 ISBN 978-3-8007-3343-9 © VDE VERLAG GMBH

WSN Implementation of the Average Consensus Algorithm

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

***Abstract*—This paper is motivated by the lack of distributed algorithm implementations on wireless sensor networks (WSN) in hardware. We deal exemplary with the implementation of the well-known average consensus algorithm. By formulating the algorithm into nesC, a C derivative, it is possible to enrich the knowledge of the algorithm with practical information, specific to embedded devices such as nodes. We created a simple mechanism of a time scheduled access to share the wireless channel among the nodes and guarantee a collision free environment, in which our implementation is tested. Our achieved results are consistent with theory.**

***Index Terms*—WSN; average consensus; distributed algorithms; implementation**

I. Introduction

Distributed algorithms on wireless sensor networks (WSN) are currently an interesting research field in signal processing [1-8]. The main concept of WSNs as defined in [1] is based on the idea of distributed devices equipped with sensors for cooperative monitoring of environmental quantities. While the research in this field is mostly based on simple models of WSNs including the wireless channels, an implementation of a distributed algorithm needs to be suited for a given hardware platform [9,10]. In order to obtain more realistic scenarios for modelling we implemented a typical distributed algorithm, the average consensus algorithm, on an available platform. The paper is organized as follows: in Section II we briefly explain the well-known average consensus algorithm and we add simulation results to serve as a reference for our measurements. Section III contains a brief hardware overview and Section IV describes the actual implementation process. In Section V we explain our experimental setup and in Section VI we present simulation as well as experimental results for a link error analysis. Section VII presents experimental results and finally some conclusions in Section VIII close the paper.

II. Average consensus algorithm

The average consensus is a relatively simple, well known algorithm that is instrumental for many advanced schemes like subspace tracking [5,6] or joint probabilistic approaches [7,8]. Because of its simplicity it was chosen as potentially useful for an implementation on a real WSN hardware. A consensus is defined in [2] as reaching an agreement of participants (in case of WSN nodes) in the question of a particular value of some quantity, depending on the local values of all participants. A consensus algorithm is a set of rules, defining shared information between participants, eventually leading to reach the agreement.

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

The average consensus algorithm was selected from [2] in its straightforward fully synchronous form, stated in discrete time 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, $x_i(k)$ 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* and ε is the so-called mixing parameter. The element a_{ij} of the adjacency matrix *A* is defining, if the node *i* and the node *j* are neighbours. For neighbouring nodes the value is equal to 1, in the opposite case it equals 0. All elements on the main diagonal of the matrix *A* are equal to 1.

Before the experimental part was performed, some simulations were run to determine the basic behaviour of the average consensus algorithm. As we expect in our network that all nodes are included in their neighbourhoods ($|N_i| = N$ for every node in the network), we set the simulations to a fully connected network (all elements of the matrix *A* equal to 1) with a varying number of nodes *N* and a varying mixing parameter ε. In Fig. 1 we show the connection between the mixing parameter and the average number of iterations needed for reaching the consensus. In the simulation, the consensus is reached when all nodes converge to the same value of the average. As Fig. 1 shows, the fastest convergence is obtained for $\varepsilon = 1/N$, that is in a single step. We will call this specific value the *threshold value*.

For the number of iterations needed to reach the consensus the following statement is of importance: when the mixing parameter is set to the *threshold value* ($\varepsilon = 1/N$),for *N* also being the number of iterations to achieve a certain quality, the achieved consensus value quality would be the same no matter what size *N* the network has. It means, that the number of required iterations is the same in the case of five nodes as well as in the case of ten nodes, when the mixing parameter is set to $\varepsilon = 1/5$ for five nodes and $\varepsilon = 1/10$ for ten

nodes. An equivalent behaviour is obtained also for multiples of the threshold value, e.g. $0,5/N$. Assuming a fully connected network (all $a_{ij} = 1$) in (1), the first iteration is computed as

$$x_i(1) = x_i(0) + \frac{1}{N}\sum_{j \in N_i}(x_j(0) - x_i(0)). \quad (2)$$

In the first iteration an average of differences is computed, so that every node directly obtains this average value. This property is used for the comparison of the results in Section VII.

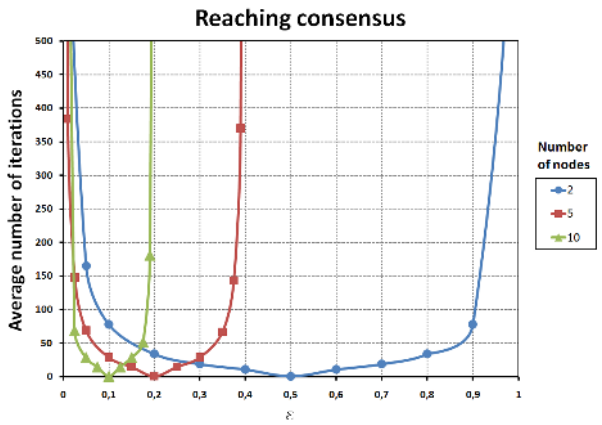

Fig. 1. Simulation results

For mixing parameters different from the *threshold value*, the number of iterations is increasing with an increased difference from the *threshold value*. The width of the converging range varies due to the number of participants: for two participants, an upper convergence bound for the mixing parameter is 1, for five participants it is 0,4 and for ten participants it is 0,2. In general, the upper convergence bound for the mixing parameter is $\varepsilon = 2/N$. The lower convergence bound of the mixing parameter is 0 independent of *N*. Thus, for the fully connected network consisting of *N* nodes, the network converges to the consensus if ε fulfills the condition

$$0 < \varepsilon < \frac{2}{N} \quad (3)$$

For two participants, the *threshold value* is 0,5, the upper convergence bound is 1. In this case for a mixing parameter set to $\varepsilon = 0,9$ the algorithm needed 78 iterations to reach the consensus, for $\varepsilon = 0,99$ it needed 857 iterations, and for $\varepsilon = 0,999$ it needed 8624 iterations. The lower bound is 0, for $\varepsilon = 0,001$ the algorithm needed 8640 iterations for reaching the consensus.

III. WSN HARDWARE PLATFORM

The average consensus algorithm was implemented on a WSN hardware platform from the company Memsic. It is called *Classroom kit* and it is intended for academic programs and testbed setups. The set consists of 30 wireless modules, 20 sensor and data acquisition boards, as well as 10 gateway and programming boards. Besides hardware, the set is equipped with software to form and operate a network, including tools for visualisation and real-time analysis. Fig. 2 displays a part of this kit.

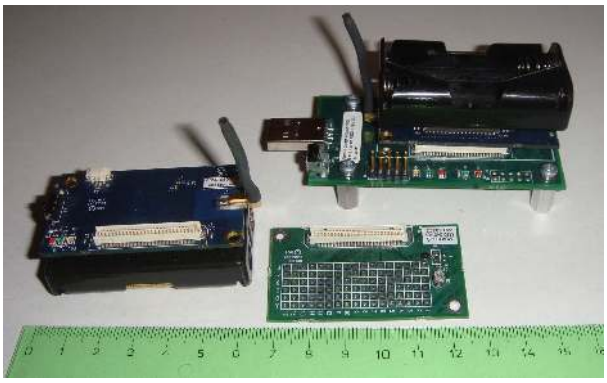

Fig. 2. Memsic's classroom kit.

A. Communication platform

A hardware module called IRIS is responsible for handling the communication between the nodes as well as data processing. The IRIS RF part is a 2.4 GHz IEE 802.15.4 module tailored for WSNs. Its main properties from [11] are listed in Table 1.

TABLE I
IRIS PLATFORM PROPERTIES

Program Flash Memory	128 KB
Measurement Flash	512 KB
RAM	8 KB
Configuration EEPROM	4 KB
Serial Communications	UART
Analog/Digital Converter	10 bit ADC
Frequency band	2405 MHz to 2480 MHz
Transmit data rate	250 kbit/s
RF power	-17 dBm to 3 dBm
Receive Sensitivity	-101 dBm

IV. ALGORITHM IMPLEMENTATION

The average consensus algorithm was implemented in the programming language *nesC*, an extension of the well-known C language, suitable for embedded systems such as nodes. NesC is running under *TinyOS*, an event-driven operating system, specifically developed for wireless sensor networks. Considering unusual communication properties of WSNs in comparison to traditional networks like TCP/IP or others, the programming language has to support special needs. The most important properties supported by nesC are an event-driven execution, a flexible concurrency model, and a component-oriented application design [12].

In Fig. 3 we exhibit a flow-diagram of the implemented average consensus algorithm. The functionality of the algorithm is placed into *tasks*. A *task* is a building block of nesC implementations and contains source code, performing some executable function, e.g. computing $x_i(k)$.*Tasks* are handled

by the TinyOS scheduler, obey run-to-competition rules and they are performed synchronously. This is very important, because it allows the creation of the synchronous application. The source code of the synchronous application is more resistant to errors, for example data races. One of the basic rules, how to write a source code in nesC, is keeping it synchronous as much as possible [13]. These *tasks*, performing the algorithmic functionality, are called from inside of *events*. In contrast to a *task*, an *event* is signalled asynchronously. Due to the properties previously mentioned in this section, the nesC code has to be able to interact with the environment- in this case neighbouring nodes, participating in the average consensus computation. When the node receives a message from the neighbouring node, TinyOS signals the *message received event*. The *event* is responsible for proper writing received data to memory, performed by *tasks* called within the message receive *event*. Another type of *event* is called the *timer fired event*. When a timer reaches zero, then its *timer fired event* is called. This *event* triggers *tasks* to perform some action, e.g. computing some value or a sending a message.

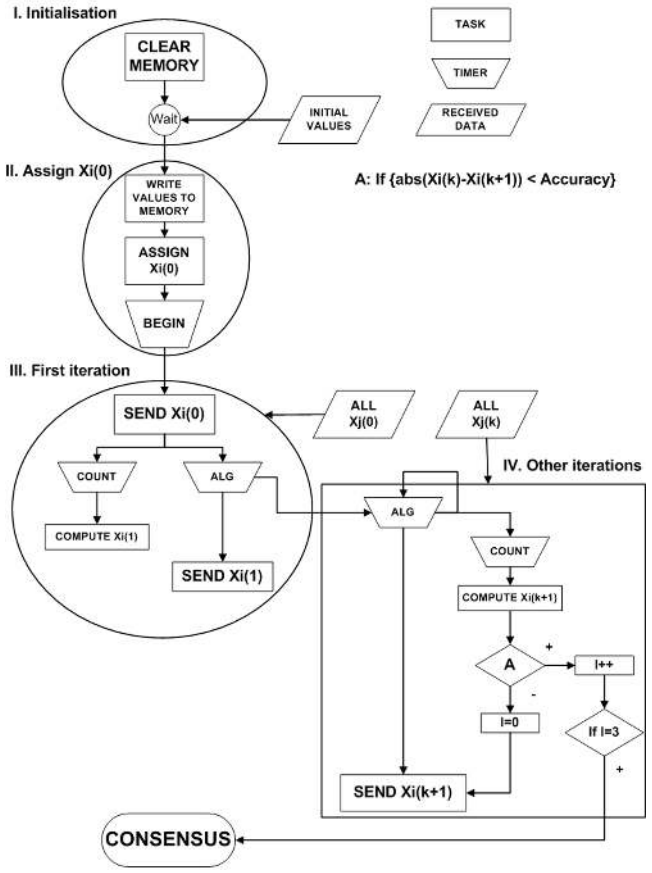

Fig. 3. Average consensus implementation.

The algorithm implementation is performed on the node with limited resources. Thus, the source code needs to be as much effective as possible: in the memory only actually needed data is stored, messages are kept very short and sent as broadcast to reduce their number. The algorithm works without human interaction and is able to deal with a node failure, because all needed information for the computation is gained in every algorithm's iteration. Thus when a node fails, the algorithm remains consistent.

The implementation is divided into four parts: *initialisation, assign local value, first iteration* and *other iterations*.

A. Initialisation phase

This phase begins after turning the node on- the node's on-board memory is cleared and the node waits until a message with initial values is received. Initial values are received in a special message sent by the initializing node. It is divided from regular messages with a different AM (Active Message number- included in a TinyOS header of every message).

B. Assign local value

After receiving the initial message, tasks are called for writing these values to the memory and selecting the initial value for each node, as the initializing message is broadcast with values for all nodes. Also the timer labelled *BEGIN* starts the average consensus computing.

C. First iteration

When the timer *BEGIN* is fired, every node broadcasts its value $x_i(0)$. There are two other timers started, called *COUNT* and *ALG*. Before they are fired, every node receives all $x_j(k)$. The timer *COUNT* is fired first and counts the local average estimation in the first step $x_i(1)$. When *ALG* is fired, computations are finished and nodes broadcast $x_i(1)$. The timer *ALG* is a periodically repeating timer. After firing it is started again.

D. Other iterations

Other iterations are slightly different from the first one, because *COUNT* is called from *ALG*. After sending $x_i(1)$, all $x_j(1)$ are received and when *COUNT* is fired, they are stored in the memory. *COUNT* computes $x_i(2)$ until *ALG* is fired. In all iterations except the first *COUNT*, the node performs a consensus test in *A*, that is, it compares the actual value $x_i(k)$ with $x_i(k-1)$.If the difference is smaller than a defined accuracy parameter (*Accuracy*), the counter l is increased. When l is 3, the computed value is changing in a smaller interval than the accuracy parameter or it is even completely the same for three times in a row. Then the computations are stopped and the consensus is reached.

V. Experimental setup

A. Communication parameters and Topology

During our experiments nodes were placed in a room of approx. 4x10m. The nodes communicated at 2405MHz and were set to the minimal transmitting power (-17dBm). In a room with such dimensions every node is in the communication range of every other node, creating a fully-connected topology. This corresponds to $a_{ij} = 1$ for $i = 1...N, j = 1...N$, in (1), as proposed in Section II.

B. Messaging

Every message was sent as broadcast to reduce the number of messages to a minimum. There are two types of messages sent in the implementation: initializing messages, consisting of 10 numbers in an 8bit integer format, and regular messages. From these values every node chooses an initial value, depending on the node's ID. The format of a regular message that is sent between nodes is shown in Fig. 4.

In the payload of the message the address of the transmitting node is placed, as well as the number of iterations, the value $x_i(k)$ in integer as well as in a floating point format. The floating point format is used for computations, the corresponding integer value is only informal to show results in readable form. Every message includes a TinyOS header, needed for sending and receiving messages in TinyOS, containing the destination address, length, AM group and the message type. At the end of each message a CRC is included for data verification. A message thus consists of 14 bytes.

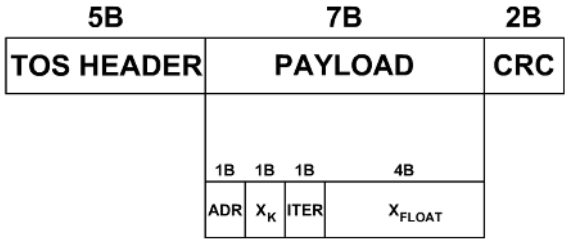

Fig. 4. Regular message.

C. Collision free setup- TDM

To avoid collisions on shared wireless channels, nodes are often equipped with a CSMA-CA protocol. This "best effort" solution is far from being perfect [14], especially when nodes are close to each other and transmit almost at same time.

On five nodes with different mixing parameters ε we tested how much collisions can affect the result of the algorithm. The nodes were placed close to each other and performed the algorithm. In Fig. 5 we show the relative error of the average consensus after ten algorithm's executions (in a row). In Fig. 5 the relative error of the computed result after every single execution is shown for different mixing parameters.

The graph in Fig. 5 shows the dependence between the relative error and the mixing parameter. The number of iterations is bound to the mixing parameter value. Thus, the graph also suggests the dependence between the number of iterations and the relative error. When the value of the mixing parameter results in a smaller number of iterations (for $\varepsilon = 0,2$ it was from 3 to 7 iterations, for $\varepsilon = 0,1$ from 10 to 16, for $\varepsilon = 0,05$ from 19 to 31), the computed relative error was higher than in cases where the mixing parameter results in more iterations. Especially in the case with less iterations (when the mixing parameter is set to $\varepsilon = 0,2$), the relative error ranges from 30% to almost 50%. As a result, the effect of collisions was much more evident for the algorithm's performances with a smaller number of iterations.

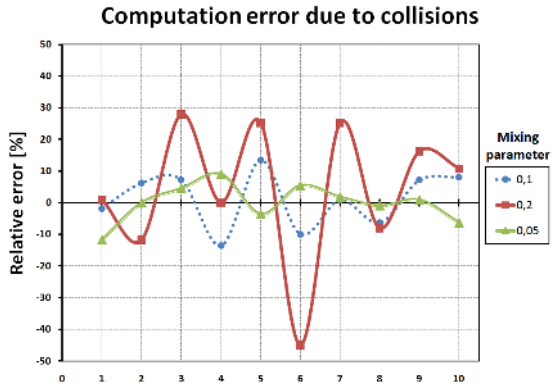

Fig. 5. Relative computation error over ten algorithm executions.

To circumvent this problem we propose a different solution. Because of the close distances between nodes, our implementation provides the solution of the collision problem by accessing the shared channel in a well defined order. In all iterations, every node has its own time slot to send its message without interfering with other nodes. Our implemented collision free TDMA setup scheme is illustrated in Fig. 6.

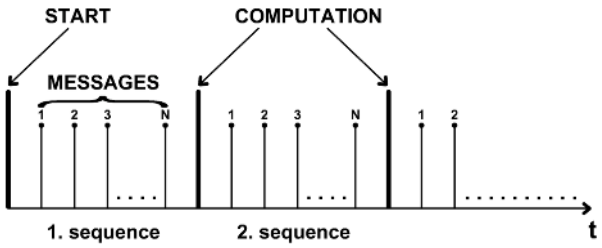

Fig. 6. Collision free setup message timing.

The selection of the time slot is based on the node's ID, a unique identifier of the node. Nodes are synchronised at the moment of receiving their initial message, but the transmission of messages is delayed for every node by adding *100*Node_ID* [ms] to the *BEGIN* timer. Other timers are keeping the synchronisation until reaching the consensus. Due to this simple mechanism nodes are able to coexist in a collision free setup.

D. Initial values

Initial values were sent from an initialising node in the initialising message as mentioned before. The initialising node was not participating in the average consensus computation. This node sent the initialising message after being switched on and was switched off right after. Initial values were integer values with different standard deviation to investigate the effect of different values on the consensus algorithm. In a WSN without such initializing node, the nodes first need to find a solution of this slot assignment problem in their own neighbourhood.

E. Computation accuracy analysis

Due to their limited storage and transmission quality WSNs may suffer accuracy. On the other hand offering too high precision may cause a large and undesired communication overhead. It is thus of importance to find the best word length. Alternative formulations of the consensus algorithm to overcome accuracy effects have been proposed [15] and analysed [4] recently. As first, we limited our computation precision to integer numbers (the *uint_8t* format). Thus, the value of $x_i(k+1)$ was calculated from the set of received $x_j(k)$, which were transmitted in the integer format. Though the value of $x_i(k+1)$ was on the node *i* stored as the real number in floating point format (the *float* format), we lost too much precision in every algorithm's iteration. The final result was greatly affected by such an inaccuracy. Then we tested versions of the algorithm with rounded real numbers in the float format, trimming the values to only two decimal positions before transmitting. Although our results were better now, the loss of parts of the information led to a limit-cycle behaviour. We had the nodes continuing with counting towards infinity (the maximal tested time was one hour, the expected time for reaching the consensus was three minutes). In a final version of the algorithm we developed a different point of view on computation accuracy. We used real numbers also for transmitting and in the stopping criteria we included the parameter *Accuracy*. In every iteration we count the absolute value of the difference between $x_i(k)$ and $x_i(k+1)$ If this difference is smaller than the defined *Accuracy* for three times in a row, then the nodes reach a consensus. Although the results on individual nodes may not be completely the same, the maximal possible difference is guaranteed be the parameter *Accuracy*.

This method was tested in hardware experiments with five nodes. The value of *Accuracy* was set to 0, 0,1, 0,01, 0,001 and 0,0001. In the case when the value of *Accuracy* equals to 0, the consensus was reached when the difference between $x_i(k)$ and $x_i(k+1)$ was zero for three times in a row. Results are shown in Fig. 7. In the case with *Accuracy* set to 0 we used the maximal possible precision of the *float* number, labelled as *Float* in the graph. Due to the *Accuracy* parameter, a smaller number of iterations was required to reach the consensus in comparison with our previous versions without this parameter. With the accuracy set to 0,1 we saved more than 50% of the iterations in comparison to the accuracy set to 0,0001. As a consequence this means also saving half of the energy required for reaching the consensus with only a small effect on the average accuracy. For such devices as nodes with strictly limited energy sources, this should greatly increase the lifetime of nodes (see also [16] for energy consumption). Using integers for initializing the algorithm shows an equivalent behaviour to using float numbers with the *Accuracy* set to 0,00001. Another benefit of defining the accuracy is the achieved resistance to limit cycle behaviour in steady-state. While the algorithm showed such behaviour in sporadic cases, it was never experienced once the accuracy was predefined. As a conclusion, defining computation accuracy seems to be very useful for algorithmic implementations in WSNs.

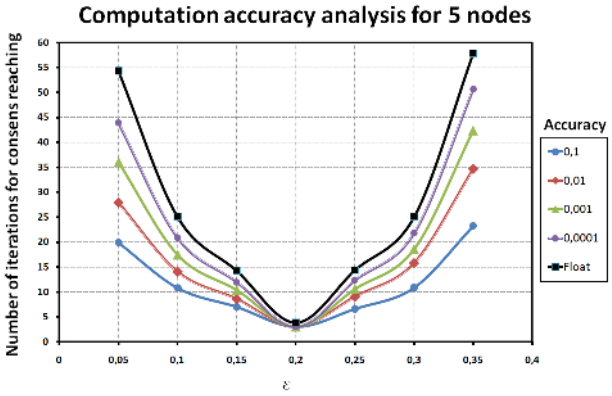

Fig. 7. Computation accuracy analysis as a function of the mixing parameter.

F. Synchronisation

Keeping temporal synchronisation is very important, because nodes coexist without collisions only as long as they remain synchronised. Due to the limited computing abilities of nodes it is important to know how long they remain synchronised. In other words, how many cycles can the algorithms perform without loosing the collision free environment. Experimental results are shown in Fig. 8.

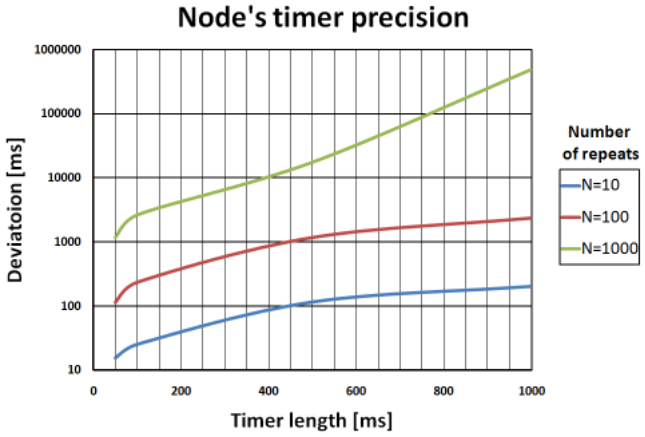

Fig. 8. Timer's precision analysis.

The deviation in Fig.8 is the difference between the measured time and an ideal value, computed as *NxTimer_length*. The node started the periodically repeated timer and the accumulated time of the execution of all proposed timer repeats was taken as the measured time. The blue line represents the deviation of timers of various lengths repeated 10 times, the red is for 100 times and green for 1000 times. From the graphs it is evident that the timer accuracy decreases with an increasing number of repetitions and timer length. With respect to these results, the time between two transmissions is set to 100ms. That is a much higher value than it is required

for avoiding a collision. Recall that we selected a time slot interval of 100ms as illustrated in Fig. 6. The increase of the time between transmissions results in increasing distances between messages sent in the sequence. If the timer would be very precise, only a very small difference between individual messages is needed to avoid collisions. Due to the timer limitations, the length of the sequence is selected larger as in a larger sequence every node has more time available for the collision free transmission. In other words, the width of the node's time slot was increased. The wider time slot provides more tolerance to the node's timer inaccuracy. When the time between transmissions is set to 100ms, a collision occurs only when the timer counts 200ms instead of 100ms, what is highly unlikely.

VI. Link error analysis

The analysis of a wireless link between nodes is a very important part of research in the area of WSNs. The reliability of the link between nodes greatly affects the performance of WSNs, requiring realistic models [17]. Sophisticated models are beyond the scope of this paper, but due to the importance of the link reliability we included a simple model and analysed it. Instead of using the matrix A from (1) as proposed in Section II, which is describing a fully connected network with perfect links, we randomly generate the entries of matrix A in every iteration resulting in $A_{Rand}(k)$. The matrix $A_{Rand}(k)$ for N nodes is stated as follows:

$$A_{Rand}(k) = \begin{bmatrix} 1 & a_{12}(k) & \dots & a_{1N}(k) \\ a_{21}(k) & 1 & \dots & a_{2N}(k) \\ \vdots & \dots & \ddots & \vdots \\ a_{N1}(k) & \dots & a_{NN-1}(k) & 1 \end{bmatrix}$$

The elements $a_{ij}(k)$ are randomly generated in the k-th iteration with respect to the probabilistic values $p(a_ij(k))$. Since $a_{ij}(k) \in 0, 1$, we have $p(a_ij(k) \mid 1) = 1 - p(a_{ij}(k) \mid 0)$. If the probability $P = 0,9$ that a link is 1 then the probability of receiving the message equals 90%. Or in other words, the node misses 10% of messages. Such assumption of link errors is very simplified in comparison with real life wireless links and their time-varying characteristics [18]. But as mentioned before our main focus is not determining the link characteristics. Our main focus is only illustrating the relation between the algorithm performance and the quality of the links in the network.

A. Simulation results

According to the model presented in the previous paragraph, we tested the algorithm performance in relation to different values of P. The results presented in Fig. 9 are achieved in a network consisting of 10 nodes. Results shown on Fig. 9 are illustrating the algorithm performance for$P \in \langle 0,1;1 \rangle$. The case with $P = 1$ is the ideal case, when all messages are received and it is illustrated for comparison. For $P \in \langle 0,4;1 \rangle$ it is possible to obtain, that for $\varepsilon \leq 1/N$ the number of iterations needed for reaching the consensus is increasing with an increasing number of missed messages. This result is quite intuitive - due to missing messages the number of required iterations increases because in every iteration less messages are received than in the ideal case and with an increasing number of missing messages, the number of iterations is increasing as well. For $\varepsilon \geq 1/N$ the situation is different. The shape of the curves is different than in the ideal case and the number of iterations is decreasing instead of the expected increase.

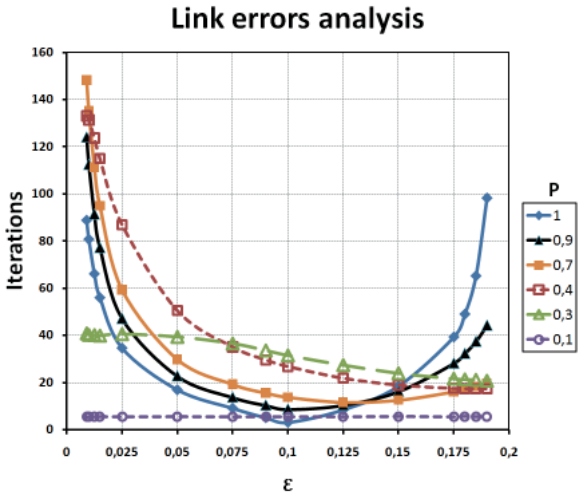

Fig. 9. Link error analysis.

The results from Fig. 9 achieved for $P \in \langle 0,1;4 \rangle$ are different from results gained for $P \in \langle 0,4;1 \rangle$. It is clear that for $P < 0,4$ the shape of the curves is different. Due to the increasing number of missing packets, the network is not able to work properly. When the node misses three messages in a row, it simply reaches the consensus while its value of $x_i(k)$ remains the same for three iterations (due to our test). With increasing number of missing packets, such situation occurs with an increased probability.

While the algorithm leads to reaching the consensus even in the case with a high number of lost packets, our results need to be analysed also from a qualitative point of view. To quantify the quality of the gained result, we compared the gained results with the expected result. The expected result is the average of the initial values labeled as y_{exp} and stated as follows:

$$y_{exp} = \frac{1}{N} \sum_{i=0}^{N} x_i(0). \tag{4}$$

Thus, for the determination of the computational precision the comparison between computed average of every single node and the average of the initial values is used. To quantize the computation error of the single node $n(i)$ we use the following formula to describe a relative error error:

$$er_i = \frac{|y_{exp} - y_i|}{y_{exp}} * 100[\%]. \tag{5}$$

The value y_i is the gained result of the node n_i. To quantize the computation error of the whole network, the average value of computation errors of single nodes is aggregated as follows:

$$E = \frac{1}{N}\sum_{i=0}^{N} er_i[\%]. \tag{6}$$

The relation between E and P is shown in Fig. 11.

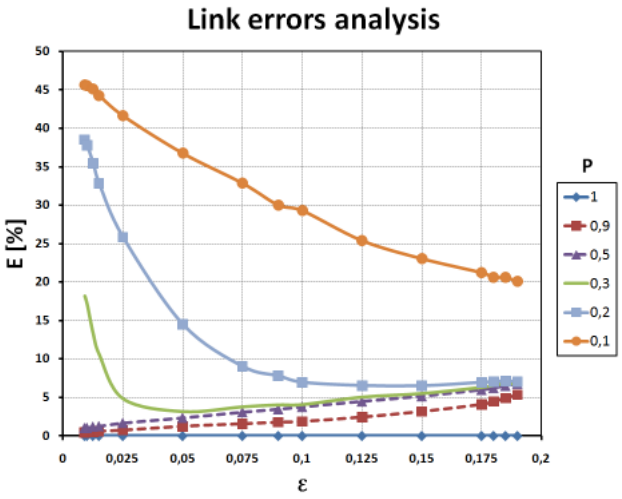

Fig. 10. Relation between E and P.

The dependence between E and ε is greatly affected by the value of P. For $P > 0,3$, E is increasing with an increased value of ε. For larger ε, it is obvious that lost messages affected the result more than in the case with smaller ε. From (1) it is clear, that for the larger ε the value of $x_i(k+1)$ depends more on the received values than in the case with the smaller ε. For $P < 0,3$ the relation between E and ε is reversed. For smaller ε is the value of E very high. For very small P, only small parts of messages are received. For small ε the value of y_i is closely related to the value of $x_i(0)$, while receiving the message changes the value $x_i(k)$ only very slightly. For larger ε the change caused by the received message is more evident, as also the value of y_i is closer to y_{exp}.

VII. Results

Eventually the implementation was tested in an experimental scenario. The average consensus was implemented on real nodes, as described in Section III. Following the description from the previous sections, nodes were deployed in a small room and created a fully connected network. The scenario was focused on imitating ideal conditions. Nodes were close to each other without obstacles affecting the link quality and they were able to receive all messages without the need for retransmissions. None of the nodes failed during the experiments. We also deployed a so called *sniffing node*, responsible for the monitoring of messages sent between nodes. The *sniffing node* provides a connection with a PC and displays the messages on the screen in the application *XSniffer*.

Experiments were focused on the comparison to theoretical knowledge of the average consensus learning. The main interest was testing the algorithm with a mixing parameter set of typical values for which convergence would occur in a small number of iterations, maximal 160 iterations. The algorithm was implemented on 5, 7 and 10 nodes and results are shown in Fig. 12. The achieved results are close to the theoretical results obtained from simulations. Compare to Figure 1. The most evident difference is in the *threshold value* point, where simulation results show only one iteration while results gained from the experiment show three iterations for every tested node's number. This is a consequence from the stopping condition mentioned in Section IV and is shown on Fig. 3 in the formula A: nodes reach the consensus, when the difference between $x_i(k)$ and $x_i(k+1)$ is three times in a row smaller than the *Accuracy parameter*. Although the average was computed correctly in the first iteration, two additional iterations were needed to satisfy our consensus condition. In other operating points, the gained results match very well the theoretical knowledge.

The implementation also handles an increased number N of participating nodes without affecting the results. On the graphs in Fig. 12 we highlighted a few "points of interest" bound to multiples of the threshold value of ($\varepsilon = 1/N$). In these points it is clear that the number of iterations remains almost the same for all three tested node numbers.From the gained results we are also able to conclude, that in our experimental setup are thanks to TDM collision free scheme not present errors on the link.

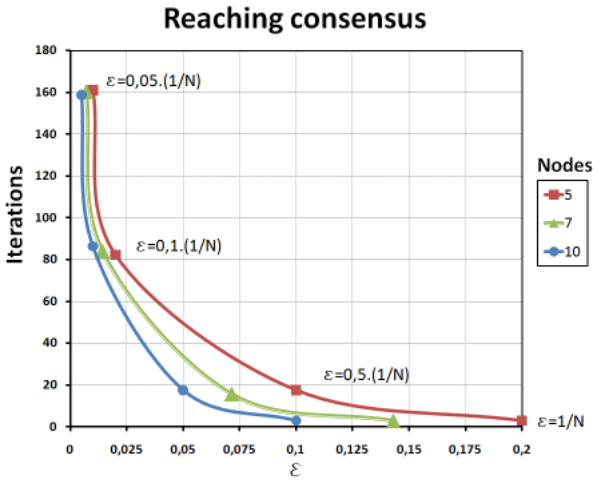

Fig. 11. Experimental scenario.

In previous results, shown on Fig. 7 and 12, an average number of iterations is used. The previously mentioned experiments were repeated many times, so that the average was counted from a large number of results. However, the number of iterations in an algorithm cycle (a single execution of the average consensus algorithm- from the initialization to reaching the consensus) is not constant, but it varies due to the

different initial values. If the standard deviation of the initial values was smaller, the number of iterations needed to reach a consensus was also smaller. This property was tested in the experimental scenario. Five nodes with the *Accuracy* set to 0,1 were initialized with different sets of initializing values. In some of these sets the initialized values were closely related and the difference between them was small, in others the initialized values were more different. The observed difference between the smallest and the highest number of iterations as well as the average number is shown in Fig. 13.

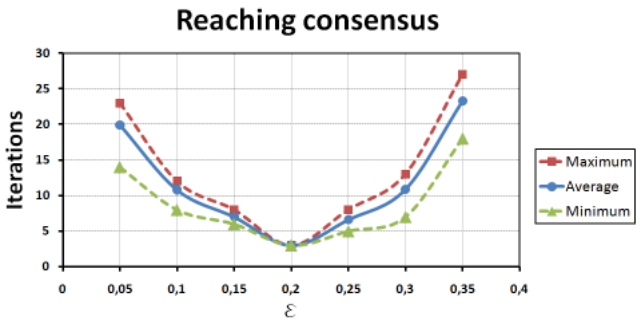

Fig. 12. Algorithm cycle analysis for N=5.

Experimental results in Fig. 13 show an increase of the difference between the minimal and maximal number of iterations with the number of iterations. For the mixing parameter set to the *threshold value* there is no difference, but with different mixing parameters the difference starts to increase due to the increased number of iterations needed for reaching the consensus.

VIII. Conclusion

The average consensus algorithm was successfully implemented and tested on the WSN hardware. After modifying the protocol to make it collision free, the gained results are very consistent with theoretical predictions. The implementation is also a very good starting point for the implementation of a variety of distributed algorithms. Experimental knowledge allows to implement more sophisticated distributed algorithms and the results may serve as a reference base for future implementations. Also the average consensus implementation offers many topics for future work, e.g. creating a multi-hop version for covering larger distances, creating a low-power optimized version or using the average consensus as a part of a larger project, for example [7,8]. In particular the collision aspect has not received sufficient attention. Rather than forcing a synchronous update, an asynchronous update is much more feasible [19] and needs more research.

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] 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.

[3] O.Sluciak, T.Hilaire, and M.Rupp: "A General Formalism for the Analysis of Distributed Algorithms," IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2010), Dallas (TX), USA, March 2010, pp. 2890 - 2893.

[4] O.Sluciak and M.Rupp "Steady-State Analysis of a Quantized Average Consensus Algorithm Using State-Space Description," Proc. of Eusipco, Aalborg, Aug. 2010.

[5] C.Reyes, T.Hilaire, and C.Mecklenbruker, "Distributed Projection Approximation Subspace Tracking based on Consensus Propagation," IEEE 3rd International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP 2009), Aruba, Netherlands Antilles, Dec. 2009.

[6] C.Reyes, T.Hilaire, S.Paul, and C.F.Mecklenbruker, "Evaluation of the Root Mean Square Error Performance of the PAST-Consensus Algorithm," Proceedings of workshop on smart antennas (WSA), Bremen, Feb. 2010.

[7] O.Hlinka, O.Sluciak, F.Hlawatsch, P.M.Djuric, and M.Rupp, "Likelihood Consensus: Principles and Application to Distributed Particle Filtering," Asilomar Conference on Signals, Systems, and Computers, Pacific Grove (CA), USA, Nov. 2010.

[8] O.Hlinka, O.Sluciak, F.Hlawatsch, P.M.Djuric, and M.Rupp, "Distributed Gaussian particle filtering using likelihood consensus," Proc. of ICASSP 2011, Czech Republic, May 2011.

[9] H.Park and M.B.Srivastava, J.Burke, "Design and Implementation of a Wireless Sensor Network for Intelligent Light Control," Proc. IPSN 07, Cambridge, MA, USA, 2007.

[10] S.N.Pakzad, G.L.Fenves, S.Kim, and D.E.Culler, "Design and Implementation of Scalable Wireless Sensor Network for Structural Monitoring, " Journal of infrastructure systems ASCE, pp. 89-101, March 2008.

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

[12] 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

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

[14] M.Bertocco, G.Gamba, and A.Sona, "Is CSMA/CA really efficient against interference in aWireless Control System? An experimental answer", in Emerging Technologies and Factory Automation, 2008. ETFA 2008, ISBN 978-1-4244-1505-2

[15] A.Censi and R.M.Murray, "Real-valued average consensus over noisy quantized channels". In Proceedings of the American Control Conference (ACC), 2009, pp. 4361-4366.

[16] S.L.Howard, C.Schlegel, and K.Iniewski " Error Control Coding in Low-PowerWireless Sensor Networks: When Is ECC Energy-Efficient?" EURASIP Journal on Wireless Communications and Networking, vol. 2006 (2), pp. 29-29, April 2006.

[17] M.Zuniga and B.Krishnamachari, "Analyzing the transitional region in low power wireless links," First Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks, 2004. , pp. 517- 526, 4-7 Oct. 2004

[18] S.Glisic and J.Vikstedt, "Effect of wireless link characteristics on packet-level QoS in CDMA/CSMA networks," IEEE Journal on Selected Areas in Communications, , vol.16, no.6, pp.875-889, Aug 1998

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