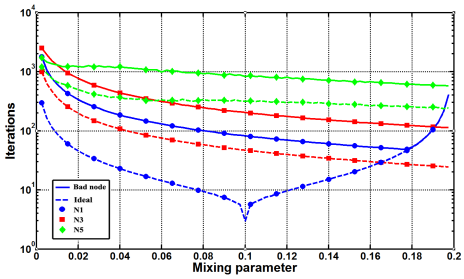

Fig. 8. Error rate vs. mixing parameter

A. Analysis

In our implementation we built a simple mechanism, how to avoid such catastrophic situation. After the message is received, the receiving node checks the number of the iteration in the received message. And if the number is not corresponding to its local iteration number, the message is discarded. Thus, we were interested in two similar scenarios. In the first sub scenario the fixed node is not able to update its local value and while this update is placed in the same task as the iteration update, all other message besides the first are denied. In the second sub scenario, the node is able to update its iteration counter. But if everything in the node works correctly, besides updating the fixed value of course, than the node is able to perform the consensus test and after three iterations this node reaches the consensus. As a result, the consensus is reached also for the network with more than one n_{fixed}. Nevertheless, also these sub scenarios are far from perfect. Receiving fixed values only in the first iterations affects the final result at most, thus also in this case its final error rate is clearly increased. The gained results for the second sub scenario are shown in Fig. 9.

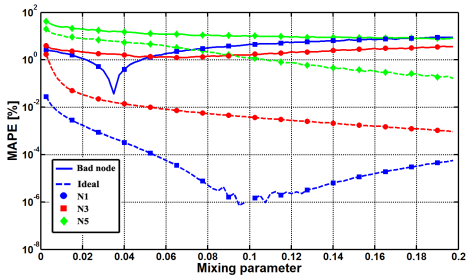

Fig. 9. Error rate vs. mixing parameter

Another interesting aspect is the relation between $d(n_{fixed})$ and the caused effect on the network performance. We expect that with the increasing $d(n_{fixed})$ also the error rate is increasing, while it means that more nodes receive the fixed value. In contrast to the first scenario with a dead node, the relation is in this case very clear. The final error rate is a function of the value of $d(n_{fixed})$. The results for the network N_2 are shown in Fig. 10 for $d(n_{fixed}) = \{1, 3, 6\}$.

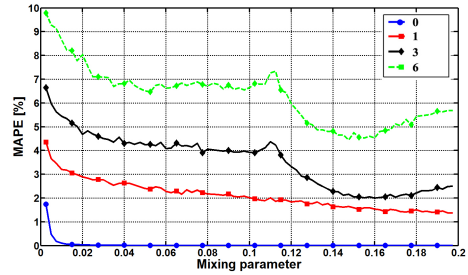

Fig. 10. Error rate vs. mixing parameter

V. Misbehaving update

The last node's failure scenario from [3] is again related to the node's update. In this case, the corrupted node (n_{badUp}) is not able to calculate its value $x_i(k)$ and instead of this, it updates its state with one of the received values from its neighbours of the previous iteration. The effect of such error on the error rate is shown in Fig. 11. The number of iterations

in this case is very similar to the number of iterations in the dead node scenario, i.e. this kind of error does not affect the number of iterations significantly. On the other hand, the error rate is affected greatly. From the graph it is clear, that the effect is very similar to the effect of the fixed update node from the second presented sub scenario. But in contrast, in this case we did not achieve an error rate close to 1%, while in the previous case it is achieved at least for small values of the mixing parameter.

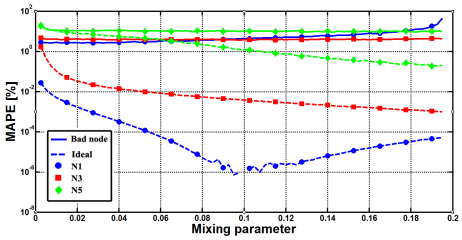

Fig. 11. Error rate vs. mixing parameter

A. Analysis

In comparison with the fixed update scenario, this error does not lead to such catastrophic results. But on the other hand, in our implementation, mechanism is built in to reveal the corrupted node. Not even n_{badUp} is able to recognize, that something is wrong. Consequently the gained results are worse than in the fixed update scenario after applying control mechanisms included in our implementation. We are expecting, that n_{badUp} takes randomly one of the received values, thus $x_badUp(T_{k+1}) = x_i(T_k)$ where ni is one of n_{badUp} neighbours. The node n_{badUp} reaches the consensus when $d(n_{badUp}) = 1$. In this case n_{badUp} is receiving only one value and reaches the consensus because its local value is not changing anymore.

As in the previous scenario, another interesting aspect is the relation between n_{badUp} and the caused effect on the network performance. The results for the network N_2 are shown in Fig. 12. The gained result is very similar to the results from Section IV, it is clear that the effect of the misbehaving update is increasing with increased $d(n_{fixed})$.

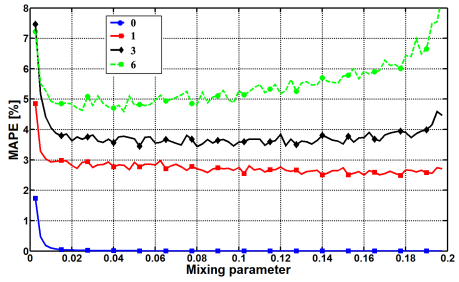

Fig. 12. Error rate vs. mixing parameter

VI. Conclusion

Our analysis is suggesting the effect of the node's failure on the WSN performance.In the first node's failure scenario- the node's death, we tested the ability of the network to adapt on the new situation, after one of nodes is not transmitting anymore. While the defined robustness condition hold, the network was able to reach the consensus. We were also interested, if it is possible to predict the efficiency of the adaptation based on the $d(n)$ of the dead node and also if the death of more nodes led to predictable behaviour. The gained results showed, that such complex process is not possible to describe without deeper analysis of the network topology. Avoiding or reducing the effect of the node's death is possible only by the increased connectivity in the network. The second scenario- the fixed update error, is very dangerous to the network performance, while without any protective mechanism it leads to catastrophic results. We presented two possible scenarios, how to reduce the effect of such error. The network behaviour is much more predictable than in the scenario with a death node, while the caused effect on the network performance clearly depends on the value of $d(n_{fixed})$. From our gained results it is also clear, that for networks with the stronger connectivity it is possible to minimise the effect of the fixed update error by using smaller values of the mixing parameter, where the error rate is close to 1%. The results gained in the third scenario- the misbehaving update, was very similar to the fixed update scenario with applied control mechanisms of our implementation. On the other hand, decreasing the effect of misbehaving update is much more complicated. The node itself is not able to recognize, that it is misbehaving. Only possibility is to create an additional control mechanism that allows the node to predict, how the value of $x_i(T_k)$ changes in the next iterations. The only possibility how to identify the corrupted node is monitoring of the unusual fluctuation of $x_i(T_k)$, but due to the limited information available locally in the node it is not a trivial task. In a general, all these scenarios points on a need for a presence of control elements in WSNs applications, while a corrupted node means serious threat to the performance of whole network.

References

[1] I.F.Akyildiz, W.Su, Y.Sankarasubramaniam and E.Cayirci,"Wireless sensor networks: a survey, " in Computer Networks, Vol.38(4), pp. 393-422, 15 March 2002.

[2] J.Kenyeres, M.Kenyeres, M.Rupp, and P.Farka, "WSN implementation of the average consensus algorithm," in Proc. of 17th European Wireless Conference, Vienna, Austria, Apr. 27-29, 2011.

[3] O.Sluciak and M.Rupp, "Reaching Consensus in Asynchronous WSNs: Algebraic Approach," in Proc of ICASSP 2011, Prague, Czech Republic, May 2011.

[4] R.Olfati-Saber, J.A.Fax, and R.M.Murray, " Consensus and Cooperation in Networked Multi-Agent Systems , " Proceedings of the IEEE, Vol. 95, No. 1, pp. 215-233, Jan. 2007.

[5] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler, "The nesC Language: A Holistic Approach to Networked Embedded Systems," in Proceedings of Programming Language Design and Implementation (PLDI) 2003, June 2003.

[6] P. Levis, "TinyOS Programming," csl.stanford.edu/ pal/pubs/tinyos-programming.pdf

[7] Memsic, "IRIS wireless measurement system," http://www.memsic.com/products/wireless-sensor-networks/wireless-modules.html

[8] R. Diestel, "Graph Theory (3rd ed.)", Berlin, New York: Springer-Verlag, ISBN 978-3-540-26183-4