8. Fachgespräch
Sensornetze

der GI/ITG Fachgruppe
"Kommunikation und Verteilte Systeme"

TU Hamburg-Harburg
Institut für Telematik (Hrsg.)
http://www.ti5.tu-harburg.de
Technischer Bericht
urn:nbn:de:gbv:830-tubdok-5812

13. & 14.
August
2009

Institut für Telematik
Technische Universität Hamburg-Harburg
TUHH

Inhaltsverzeichnis: Tag 1

Session 1

Session 2

Poster Session

Session 3

Inhaltsverzeichnis: Tag 2

Session 4

Session 5

Session 6

Sensor Network Support for Real-time Indoor Localization of Four-rotor Flying Robots

Juergen Eckert, Falko Dressler and Reinhard German
Computer Networks and Communication Systems
Department of Computer Science 7, University of Erlangen, Germany
{juergen.eckert, dressler, german}@informatik.uni-erlangen.de

***Abstract*—We present a sensor network based indoor localization system that uses ultrasonic distance measurements for real-time localization of flying four-rotor robots. Such quadrocopters are on-board sensor controlled systems. They are very sensitive to lateral drifts, which cannot be compensated by mounted sensors. In our work, we provide a framework for time-of-flight based localization systems relying on ultrasonic sensors. It is optimized for use in sensor nodes with low computational power and limited memory. Nevertheless, it offers scalability and high accuracy even with erroneous measurements. We implemented the system in our lab using ultrasound sensor that are light enough to be carried around by the flying object. Using this real-time localization system, a position controller can be implemented to maintain a given position or course.**

I. Introduction

Flying four-rotor robots are similar to helicopters. In contrast to mono-rotor systems, these so-called quadrocopters usually provide more sensors and more robust controllers. A combination of gyrometers and acceleration sensors is used to determine its current state. Based on these measurements, a digital controller continuously adjusts the orientation of the platform. In such a way devices can easily be piloted by other digital systems such as a sensor network. By only controlling the pitch and the roll angles, the current position cannot be obtained. The quadrocopter always hovers on top of an air cushion. Thus, any minimal measurement error or any airflow may cause a drift to a random direction. The system remains highly in-stable w.r.t. position maintenance. Angle corrections must be permanently applied in addition to the board instruments to keep the flying robot in position.

Figure 1 shows the scenario. A quadrocopter is relying on an external positioning system to continuously update its system parameters. In general, there are many cases in which applications benefit from getting more accurate positioning information. A discussion of preferences for systems using active or passive mobile devices can be found in [1]. If privacy is an issue, passive localization systems should be preferred. For example, the infrastructure of the Cricket system [2] has no knowledge about the current position of any mobile device. However, this system architecture also has several disadvantages. The accuracy suffers if the mobile device moves during a series of (at least three) measurements. In some cases, e.g. using ultrasound, this is a strong limitation because a set of measurements can take up to several hundred milliseconds.

Therefore, we investigated appropriate real-time localization techniques and came up with a new solution that perfectly meets the needs in this application domain. We implemented a system based on ultrasonic distance measurements that is lightweight and can be carried by our quardrocopter. In summary, we not only provide a framework for our chosen scenario but also for other cases of real-time indoor localization. More detailed information can be found in [3].

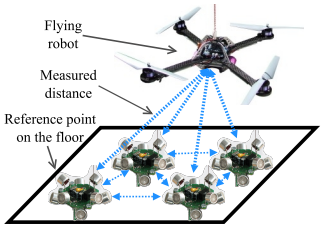


Fig. 1. Four-rotor flying robot hovers over reference points

II. Mathematical Procedure

This section covers the procedure of computing position information out of gathered distance measurements. We rely on ultrasonic distance estimation for time-of-flight (TOF) based lateration.

A. Preliminarities

We assume to start with a set of n tuples T_i, each consisting of a distance d_i to a reference point with a known position and the coordinates of this point $\overrightarrow{x_i}$:

$$T_i = (d_i, \overrightarrow{x_i}) : \overrightarrow{x_i} = (x_i, y_i, z_i)^T, i \in [1, n] \qquad (1)$$

The trilateration problem can be solved for the unknown position $\overrightarrow{x} = (x, y, z)^T$ in different ways. Theoretically, the problem can be solved by a closed mathematical expression as shown in Equation 2. However, in practice, it is impossible to solve those n equations at once due to error-prone measurements.

$$(x_i - x)^2 + (y_i - y)^2 + (z_i - z)^2 = d_i^2;\ i \in [1, n] \qquad (2)$$

Several iterative optimization algorithms exist for the problem. For example, Foy [4] uses a Taylor-series estimation. At least for 2-dimensional problems, the method converges to a good solution within a few iterations. Another common approach is the use of an extended Kalman filter [5]. Abroy

and co-workers [6] present such a non-iterative solution, however, with tremendous restrictions in terms of scalability and variability. Exactly three reference points, precisely oriented to each other are required: the coordinates have to be $\overrightarrow{x_1} = (0,0,0)^T$, $\overrightarrow{x_2} = (x_2,0,0)^T$, and $\overrightarrow{x_3} = (x_3,y_3,0)^T$. In order to apply this system to a general case, a coordinate transformation (offset and rotation) would be needed. Because this requires non-negligible computational effort, this method cannot be applied in many scenarios.

B. Position calculation

One common feature of all indoor location systems attracted our attention. Given that all reference points are mounted to the ceiling, the wall, or the floor, they all have one coordinate in common. Let us denote this as the z coordinate. We exploit this information for a closed position calculation.

First, a distribution of all tuples T_i into m subsets S_j to pairs of three different points must be done. The precise subset generation method will be explained later in Section II-C. For the moment, we assume we have m subsets that fulfill the condition that all z coordinates within a subset S_j are equal. Furthermore, it must be defined *a priori* whether the object to be localized is above the selected c_j, i.e. $z \geq c_j$, or below, i.e. $z \leq c_j$. Then, we can compute m possible coordinates for the unknown object out of the m subsets. Using a set of three single equations from (2) and taking the characteristics of each subset S_j into account, we can form a linear equation system:

$$A\overrightarrow{x} = \overrightarrow{b} : A \in \mathbb{R}^{2\times 2}, \overrightarrow{x} \in \mathbb{R}^2, \overrightarrow{b} \in \mathbb{R}^2 \tag{3}$$

This 2-dimensional problem can be solved easily be applying Gaussian elimination. For the computation of the x and y coordinates, only simple arithmetic operations are needed such as addition, subtraction, and multiplication. Those are very basic (and fast) operations, available on low cost micro-controllers. The z coordinate can be generated in two ways. The easiest way is simply to measure it, which is straightforward using an ultrasound system. Alternatively, the already computed values can be inserted in Equation 2, which, however, requires a square root function for the used micro-controller.

C. Subset generation

In theory, one subset S_j, which contains three tuples T_i, would be sufficient for position estimation. However, taking measurement errors into account, more subsets are required.

Casas and co-workers [7] investigated all kinds of ultrasonic measurement errors. They came up with an average rate of measurement failure of $P_{mf} = 30\,\%$. A position estimation can only be successful if at least one correct subset S_j is used for evaluation, where a correct subset corresponds to one that contains only accurate measurements. P_m denotes the probability that none of the chosen subsets is correct. The required number of subsets can be calculated as follows [7]:

$$m = \frac{log(P_m)}{log(1-(1-P_{mf})^3)} \tag{4}$$

Thus, for example, 11 subsets are required if we accept a failure probability of $P_m = 1\,\%$. Furthermore, the authors suggest that Monte Carlo techniques should be applied to randomly pick m subsets. However, more information about the subsets could help to improve the selection. In general, subsets with geometric shapes that minimize the error rate of the position calculation should be preferred (e.g., regular or well-formed triangles). Thus, the basic idea is to generate and, subsequently, to qualify a subset. Afterwards, it can be placed in a sorted list. Finally, the first m elements in this list are then used for the position calculations.

We decided to use a weighted combination of the average measured distances and the covered ground of the three points would be suitable. Both values are important for a well-formed but (mostly) non-regular tetrahedron (3 reference points plus the unknown point). The base area of the figure is a triangle. Usually, this can not be computed very fast because square root or trigonometric functions would be needed. Therefore, we used the cross product $\overrightarrow{n} = \overrightarrow{a} \times \overrightarrow{b}$ (with $\overrightarrow{a} = \overrightarrow{x_2} - \overrightarrow{x_1}$ and $\overrightarrow{b} = \overrightarrow{x_3} - \overrightarrow{x_1}$). Its length directly corresponds to the covered area. The base area is parallel to the x–y plane, so the cross product only contains a z component. This length can therefore be computed very fast.

D. Position estimation

Finally, the m possible positions (stored in $X(k+1) : X \in \mathbb{R}^{3\times m}$) have to be merged to one position $\overrightarrow{x}(k+1)$. The trivial approach would be the calculation the mean of all positions. However, outliers would significantly influence the result. Casas et al. [7] used an approach where a squared residual vector between all measured and all theoretical distances for each subset is computed. By taking the minimum median of the individual elements the influence of the outliers vanishes. Unfortunately, the computational effort for this method increases with the number of reference points and, therefore, is not very scalable.

We incorporated prior knowledge into the position estimator. Casas method [7] provides localization without any state information. However, already collected information could be exploited to gain better localization results. Thus, we split the estimation process into two steps in a similar way like an extended Kalman filter. In the first step, we predict the current position $\overrightarrow{x_p}(k+1)$ using a state vector:

$$\overrightarrow{x_p}(k+1) = \overrightarrow{x}(k) + \Delta\overrightarrow{x}(k,k-1) \cdot r \cdot \kappa(r) \tag{5}$$

$$r = \frac{\Delta t(k+1,k)}{\Delta t(k,k-1)} \tag{6}$$

For this vector, in each step we store the position and the localization time. The second step is slightly different from the original design of the filter. We generate the new position $\overrightarrow{x}(k+1)$ by selecting the nearest computed position to the predicted position out of the set $X(k+1)$.

The more time has elapsed since the last computation in relation to the last interval, the less reliable the prediction gets. The correction function $\kappa()$ in Equation 5 has been designed for compensating this effect. By using the ratio between the

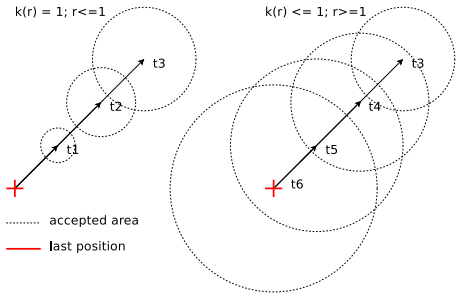


Fig. 2. Position prediction

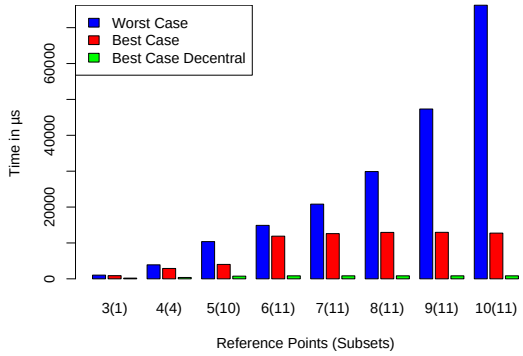


Fig. 3. Total localization time

localization attempts, this mechanism can be automated. Furthermore the absolute computation frequency is not relevant. $\kappa()$ is a function of r (Equation 6), which denotes the ratio of two time intervals. $\kappa()$ is a simple function that returns 1 for values between 0 and 1. For greater values, the output slowly decreases 0. Figure 2 illustrates the prediction vector and the growing space of the position acceptance. As shown on the left side, the prediction vector grows uninfluenced over time if the ratio r is smaller than 1 and, therefore, $\kappa()$ is 1. Thus, $\kappa()$ does not influence the prediction. The right side shows the situation if the ratio r increases beyond 1. This means that the last localization interval (i.e., the time between two accepted positions) was shorter than the elapsed time since the last position was accepted. Now, $\kappa()$ is being decreased because at this time a proper prediction based on the movement during the last interval can not be guaranteed.

III. Localization Performance

Scalability is one of the biggest issues in the context of sensor networks. In order to proof our localization algorithm works even on resource constricted embedded systems, we implemented the system and evaluated it in a lab scenario. In particular, we used the SunSpot sensor node platform [8] running JavaME as the host operating system. One of the key issues is the creation of the subsets. For reasons explained in Section II-C, we limited the number of subsets to 11. Independent of the number of reference points, an upper boundary for the classification can be given. The limitation of subsets implicitly restricts the position vector calculation time to an upper boundary, too. Thus, not every possible position needs to be calculated: Only positions from subsets that meet a certain threshold in the qualification are being considered.

Figure 3 shows the total computation time. The worst case scenario (blue) is a combination of the techniques that are not bounded in computational time. All subsets are computed and the residual based position estimation [7] was applied to the best 11 subsets. In the best case scenario (red), only techniques with a bounded computational time are used. So an upper boundary for the localization algorithm can be given independent of the number of used reference points. This is important to fulfill the real-time specification. The best case decentral scenario (green) describes the absolute minimal computational time consumption for the initiator of the localization, if subset grouping and position vector calculations are distributed on the entire sensor network. Unfortunately, the overhead of the communication latency is far too big to benefit from it, at least using our available hardware.

IV. Test System

Figure 1 shows the latest version of our ultrasonic measurement system including the sensor nodes. Despite the classical master-slave topology, we decided for a hybrid measurement architecture. Whether a device is master (transmitter) or slave (receiver) is completely hardware independent and can be controlled on application level. The detection field of the system is designed to be a hemisphere. Thus, the reference points on the floor can not only detect the flying object but also each other (this architecture is depicted in Figure 1). This way, it is possible to span up the grid automatically by attaching the reference points on top of mobile robots. Another advantage of a flying active beacon, as mentioned before, is that by sensing the TOF of its own active chirp the altitude of the object can be computed without the help of the localization infrastructure.

For the measurements shown in Figure 4, we placed the four-rotor robot at an arbitrary but fixed position over the detection field (four reference points arranged in a square of $2\,\mathrm{m}$ of edge length). It can be seen that there are four centers of gravity. Each subspace is the region for the computed position of one of the four possible subsets. Within this space, the maximum variance is about $\pm 2\,\mathrm{cm}$. The estimated position is normally confined to one of those regions. But as soon as the used subset is missing, the estimated point jumps to another subspace. The temporary vanishing of a subset can have two main reasons. First, one of the measurements was wrong and, therefore, the position was too far away. Secondly, the wireless communication may be disrupted. The generation of the regions is based on systematic errors of the reference points' positions. In our tests, we ensured an accuracy of about $\pm 3\,\mathrm{cm}$. With increasing deployment accuracy of the reference points, the resulting regions merge into a single one.

V. Communication

For communication we used the on-board radio capabilities of the SunSpot. It contains an IEEE 802.15.4 compatible physical interface (Chipcon CC2420) for wireless communication.

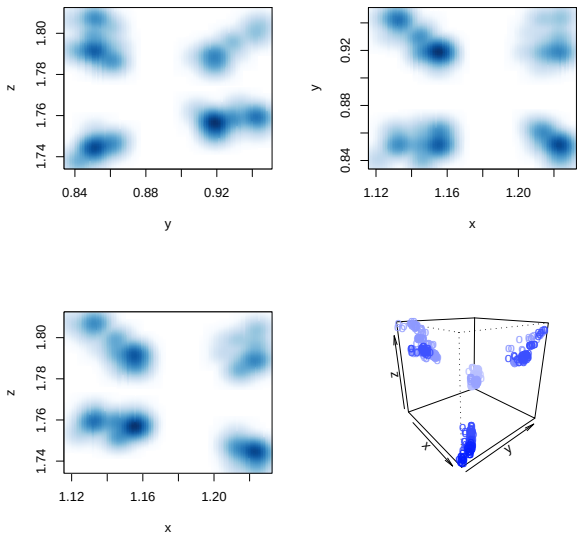


Fig. 4. Localization accuracy: the quadrocopter is fixed in a position, the measured x, y, z coordinates are plotted

Due to its low power specifications the emitted signal is not very strong and therefore the range is limited as well.

After performing a measurement, the measured values are distributed on the sensor network. These data have to be transmitted back to the active beacon for location computation. Our first attempt was to dispatch a request and to have all sensor nodes with relevant data reply to this using the Aloha technique [9] (using broadcast). But in our environment shown above the probability of getting three or more measurements responses were under $65\,\%$. Relying on unicast packets is not feasible, because if a packet can not be delivered, the whole transmission unit can be blocked for more than $10\,\mathrm{s}$. The impact on the flying object of course would be dramatic. The system would not be real-time capable any more. Therefore we developed a custom agent user-level protocol based on broadcasts. The initial frame is broadcasted by the active beacon without any payload. One field of the frame must contain an identifier. That allows the agent to de- and emerge while hopping over the nodes and collecting the data. Three things can happen, if a node receives an agent frame. If the agent frame is already known, it is simply dropped. If it is new to the system, a departure time is generated depending on its options. Additionally, relevant measurements are added before the departure. If an agent arrives with the same identifier as one in the departure queue (duplicate agent), new information from the more recent frame is copied to the one in the queue. Then, the agent frame is dropped. The data collection is not only possible on the initiator side but also on the rest of the system, if desired. All measured information can be made accessible on every sensor node in range without extra radio load.

For the latest agent protocol the sum of probabilities for three and four reference points per cycle is more than $95\,\%$ (compared to $67\,\%$ for the simple broadcast case). Furthermore, the probability for four reference points has significantly increased from $30\,\%$ for the broadcast case to about $82\,\%$. This allows for a much more precise position estimation.

VI. Conclusion

We investigated the problem of continuous indoor localization for flying autonomous robots. In contrast to ground-based robots, any waiting until position measurements have been completed or taking advantage of additional support systems such as odometry are not possible in this case. Thus, a real-time localization is needed that must also take weight constraints into account.

Considering these requirements, we developed an algorithmic procedure that advances the state of the art in indoor localization by being able to perform real-time localization based on possibly error-prone distance measurements. The basic assumption is that one coordinate of the reference points needs to be equal. Without loss of generality, we set the z coordinates to a constant value. This allows a closed mathematical calculation that is even possible to be performed by low resource sensor nodes. If, however, a coordinate transformation needs to be executed, the localization algorithm suffers from the computational complexity of this transformation. We implemented and evaluated the algorithm in our lab. The results demonstrate the feasibility of the solution. We consider our ultrasound lateration technique a necessary step for completely autonomous operation of flying robots in indoor environments.

References

[1] A. Smith, H. Balakrishnan, M. Goraczko, and N. Priyantha, "Tracking Moving Devices with the Cricket Location System," in *2nd International Conference on Mobile Systems, Applications, and Services (MobiSys 2004)*. Boston, MA: ACM, June 2002, pp. 190–202.

[2] N. B. Priyantha, A. Chakraborty, and H. Balakrishnan, "The Cricket Location-Support System," in *6th ACM International Conference on Mobile Computing and Networking (ACM MobiCom 2000)*. Boston, MA: ACM, August 2000, pp. 32–43.

[3] J. Eckert, F. Dressler, and R. German, "An Indoor Localization Framework for Four-rotor Flying Robots Using Low-power Sensor Nodes," University of Erlangen, Dept. of Computer Science 7, Technical Report 02/09, May 2009.

[4] W. Foy, "Position-Location Solutions by Taylor-Series Estimation," *IEEE Transactions on Aerospace and Electronic Systems*, vol. AES-12, pp. 187–194, March 1976.

[5] L. Kleeman, "Optimal estimation of position and heading for mobile robots usingultrasonic beacons and dead-reckoning," in *IEEE International Conference on Robotics and Automation*, vol. 3, Nice, May 1992, pp. 2582–2587.

[6] M. Abreu, R. Ceres, L. Calderon, M. Jimenez, and P. Gonzalez-de Santos, "Measuring the 3D-position of a walking vehicle using ultrasonic and electromagnetic waves," *Sensors and Actuators: A. Physical*, vol. 75, pp. 131–138, June 1999.

[7] R. Casas, A. Marco, J. Guerrero, and J. Falco, "Robust Estimator for Non-Line-of-Sight Error Mitigation in Indoor Localization," *EURASIP Journal on Applied Signal Processing*, vol. 2006, pp. 1–8, 2006.

[8] R. Smith, "SPOTWorld and the Sun SPOT," in *6th International Conference on Information Processing in Sensor Networks*, Cambridge, April 2007, pp. 565–566.

[9] N. Abramson, "THE ALOHA SYSTEM: another alternative for computer communications," in *AFIPS Joint Computer Conferences*. Houston: ACM, November 1970, pp. 281–285.

Distance-Based Distributed Multihop Localization in Mobile Wireless Sensor Networks

Heiko Will, Norman Dziengel, Jochen Schiller
Department of Mathematics and Computer Science
Freie Universität Berlin
Takustr. 9, 14195 Berlin, Germany
Email: {will,dziengel,schiller}@inf.fu-berlin.de

***Abstract*—Localization in wireless sensor networks has been a big challenge for researchers in the past years. Besides physical problems like estimating the distance between two nodes, applicable algorithms are still on the list of open research issues. While single-hop localization with direct connection to fixed anchors is well researched, the localization with fixed anchors over a multi-hop route is still at its beginning. Especially the combination of multi-hop networks and mobile nodes needs further research. In this paper, we present and discuss a refined algorithm and a simulation-based approach regarding the mentioned scenario. Using a five phase structure that pursues a greedy approach, including a refining anchor selection, we investigate and discuss the precision of localization in a mobile environment. To approximate the distance between node and anchor over a multi-hop route, we make use of the mentioned greedy algorithm which fits best for processing multiple distance measurements between nodes. Furthermore, we evaluate different simulation-based experiments with mobile nodes and multi-hop routes.**

***Index Terms*—wireless sensor networks, lateration, mobile localization, radio runtime measuremnt, indoor localization**

I. Introduction

Nowadays, wireless sensor networks (WSNs) are used for different scenarios. They are installed in smart homes for metering and monitoring of enviromental parameters, they are used for ecological enviroment monitoring and they are used for vital parameter monitoring for rescue forces or military purposes. For some of theses scenarios, not only the collected sensor data itself is of interest but also the exact global or relative position *where* the data has been collected.

Gathering this position has been in research focus for years and different approaches exist. While the localization under open air conditions can be done quite simple with mounting Global Positioning System (GPS) receivers onto the nodes, indoor localization is a much more challenging topic. Especially for nodes attached to humans who can freely move inside unknown buildings several problems have to be solved. We need nodes in the network which already know their position and we need a technique to approximate the distance between nodes in order to estimate the position of a sensor node. Immobile nodes with a priori knowledge concerning their postion, are called anchors in this paper. All none anchor nodes are simply called nodes in this paper; they have to calculate their positions using the sensor network localization algorithm distributively.

The major problem of localization in mobile multi-hop networks is the volatile position of nodes while they are in the process of localization, especially when attached to humans or vehicles. This leads to the threefold challenge of our scenario. First, using mobile nodes implies the problem that not all nodes are in range with an anchor during the whole operation. Second, we are not able to calculate a precise position while the node is moving, because the node may continue movement between the needed distance measurements which are called rangings in this paper. The third problem results from problem one and two: To do a localization without having a direct connection to the anchors, the node has to use its neighbors which already could have changed their positions after their last localization.

In this paper, we purpose an algorithm based on a new five phase structure that is applicable for a range-based localization approach in mobile WSNs. The algorithm is usable for multi-hop, infrastructureless WSN deployments and for varying node to node ranging approaches. We introduce a simulation environment which allows the evaluation of current ranging technologies concerning their operational capability and the mentioned problems of mobility and latency during localization. The simulator allows to view the whole network in realtime with a graphical 3D interface.

The main contribution from this paper is twofold:

- We present a distributed multi stage algorithm for indoor localization of mobile sensor nodes and discuss its performance in different scenarios.
- We present a simulation environment to simulate distributed mobile WSN localization algorithms with different parameters and different scenarios in realtime.

Our designated goal is to implement our simulated and evaluated architecture in a real world scenario that inherits the mentioned restrictions. In our current work, we plan to make use of the MSB-A2 sensor node [1] with a 32-bit ARM7TDMI-S based microcontroller in conjunction with chirp technology based NanoLOC modules [2], which use distance calculation via radio signal runtime measurements for the process of ranging.

In Section II we introduce related work, Section III introduces lateration, Section IV presents our localization algorithm and Section V gives an evaluation of our simulated results. We present a conclusion and discuss future work in Section VI.

II. Related Work

Localization of nodes in WSNs can be split up into two parts: First, the process of distance estimation or measurement and second, the localization algorithm. There are different approaches for estimating the distance between a node and its neighbors or fixed anchors. Some techniques rely on the calculation of these distances with physical measurements like radio signal runtime, ultrasonic based-measurements or received signal strength indication (RSSI) measurements. Others try to approximate the distance with a hop-count indicator. For mobile WSNs only processes come into consideration which are applicable with a minimum of infrastructure. We focus on methods which can deliver a node to node distance without any infrastructure besides the nodes themselves. [3]

If the distances between nodes are known, there are several aproaches to calculate the position of a node. If the network is only single-hop and the nodes have a direct connection to anchors which know their position (e.g. from GPS), the approach is simply to do a lateration if enough anchors are in range. In multi-hop networks the position can be calculated centrally or distributively [3]. Our approach is to use a distributed approach, because it needs less infrastructure and less network traffic. Nodes knowing their position are able to use this information for additional possibilities like geo-routing.

Comparing to the current state of the art, our approach takes advantage of network dynamics and chaotic node distribution. Moreover, we desire a solution that handles the lateration problem while contact to anchors is not available. The nodes to be localized are allowed to move at different speeds and may change their orientation whenever they want.

Because of the open and previously mentioned problems, typical evaluations in localization scenarios do not focus on the node mobility. The exemplary paper of Langendoen [4] is comparing three localization systems in static WSN scenarios. Their comparison includes a lateration algorithm and concludes that all algorithm do share a common structure while none performs best.

In contrast to single-hop environments, a multi-hop environment has to minimize the error accumulation that may appear in the network as suggested by Savvides in [5]. They optimized the localization by preventing the error accumulation with the usage [3] of a recursive position refinement, in which they did not take node mobility into consideration, as we are going to.

DV-Distance [6] uses rangings to determine the distance to a neighbor and uses lateration over the anchors just as we do. Based on their DV-Hop-Algorithm, they use the first route established to an anchor through the WSN, to establish accumulated distances. This route represents a suboptimal distance between unknown node and anchor. In contrast to this, our approach is to find a shorter route, using the greedy algorithm. Moreover, we evaluate our system with different anchor selection algorithms.

By using the taxonomy of [7], our approach falls into the category of communication based localization and tracking. In order to track or localize a person while moving, this person needs to wear a sensor node with an unique identifier. The environment itself communicates with the nodes by additional anchors with a priori knowledge of their positition as suggested in [4]. The position of a node has to be investigated by rangings between the nodes, possibly worn by humans, and anchors, in order to provide the basis for a lateration-based localization algorithm.

III. Quad Lateration

Our scenario deals with nodes moving around freely, in other words the localization has to cover all three dimensions. In order to calculate a position of a node in a room, we use the principle of lateration which is a well-established technique to approximate the position out of rangings as introduced in [8]. Each ranging describes with its distance the estimated radius to the position of the unallocated node. If we draw a circle around the appropriate measuring node, by using the ranging as the radius, multiple rangings will intersect themselves and define an area where to expect the unlocated node. If 2D-lateration obtains less than three rangings or 3D-lateration less than four rangings, uniquely defined results are impossible due to the problem of flip ambiguity as defined in [9]. To guarantee uniquely defined results due to the 3D lateration algorithm, we specify that exact four rangings are necessary. It is possible to laterate with more than the specified number of rangings by using an overdetermined system of linear equations, with the goal of optimizing the localization. As our mobile scenario has to deal with the problem of undersupplyed number of rangings we abandon to use this approach. Moreover, the approach decreases the speed of the algorithm. Beside this, it is not given that more rangings will result in a better approximation in reality. A more efficient strategy is to find the best rangings out of a given set. Adapted from [8], we define our lateration by the following equations: Where (x_k, y_k), $k \in [1, ..4]$ represent positions of anchors, (x_u, y_u) define unknown positions of the node while the rangings are defined by $r_u, u \in [1, ..4]$:

$$(x_k - x_u)^2 + (y_k - y_u)^2 + (z_k - z_u)^2 = r_k^2, k = 1..4 \quad (1)$$

To solve the system of equations, we transposed in into a general matrix form:

$$x = A^{-1} * b \quad (2)$$

Where

$$A = 2 \begin{bmatrix} x_1 - x_4 & \cdots & z_1 - z_4 \\ \vdots & \ddots & \vdots \\ x_1 - x_2 & \cdots & z_1 - z_2 \end{bmatrix} \quad (3)$$

and

$$b = \begin{bmatrix} (r_4^2 - r_1^2) - (x_4^2 - x_1^2) - (y_4^2 - y_1^2) - (z_4^2 - z_1^2) \\ \vdots \\ (r_2^2 - r_1^2) - (x_2^2 - x_1^2) - (y_4^2 - y_1^2) - (z_2^2 - z_1^2) \end{bmatrix} \quad (4)$$

In order to come closer to a real world environment, the simulation inserts randomly calculated ranging errors within a

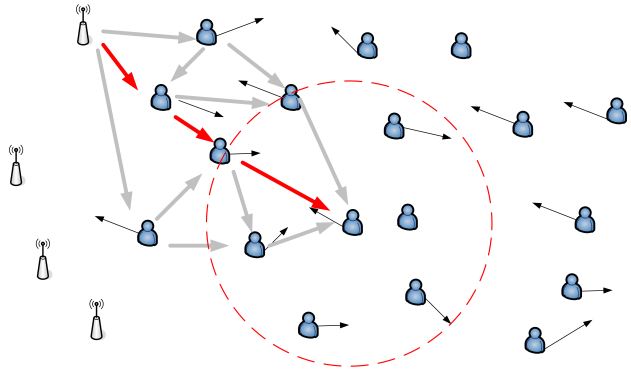


Fig. 1. Choosing the shortest path as anchor distance

fixed range as we desribe in Section V. As described in [3], it is important to interpret effects rising by noisy rangings, hence we need to understand the error propagation characteristics occurring by error-prone range measurements.

IV. Localization Algorithm

Our algorithm reuses the well established three phase architecture as introduced in [4] and extends it to a five phase structure. The five phases are described as follows:

- Phase 1 **aquire neighbor distances -** In this phase a node estimates the distances to all its neighbors with a distance measurement technique. Also, a node has to get the distances from its neighbors to the anchor nodes in this phase. In case of radio runtime measurement this could be done with one radio transmission.
- Phase 2 **calculate anchor distances -** With the aquired data, the node calculates the total distances to all anchors. Probably there will be more than one path to a single anchor which is refined in the next phase.
- Phase 3 **greedy phase -** The node iterates over all distances for each anchor and choses the shortest one and stores it. In all radio runtime based measuremnt systems the calculated distance will be too long and not too short because of reflections and multipath effects. So this simple greedy algorithm chooses the best possible path to the anchor as shown in Figure 1.
- Phase 4 **anchor selection -** The anchor selection phase allows us to choose between different sets of anchor quadruplets; hence the anchor selection is only accessible if more than four different anchors are reachable. These algorithms are exchangeable and we evaluate three anchor selection algorithms afterwards.
- Phase 5 **lateration -** The last phase quad laterates the position of the initiating node with help of the shortest range path to four suitable anchors.

A. Algorithm description - Volume

By using four anchor positions in the $\mathcal{R}^3$, they describe a tetrahedron. Likewise GPS needs to select appropriate satellites, in order be able to locate the unknown position of a node as precise as possible. One simple approach is to use the satellites forming the tetrahedron with the the biggest volume [10]. Inspired by GPS, we adapt their approach and make use of non-collinear anchors forming a tetrahedron with the maximum volume.

B. Algorithm description - nearest Neighbor

Another quite simple approach is to use the four nearest but non-collinear anchors as a lateration basement. This approach is often used in lateration environments and performs well in [4] and [11], moreover, it reduces the possibility to have an obstacle between node and anchor [12].

C. Algorithm description - brute force selection

Our last algorithm is a brute force algorithm which is only deployable in simulators. This algorithm shows us the theoretically possible precision that could be reached by a lateration based system with an optimized anchor selection. This algorithm simply tries all possible anchor permutations and calculates the relative lateration error and then choses the best permutation.

V. Evaluation

A. Simulation Environment

Due to the lack of well introduced localization simulators for mobile nodes in ad hoc networks, we developed a simulation framework for this purpose. To gain a better understanding of the complex scenarios, we designed a real-time graphical simulation enviroment with a 3D visualization component. The simulator can simulate up to 1000 nodes on a regular PC in realtime and an unlimited number of fixed anchors in a 3D area. The nodes have different movement vectors which can be configured or generated randomly. Each node can have a different radio range and is able to request the distances to his neighbors within radio range. The estimated distances can be inaccurate by a certain percentage. The enviroment has an API for applying different localization algorithms for each node. The environment simulates a volatile, manlike node movement and erroneous radio ranges, whereby the communication between nodes is simplified. Especially the media access and radio characteristics like reflection and absorbtion are not simulated.

B. Experimental Setup

For all simulations we used a $100m * 100m$ playfield with a height of $15m$ to simulate a large building. All nodes move with randomly changing directions and movement speeds which are limited to $2m/s$ to simulate humans with normal walking speed. All nodes start moving on the same spot on the border of the playfield at the same time. In every simulation there are four fixed anchors placed outside the playfield but reachable from every node at the beginning of the simulation. The ranging error is set to 3.33%, while the radio range is set to $30m$. These values where choosen because they fit for the real hardware we are going to use in the second stage of our project.

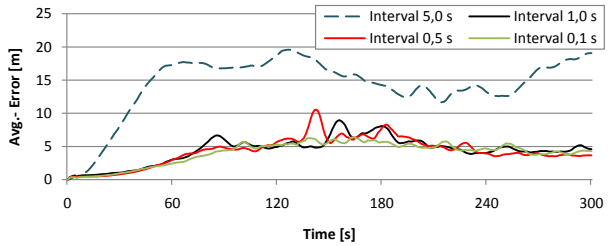


Fig. 2. Comparing average localization error by changing localization intervals. Fixed settings: nearest neighbor anchor selection, 4 anchors and 300 nodes.

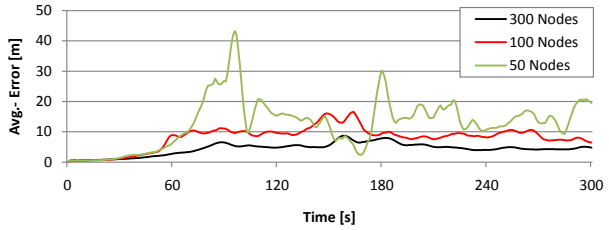


Fig. 3. Comparing average localization error by changing the quantity of nodes. Fixed settings: nearest neighbor anchor selection, 4 anchors and 1 second localization interval.

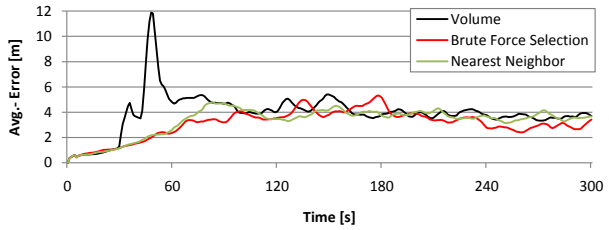


Fig. 4. Comparing average localization error by changing anchor selection algorithms. Fixed settings: 4 anchors, 1 second localization interval and 300 nodes.

C. Results

First, we wanted to analyze if the localization interval has an influence to the average position error which is likely due to the movement speed. But the experiment showed that for the relatively low movement speed of human beings the interval has no big impact which is shown in Figure 2. Only for intervals larger than one second the error begins to increase noticeably. From this observation we conclude that to achieve smaller localization intervals e.g. to save energy, the interval should be compressed so that all localization is done in a one second sub interval of the overall localization interval.

The second experiment analyzed the impact of the network density. As shown in Figure 3, the node count has a much bigger influence than the localization interval on the average position error. A node count below 50 leads to too many nodes having no connection to a sufficent number of neighbors to perform the lateration.

Finally, we wanted to know the effect of the algorithm to choose the anchors. So we placed another four anchors around the area and ran different algorithms to choose the best four of them. Figure 4 shows that our nearest neighbour algorithm is very close to the optimal algorithm in its effect to the average position error. In the diagramm it looks like that it is sometimes even better than the brute force algorithm which can be traced back to the fact, that there are some calculations in the simulator which are affected by randomness e.g. distance error. The volume algorithm performed as good as the nearest neighbor algorithm but because of its more complex implementation we propose the nearest neighbor algorithm as the best algorithm. The increase of the anchors resulted in a noticeable decrease of the average error which should be observed in future work. It is not clear if the decrease is a result of the better position of the chosen anchors or simply an effect of the obeservation that the average hop count from a node to a anchor was also decreased by placing more anchors.

VI. Conclusion & Future Work

We propose our five stage localization algorithm as a next step to achieve precise indoor localization for multi-hop mobile networks. The algorithm performs good in dense networks with an acceptable refresh rate. In future work, we will concentrate on implementig the algorithm on real sensor nodes and evaluating localization precision with a real radio layer. Further, we will research on adding an applicable QoS component that enables a local estimation for the position error.

References

[1] M. Baar, H. Will, B. Blywis, T. Hillebrandt, A. Liers, G. Wittenburg, and J. Schiller, "The scatterweb msb-a2 platform for wireless sensor networks," Freie Universitt Berlin, Tech. Rep., 2008.
[2] N. T. GmbH, "nanoloc trx transceiver (na5tr1)," 2000.
[3] G. Mao, B. Fidan, and B. D. O. Anderson, "Wireless sensor network localization techniques," *Comput. Netw.*, vol. 51, no. 10, pp. 2529–2553, 2007.
[4] K. Langendoen and N. Reijers, "Distributed localization in wireless sensor networks: a quantitative comparison," *Comput. Netw.*, vol. 43, no. 4, pp. 499–518, 2003.
[5] A. Savvides, H. Park, and M. B. Srivastava, "The n-hop multilateration primitive for node localization problems," *Mob. Netw. Appl.*, vol. 8, no. 4, pp. 443–451, 2003.
[6] D. N. And, "Ad hoc positioning system (aps)," 2001.
[7] E. D. Manley, H. Al Nahas, and J. S. Deogun, "Localization and tracking in sensor systems," in *SUTC '06: Proceedings of the IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing - Vol 2 - Workshops*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 237–242.
[8] H. Karl and A. Willig, *Protocols and Architectures for Wireless Sensor Networks*. John Wiley & Sons, 2005.
[9] A. A. Kannan, B. Fidan, and G. Mao, "Robust distributed sensor network localization based on analysis of flip ambiguities." in *GLOBECOM*. IEEE, 2008, pp. 6–11. [Online]. Available: http://dblp.uni-trier.de/db/conf/globecom/globecom2008.html#KannanFM08
[10] S. H. Dong, "A closed-form formula for gps gdop computation," 2007.
[11] P. K. Sahoo, I.-S. Hwang, and S.-Y. Lin, "A distributed localization scheme for wireless sensor networks," in *Mobility '08: Proceedings of the International Conference on Mobile Technology, Applications, and Systems*. New York, NY, USA: ACM, 2008, pp. 1–7.
[12] A. A. Yi Shang, Hongchi Shi, "Performance tradeoffs of localization methods in ad-hoc sensor networks," 2004.

Remote Incremental Adaptation of Sensor Network Applications

Waqaas Munawar, Olaf Landsiedel, Muhammad Hamad Alizai, Klaus Wehrle
Distributed Systems Group
RWTH Aachen
firstname.lastname@rwth-aachen.de

***Abstract*—Wireless Sensor Networks (WSNs) are deployed for long periods of time, during which a need often arises to dynamically reprogram or retask them. An array of solutions has been proposed to this effect, ranging from full image replacement to virtual machines. However, the capabilities of TinyOS – the current state of the art in sensor node operating systems – are still limited to full image replacement. TinyOS based applications have a modular architecture but during compilation this modularity is lost resulting in a statically linked system image.**

In this work we extend TinyOS to allow dynamic exchange of components in WSN applications by conserving their modularity during the compilation process. This generates the possibility of incremental adaptation of sensor nodes' behavior through partial code replacement. The designed system does not require any alterations in the existing user interfaces, remaining transparent to the user. The evaluation shows that our approach imposes almost no performance overhead for loaded application while keeping a smaller memory footprint than other comparable solutions.

I. Introduction

Sensor nodes could be located far from the networked infrastructure and easy human access [1], [2], [3]. Based on the evolving analysis or the environment the software application of the sensor network often requires adaptation through introduction of new code. Manually collecting all the nodes to apply a software update is dangerous in some of the situations [2], [3] and tedious in others [1], [4]. Therefore, remote software reconfiguration – even if a rare activity as compared to the application operations – becomes a highly desirable feature.

Remote retasking of sensor nodes is mainly challenged by three constraints; limited energy, limited processing power and limited available onboard memory. kilobytes. Moreover, the major hurdle in the way of mainstream adoption of WSNs remains the steepness of the associated learning curve. Considering these constraints, an ideal solution to dynamically update a nodes functionality would be the one that optimizes the energy usage, has a reduced memory footprint, and does not require any alteration in the existing user interfaces.

Our contribution in this work is design and implementation of a solution for efficient dynamic adaptation of TinyOS based applications running on sensor nodes. The proposed system works in two phases; firstly, the existing components of an application are solitarily compiled into ELF objects. Solitary generation of the software components ensures that the component structure of the TinyOS application is preserved during the compilation process. In the second step, these components are transferred to the sensor node and integrated into the running application. To allow this, a thin node runtime is designed that includes a runtime linker and enables dynamic exchange of components. Runtime dynamic linking allows the reduction of energy-toll incurred in communication by limiting the size of required communique to that of an updated component only. The presented system is tightly coupled with TinyOS, reusing its components and interfaces hence easing the adoption process. Moreover, the designed system does not necessitate any change in the existing code repository of TinyOS hence, remaining transparent to the user of the system.

II. Related Work

The existing approaches to tackle the issue of retasking a sensor network can be classified into three main areas;

Full image replacement e.g. Xnp [5], Deluge [6] and differential updates [7], offer very fine grained control on the possible reconfigurations but are quite wasteful in terms of energy-cost of communication.

Virtual machine e.g. Maté [8], perform inversely; they optimize the energy-cost of communicating the new functionality but the control offered on the possible reconfigurations is very coarse grained, moreover the trade-offs between interpreting code and executing native binaries suggest the use of the latter for long-running systems.

Dynamic operating systems e.g. Contiki [9], SOS [10] and FiGaRo [11], provide benefits of both of the former categories however, in most cases these solutions have followed a clean slate approach which has hindered the wide scale adoption. Two notable exceptions are FlexCup [12] and TOSthreads [13] that are built on top of seasoned TinyOS repository. FlexCup offers dynamic adaptation for TinyOS based applications but lacks the support for new extensions to nesC and employs nonstandard tools. TOSthreads library and its associated linker follow a polling based approach for kernel to application calls instead of nesC's more suited, event based approach and introduces a new interface for users, rendering it difficult to adopt.

III. Design

TinyOS based applications consist of large number of wired nesC components which communicate with each other via

interfaces. This component based structure results in a very modular architecture of TinyOS applications. However, once compiled, this modularity is lost. The NCC compiler when transforming nesC to C mashes up the component structure of the input to make the output conform to the semantics of C. The output is a single monolithic C source file to be compiled by the respective toolchain. We alter this process of compilation by isolating subsets of an application's constituent nesC components and compiling them into ELF files. The resulting files collectively enclose all of the constituent components of the application. These files are transferred to the sensor node which links them together and loads them in the program memory to form the executable binary image again.

The solution we present consists of two main components; *Isolater* to isolate a single or an integrated group of nesC components and compile them into an ELF object. Second, *TinyMan*, a runtime ELF linker executing at sensor node and responsible for integrating the ELF objects to form the executable binary image to be loaded in program memory. The working of both of these are detailed as follows.

A. Isolater

The *Isolater* functions by compiling parts of a single TinyOS based application separately into ELF files. It executes on PC (host) and utilizes the binary component generation feature of the NCC compiler. This feature was introduced primarily to provide better commercialization support as binary components can be used and distributed without their corresponding source code. We utilize this feature to isolate and compile a single or a set of interconnected nesC components belonging to a TinyOS application. Compiling parts of an application solitarily causes loss of code optimization possibilities as well as introduction of ambiguities which either lead to an incorrect decision by the compiler or result in a compile time error.

The main issues faced during isolation of nesC component are compile-time operators, default events and generic components and interfaces. All of these are caused due to the non-availability of information hidden in those parts of the application that are not being compiled at the moment. *Isolater* provides this missing information using the additional input in the form of a nesC configuration. This additional input consists of two main parts for each component to be isolated; a component-wrapper and an application side place holder. The component wrapper ensures that the component being isolated is provided the required knowledge of the rest of the application for the correct compilation. Likewise, the application side place-holder ensures that the application gets the required knowledge about the component which will be linked in at runtime. During this process, the actual source code of both the application and its component is not changed. This allows complete 'recycling' of existing TinyOS based applications and seamless integration of the system into the existing TinyOS skeleton, thereby remaining totally transparent to the application developer. In the next section we discuss the issues related to dissemination and integration of the generated components.

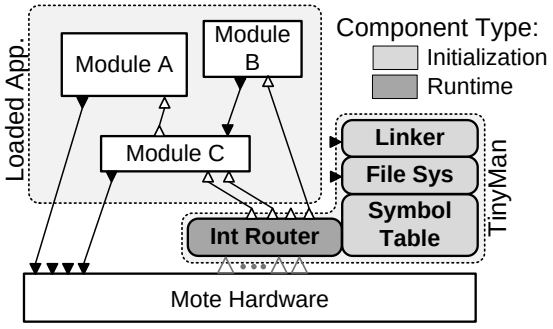


Fig. 1. Architectural elements of *TinyMan*. Only runtime components are active during the normal execution of a loaded application. Linker and File System APIs are provided by the kernel to support the application to kernel calls.

B. TinyMan

After the compilation of components, the next step consists of their dissemination and integration in the sensor application executing on the sensor node. The dissemination in the network takes place through the use of the Deluge data dissemination protocol. Other existing protocols can be employed as well and these, since treated as part of loaded application, can also be replaced remotely on runtime. This design approach makes data dissemination a concurrent process along with the normal execution of loaded application resulting in reduced downtime due to an update in progress.

After the required modules and an update command have been received the data dissemination protocol invokes the linker to integrate the received modules and place the new binary image in program memory of the sensor node. To accomplish this, the node runtime consists of the following main components:

- **File System:** provides the storage capability for large data elements such as received ELF modules.
- **Linker:** responsible for linking the the new ELF modules and placing them in code memory.
- **Global Symbol Table:** As the linking is done among dependant ELF modules, this component holds the symbols offered by one module that are needed by some other ELF module.
- **Interrupt Router:** Unlike the compile-time linker, the implemented runtime linker does not have the flexibility in placement of code segments. Therefore an interrupt router is implemented to route the interrupts to the inappropriately placed interrupt service routines.

These components in relation to a loaded application are shown in Figure 1. Apart from the interrupt router, all of the other components are inactive during the normal execution of application. This helps in minimizing the runtime performance impact due to *TinyMan*. The linker and the file system's APIs

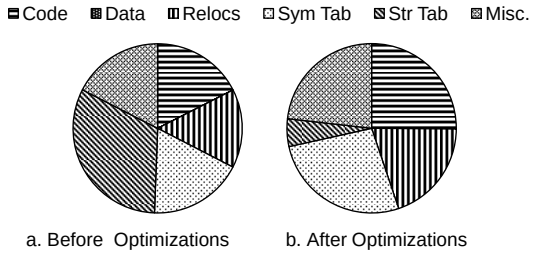


Fig. 2. Results of optimizations shown as proportionate change in sizes of different segments of the resulting ELF file.

are available to the loaded application through the global symbol table, which can be used for storing new or updated ELF modules and then integrating them into the existing application.

C. Optimizations

The ELF format, though a widely used standard, is not optimized for the low power processors. In the ELF libraries, compiled from the NCC compiler's generated code, the bulk of the contribution in size comes from the string table which holds the names of all the symbols in the ELF file. These names often tend to be quite long – about 80 characters each. We decrease the size of the symbol names down to 3 characters by replacing each symbol name with a unique string based on an alphanumeric counter. The mapping of the replaced names is stored in a database which can be used later when recompiling parts of the application. This procedure results in: (1) significant reduction in the size of ELF file, (2) reduction in the size of symbol table and (3) reduction in number of string comparison operations. The average proportionate reduction in the size of string tables for ELF files of the Blink application is shown in Figure 2.

The second set of optimizations that result in significant resource savings is applied to the symbol table which is used during the process of linking. We split the symbol table into two sub-tables; one containing static core symbols and the other filled dynamically from the symbols included within the ELF files being loaded. The static part is created at compile time and placed in ROM in a sorted order allowing binary search among the symbols. This results in a quicker hence more energy efficient linking process.

These two sets of optimizations together cause a major improvement in processing speed, resulting in energy savings of up to 66% when compared to the original ELF.

IV. Evaluation

We evaluate the proposed system along the lines of major constraints faced in WSNs i.e. energy consumption, processing requirements and memory utilization.

A. Energy

To evaluate the per-node energy consumption caused due to a network wide reconfiguration we devise a simple energy model and calibrate it using the readings taken empirically.

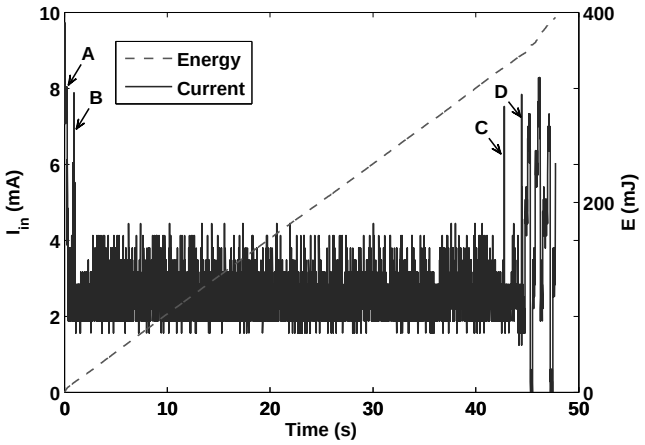


Fig. 3. Current draw and energy utilization during processing and loading of Blink application at telos platform. The peaks are generated by turning the onboard LEDs on simultaneously. A to B: saving the modules in external flash, B to C: linking the modules, C to D loading the modules in program memory, from D onwards: executing the application.

The final results are compared against Deluge [6] – the widely used in-field code replacement tool and protocol.

We model the energy cost of reconfiguration as;

$$E = E_{Tx} + E_{Rx} + E_P$$

Where,

E_{Tx} is the energy consumed in transmitting an update
E_{Rx} is the energy cost of reception, and
E_P is the energy consumed during related processing.

We assume that each node receives the update, propagates it and then processes it to reconfigure itself. This might not be accurate for bordering nodes and those nodes which do not need to propagate further because of their close vicinity to the other nodes in the network, in which case we get an upper bound on the consumption of energy. However, the assumption adapts to reality more closely in a bigger network with a lower node density. In such networks, the transmitters are tuned to transmit at maximum output power due to larger inter node distance. Under this condition, in telos, the current consumption during transmission and reception is almost the same hence, so is the energy consumption. The transmission and reception costs also depend upon the size of the component (S_C) being communicated and the protocol used for communicating it. The overhead introduced by the protocol can be measured as a constant multiplicative factor (K_F) to the size of original data to be communicated. These factors multiplied by the transfer cost of a single bit (K_{BT}) complete the expression for transfer cost of a component. Since E_P involves processing on a single node only, it can be measured empirically as shown in Figure 3. For the Deluge protocol the K_F is 3.35 [6] and a telos node configured with TinyOS consumes 0.0105 mJ per byte for transmission.

So,

$$E = E_{Tx} + E_{Rx} + E_P$$

Component	Size (B)	Transfer Energy (mJ)	Savings Factor
BlinkC	836	58.77	46.6
BlinkAppC	7156	506.92	5.4
LedsC	1600	113.34	24.2
Msp430TimerC	4644	328.97	8.33
Blink w. Deluge	39024	2743.38	–

TABLE I
SAVINGS IN TRANSFER ENERGY DUE TO INCREMENTAL UPDATES

System	ROM (B)	RAM (B)
SOS Core	20464	1163
TinyOS w. Deluge	21132	597
Bombilla VM	39746	3196
TinyMan	15826	792

TABLE II
MEMORY USAGE COMPARISON FOR *TinyMan*

$$\begin{aligned} &\Rightarrow \quad 2E_{Tx} + E_P \\ &\Rightarrow \quad 2(K_F K_{BT} S_C) + E_P \\ &\Rightarrow \quad 0.0703 S_C(mJ) + E_P \end{aligned}$$

We use this model along with the Blink application from the TinyOS repository to estimate the transfer costs of different constituent components of the application. The Blink application is broken down into four components; LedsC, Msp430TimerC, BlinkC and BlinkAppC. In the presented system, any of these components can be individually and remotely modified whereas in Deluge the whole application needs to be replaced. The resultant reduction in energy costs is presented in Table I.

B. Memory Usage

The memory footprint of the presented system is quite moderate in comparison with the popular existing solutions as shown in Table II. On telos rev. B it consumes only 7.7% of RAM and 32% of program memory with rest of the resources available for the loaded application. The external flash memory is completely available for the file system and 'Golden Images'. The optimized memory footprint results from the design approach of keeping the runtime support layer as thin as possible. This optimizes the usage of hardware resources available on the platform, hence leaving more memory space for the loaded application.

C. Performance Overhead

Keeping the runtime support layer thinner has a positive effect on the runtime computational requirements as well. During normal application execution – the most frequent activity for a sensor node – only the interrupt routing component of *TinyMan* is active, and introduces a short delay in the processing of interrupts. On telos platform the worst case delay is that of 23 instruction cycles – equivalent to processing required for copying eight bytes in memory. No performance depreciation is caused by other components and the code execution remains native.

V. CONCLUSION AND FUTURE WORK

In this paper, we presented a fine grained code update mechanism for sensor networks that offers the functionality and performance required for remote adaptation of sensor applications. The presented system is tightly and transparently integrated with TinyOS, resulting in ease of adoption and full utilization of the seasoned TinyOS code repository.

The preliminary evaluation, as presented earlier, provided a proof of concept. In the future we plan a more thorough evaluation with real life applications. Also, some of the steps during compilation need to be automated. Finally we would like to optimize the ELF format further and evaluate the system using other common hardware platforms as well.

REFERENCES

[1] D. Pompili, T. Melodia, and I. F. Akyildiz, "Deployment analysis in underwater acoustic wireless sensor networks," in *WUWNet '06: Proceedings of the first ACM international workshop on Under Water Networks*. Los Angeles, CA, USA: ACM, 2006, pp. 48–55.

[2] P. Juang, H. Oki, Y. Wang, M. Martonosi, L. S. Peh, and D. Rubenstein, "Energy-efficient computing for wildlife tracking: design tradeoffs and early experiences with zebranet," in *ASPLOS-X: In Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications*, ser. 37, no. 10. ACM, October 2002, pp. 96–107.

[3] G. Werner-Allen, K. Lorincz, M. Welsh, O. Marcillo, J. Johnson, M. Ruiz, and J. Lees, "Deploying a wireless sensor network on an active volcano," *IEEE Internet Computing*, vol. 10, no. 2, pp. 18–25, 2006.

[4] R. Szewczyk, E. Osterweil, J. Polastre, M. Hamilton, A. Mainwaring, and D. Estrin, "Habitat monitoring with sensor networks," *Commun. ACM*, vol. 47, no. 6, pp. 34–40, 2004.

[5] J. Jeong, S. Kim, and A. Broad, "Network reprogramming," Aug 12, 2003. [Online]. Available: http://www.tinyos.net/tinyos-1.x/doc/Xnp.pdf

[6] J. W. Hui and D. Culler, "The dynamic behavior of a data dissemination protocol for network programming at scale," in *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*. Baltimore, MD, USA: ACM, 2004, pp. 81–94.

[7] N. Reijers and K. Langendoen, "Efficient code distribution in wireless sensor networks," in *WSNA '03: Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications*. San Diego, CA, USA: ACM, 2003, pp. 60–67.

[8] P. Levis and D. Culler, "Maté: a tiny virtual machine for sensor networks," in *ASPLOS-X: Proceedings of the 10th international conference on Architectural support for programming languages and operating systems*. ACM, 2002, pp. 85–95.

[9] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors," in *LCN '04: Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks*, Washington, DC, USA, 2004, pp. 455–462.

[10] C.-C. Han, R. Kumar, R. Shea, E. Kohler, and M. Srivastava, "A dynamic operating system for sensor nodes," in *MobiSys '05: Proceedigs of third international conference on Mobile Systems, Applications and Services*. Seattle, Washington: ACM, 2005, pp. 163–176.

[11] L. Mottola, G. P. Picco, and A. A. Sheikh, "Figaro: Fine-grained software reconfiguration for wireless sensor networks," in *EWSN '08: Proceedings of the fifth European Workshop on Wireless Sensor Networks*, Bologna, Italy, 2008, pp. 286–304.

[12] P. J. Marrón, M. Gauger, A. Lachenmann, D. Minder, O. Saukh, and K. Rothermel, "Flexcup: A flexible and efficient code update mechanism for sensor networks," in *EWSN '06: Proceedings of the third European Workshop on Wireless Sensor Networks*, 2006, pp. 212–227.

[13] R. Musaloiu-E., C.-J. M. Liang, and A. Terzis, "A Modular Approach for WSN Applications," CS. Dept. Johns Hopkins University, HiNRG, 2008.

Verteilte Abfragebearbeitung von Sensordaten

Dirk Bade
Verteilte Systeme und Informationssysteme
Department Informatik, Universität Hamburg
Vogt-Kölln-Strasse 30, 22527 Hamburg
Email: bade@informatik.uni-hamburg.de

***Zusammenfassung*—Intensive Forschung und Entwicklung im Bereich von Sensornetzen haben Technologien für Hard- und Software ein inzwischen gebrauchstaugliches Niveau erreichen lassen. Trotzdem finden sich Sensornetze heutzutage selten im industriellen sowie privaten Einsatz wieder. Einer der Gründe hierfür sind die technischen Barrieren mit denen sich Nutzer von Sensordaten auseinandersetzen müssen. In dieser Arbeit wird daher eine Middleware-Infrastruktur vorgestellt, die Anbieter und Nutzer von Sensordaten zusammenbringen und eine transparente Abfrage von Sensordaten ermöglichen soll. Durch die Möglichkeit beider Seiten die Verarbeitung von Sensordaten innerhalb der Middleware durch Angabe eines Verarbeitungs-Workflows zu beeinflussen, können einerseits maßgeschneiderte Zugriffe auf Sensordaten gewährt werden. Andererseits trägt dieser Ansatz auch der Endgeräteheterogenität auf beiden Seiten Rechnung, da die Verarbeitungslast in die Middleware verschoben und Ressourcen der (mobilen) Endgeräte geschont werden können.**

I. Einführung

In den vergangenen Jahren führte die stetige Weiterentwicklung von Soft- und Hardware für Sensornetzwerke zu einem Entwicklungsstand, der nicht mehr nur vermehrt universitäre, sondern vor allem auch von der Industrie angetriebene Projekte entstehen ließ. Längst werden Sensoren zur Überwachung von Straßen, Ski- und Erdbebengebieten, intelligenten Häusern, Transportwegen etc. eingesetzt. Doch während sich Forschung und Entwicklung meist auf die intrinsischen Charakteristiken von Sensornetzwerken konzentrierten, bleibt häufig die Frage, was mit den Sensordaten nach ihrer Erhebung passiert, unbeantwortet. Einerseits fehlen geeignete Geschäftsmodelle und andererseits passende Softwareinfrastrukturen, die eine schnelle Umsetzung der Modelle erlauben.

In dieser Arbeit soll daher eine verteilte Middleware-Infrastruktur vorgestellt werden, welche Anbieter (z.B. unabhängige Sensornetzbetreiber) und Nutzer (z.B. Geschäftsanwendungen) von Sensordaten zusammenbringen soll. Anbieter stellen erhobene Sensordaten in nahezu Echtzeit dem System zur Verfügung und Nachfrager können ihr Interesse an bestimmten Daten im System hinterlegen und werden bei Eintreten korrespondierender Ereignisse entsprechend informiert. Das System selbst übernimmt hierbei die Aufgabe zwischen Anbietern und Nachfragenden sowie den von diesen verwendeten Technologien zu vermitteln. Auf diese Weise soll die herrschende Heterogenität verborgen und die vorgestellte Middleware auch für andere Kontextdaten (z.B. RFID, Nutzerkontext etc.) verwendet werden können.

Im folgenden Abschnitt II werden Anwendungsszenarien vorgestellt, anhand derer in Abschnitt III die Architektur und Funktionsweise der Middleware-Infrastruktur präsentiert wird. Abschnitt IV widmet sich dem aktuellen Stand der Implementierung bevor im letzten Abschnitt V eine Zusammenfassung gegeben wird.

II. Anwendungsszenario

Ein Handelsunternehmen hat eine Palette mit TV-Geräten gekauft und deren Verschiffung in einem intelligenten Container veranlasst. Dieser Container ist mit einer Reihe von Sensoren ausgestattet, welche in regelmäßigen Abständen Temperatur-, Feuchtigkeits- und Beschleunigungswerte protokollieren. Sobald der Container an seinem Bestimmungsort ankommt, sollen die Sensor-Logbücher des Containers am Eingang des Lagerhauses ausgelesen werden. Weisen die Sensormessungen daraufhin, dass die Fracht z.B. durch zu hohe Temperaturen oder Beschleunigungen beschädigt sein könnte, soll ein Mechaniker die TV-Geräte noch im Lagerhaus begutachten, bevor sie für die weitere Reise auf einen LKW verladen werden.

In diesem Szenario sind drei verschiedene Akteure beteiligt: 1) das Handelsunternehmen, 2) das Transportunternehmen und 3) das Lagerhaus. Das Handelsunternehmen hat die TV-Geräte eingekauft, Transport und Abwicklung in Auftrag gegeben und ist verantwortlich dafür, dass der letztendliche Käufer die Ware ordnungsgemäß zugestellt bekommt. Hierfür wurde ein Transportunternehmen beauftragt, welches den intelligenten Container stellt und für den sicheren Transport der Ware ins Lagerhaus verantwortlich ist. Das Lagerhaus letztlich soll die TV-Geräte vom Schiff auf einen LKW verladen und das Handelsunternehmen über aktuelle Vorkommnisse informieren. Zu diesem Zweck teilt das Handelsunternehmen dem Lagerhaus mit, dass es informiert werden möchte, sobald a) die Ware eintrifft und Sensorwerte keine Beschädigung vermuten lassen, b) die Ware eintrifft, aber möglicherweise beschädigt ist oder c) die Ware bis zu einem bestimmten Zeitpunkt nicht eingetroffen ist.

Bevor dieses Anwendungsbeispiel in Abschnitt III-A zur Erläuterung der präsentierten Middleware-Architektur wieder aufgegriffen wird, sollen im Folgenden kurz zwei weitere Anwendungsmöglichkeiten skizziert werden.

A. Weitere Anwendungsszenarien

Auf der Intensivstation eines Krankenhauses wird der Gesundheitszustand von Patienten durch eine Reihe von Sensoren überwacht. Um das Pflegepersonal zu entlasten, sollen

die Visiten-Intervalle vergrößert werden. Damit jedoch auch zwischenzeitlich schnell auf Änderungen eines Gesundheitszustands reagiert werden kann, werden individuell für jeden Patienten komplexe Ereignismuster für dessen Sensordaten definiert, die eine Verschlechterung des Krankheitsbildes andeuten. Wird ein solches Ereignismuster in den Sensorwerten erkannt, soll über verschiedene Kommunikationswege sofort ein Krankenpfleger benachrichtigt werden.

Ein weiteres Anwendungsszenario findet sich in der Hausautomation. Intelligente Häuser besitzen eine Vielzahl von Sensoren und Aktuatoren zur Wahrnehmung und Reaktion auf Vorkommnisse im Haus. Ein Server innerhalb des Hauses verarbeitet die Sensordaten und verfügt bereits 'ab Werk' über einige Standard-Reaktionsschemata. Der Besitzer eines intelligenten Hauses möchte die Funktionen dieses Hauses jedoch gerne weitergehend an sein Verhalten individuell anpassen. Hierfür soll jedes Familienmitglied eigene Ereignismuster und die zu erfolgende Reaktion des Hauses angeben können, allerdings sollen nicht alle Mitglieder der Familie die gleichen Rechte und Möglichkeiten bekommen.

Im nächsten Abschnitt III wird nun eine Architektur vorgestellt, welche die Realisierung der genannten Anwendungsszenarien ermöglichen soll. Abschnitt III-A erläutert nachfolgend die Funktionen in Bezug auf die Szenarien beispielhaft.

III. Middleware-Architektur

Der Ansatz beruht auf einer ereignisgesteuerten Architektur, deren logische Ebenen für die speziellen Anforderungen von Sensornetzen durch zwei weitere Ebenen ergänzt wurden: eine Ebene zur Vorverarbeitung der Daten und eine weitere zur nutzerspezifischen Nachbearbeitung (siehe Abbildung 1). Alle Ebenen der Architektur werden durch Softwareagenten realisiert, welche kooperativ an der Verarbeitung von Ereignissen und Nutzeranfragen beteiligt sind (weitere Details in [1]).

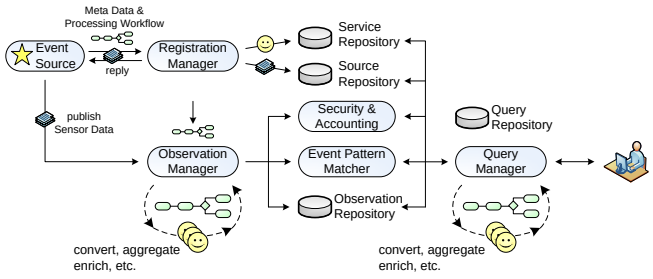


Abbildung 1. Middleware-Architektur

Die unterste Ebene stellen die *Ereignis-Produzenten*, also die Sensornetze, dar. Die in dieser Ebene erzeugten Ereignisse (primitive oder aggregierte Sensorwerte) werden vom Anbieter der Sensordaten an die Middleware übergeben. Diese Ebene ist nicht Teil der vorgestellten Architektur und es werden keine Annahmen oder Anforderungen an Entitäten dieser Ebene gestellt, sodass prinzipiell beliebige Ereignisquellen, neben Sensornetzen z.B. auch RFID-Leser, Mobiltelefone, Software-Sensoren etc., unterstützt werden.

In der zweiten Ebene bedienen spezifische Kommunikationsadapter (in der Abb. nicht dargestellt) unterschiedliche Ereignisquellen und bilden die Schnittstelle zu den durch Softwareagenten realisierten Middleware-Funktionen. Eine wesentliche Funktion bildet die Zugriffsverwaltung (*Registration Manager*), bei der sich ein Sensornetz initial registrieren muss. Bei der Registrierung werden eine Reihe von Meta-Daten über die Ereignisquelle sowie ein (optionaler) *Verarbeitungs-Workflow* verlangt. Ein Austausch von Schlüsseln und Zugriffs-Token dient der späteren gegenseiten Authentifizierung, Autorisierung und Abrechnung.

Als Teil der Meta-Daten kann eine Ereignisquelle ihre funktionalen Fähigkeiten angeben. Funktionale Fähigkeiten sind z.B. eine Anfragebearbeitung innerhalb des Sensornetzwerkes (falls Sensorknoten Messungen speichern und Nutzeranfragen selbst beantworten können) oder die Fähigkeit Instruktionen von der Middleware entgegenzunehmen (bspw. zur Adaption der Sampling-Rate, falls sich momentan kein Nutzer für die Daten des Sensornetzes registriert hat). Auf Basis der funktionalen Daten werden dedizierte Softwareagenten, welche für die Kommunikation mit dem Sensornetz zur Realisierung der Funktionen verantwortlich sind, erzeugt und in einem *Service Repository* angemeldet. Die weiteren Meta-Daten werden in einem Quellenverzeichnis (*Source Repository*), z.B. für die Abfrage durch Nutzer, hinterlegt.

Durch den erwähnten Verarbeitungs-Workflow hat eine Ereignisquelle die Möglichkeit, die von ihr bereitgestellten Sensordaten durch die Middleware in mehreren Schritten vorverarbeiten zu lassen. So können z.B. Filter-, Aggregations-, Gruppierungs- oder Übersetzungsfunktionen angegeben oder die Speicherung der Sensordaten in einer Datenbank (*Observation Repository)* veranlasst werden. Derartige Angaben erfolgen in Form eines BPEL-Prozesses, welcher durch die Middleware in eine Orchestrierung von Agenten übersetzt wird. Als Endpunkte für die Aktivitäten des Workflows können einerseits Standard-Endpunkte der Middleware, aber auch individuelle Endpunkte in Form einer beliebigen Agenten-ID angegeben werden. Somit ist es möglich die Funktionen der Middleware durch eigene, proprietäre Funktionen zu erweitern. Die Ausführung des BPEL-Workflows übernimmt der *Observation Manager*, welcher die Sensordaten an entsprechende Dienstagenten zur Ausführung der jeweiligen Prozessaktivität übergibt. Das Ergebnis der Vorverarbeitung dient als Eingabe für die nächste Ebene.

In dieser nächsten Ebene werden die vom Anbieter produzierten und von der Middleware vorverarbeiteten Ereignisse mit den Anfragen von Nutzern abgestimmt. In Abhängigkeit von den funktionalen Eigenschaften und des Verarbeitungs-Workflows der Ereignisquelle sowie den Anforderungen der Anfrage gibt es für die Abstimmung mehrere Möglichkeiten:

- Eine Laufzeitumgebung zur Verarbeitung komplexer Ereignisse (*Complex Event Processing Runtime*) kann aus einem Strom primitiver Ereignisse von den Ereignisquellen nutzerdefinierte komplexe Ereignisse, z.B. anhand von kausalen oder temporalen Beziehungen beliebiger Ereignisse, erzeugen (*Event Pattern Matching*).
- Das *Observation Repository*, eine Datenbank zur Speicherung von Sensordaten, kann abgefragt werden.
- Falls es die Ereignisquelle erlaubt, können spezielle Dienstagenten des *Service Repositories* Instruktionen an das Sensornetz schicken um eine Nutzeranfrage direkt

von der Ereignisquelle bearbeiten zu lassen.

Das Ergebnis dieser Abstimmung von Sensordaten und Nutzeranfragen kann - gleich welche Abstimmungsmethode in Abhängigkeit oben erwähnter Präferenzen gewählt wurde - auf Wunsch des Nutzers in einer letzten Ebene nachbearbeitet werden. Ziel dieser Nachbearbeitung ist die Erzeugung eines Anwendungsereignisses (*Application Level Event*), welches die für eine Anwendung relevanten Daten in einem entsprechenden Format enthält. Genau wie der Anbieter von Sensordaten einen Verarbeitungs-Workflow angeben kann, hat auch der Nutzer die Möglichkeit einen Teil der Nachbearbeitung von relevanten Ereignissen zu beeinflussen indem er z.B. weitere Filter-, Gruppierungs-, Aggregations- oder Translationsfunktionen angibt. Außerdem kann er, sobald er seine Anfrage zusammen mit dem Verarbeitungs-Workflow bei der Middleware registriert, angeben, zu welchem Endpunkt das Anwendungsereignis später gesendet werden soll. So ist es bspw. möglich das Anwendungsereignis entsprechend formatiert als SMS an ein Mobiltelefon verschicken zu lassen, obwohl die Registrierung über einen PC erfolgte.

Alle Verarbeitungsschritte, die Anbieter oder Nutzer in ihren jeweiligen Verarbeitungs-Workflows angeben, werden durch einen *Accounting Manager* protokolliert. So können Dienstleistungen der Middleware, aber auch die Bereitstellung von Sensordaten durch den Anbieter abgerechnet werden. Auf diese Weise ist es möglich nahezu beliebige Geschäftsmodelle zu realisieren, da durch die Middleware erbrachte Dienste bei Anbieter und Nutzer sowie durch den Anbieter produzierte Daten über den Nutzer abgerechnet werden können. Authentifizierungs- und Autorisierungsmaßnahmen bieten die Grundlage für entsprechende Abrechnungsmodelle.

A. Beispiel

Angenommen das Lagerhaus aus dem Anwendungsbeispiel in Abschnitt II betreibt eine solche Software-Infrastruktur und bietet seinen Klienten entsprechende Dienste an. Das Handelsunternehmen möchte, wie beschrieben, informiert werden, sobald eines von drei Ereignissen im Lagerhaus eingetreten ist, also sobald der Container eintrifft oder sich verspätet und falls Sensorwerte des intelligenten Containers eine Beschädigung der TV-Geräte vermuten lassen. Hierfür muss das Handelsunternehmen ein Ereignismuster definieren, welches vereinfacht folgendermaßen aussieht:

```
SELECT * FROM SmartContainerEvent
WHERE id BETWEEN x AND y AND
  ((overallacc > 5 OR humidity > 80 OR
    temperature > 50) OR
   [weitere Bedingungen])
```

Ein solches Muster, dargestellt in der SQL-ähnlichen *Esper Event Pattern Language* [2], wird in diesem Beispiel von der Complex Event Processing-Laufzeitumgebung verarbeitet, indem diese aus dem Strom primitiver Ereignisse korrespondierende komplexe Ereignisse zu identifizieren versucht. Die Variablen x und y stehen bspw. für RFIDs und identifizieren die TV-Geräte oder den Container, $overallacc$, $humidity$ und $temperature$ referenzieren Messungen der Container-Sensoren. Sobald das Muster in Übereinstimmung mit Ereignissen gebracht wurde, wird ein komplexes Ereignis erzeugt. Dieses Ereignis kann entweder direkt an das Handelunternehmen übermittelt werden, wobei dieses anschließend für die weitere Verarbeitung verantwortlich ist, oder noch in der Middleware weitergehend bearbeitet werden, um ein maßgeschneidertes Anwendungsereignis zu erzeugen.

Ein solches Anwendungsereignis beinhaltet nur die für eine Anwendung relevanten Daten in einem entsprechenden Datenformat. Eine Anwendung muss entsprechend nicht mehr etwaige Transformations-, Filterungs-, Aggregationsfunktionen etc. ausführen, sondern kann das Ereignis direkt weiterverarbeiten. Entsprechende Standardfunktionen müssen somit nicht von jeder Anwendung neu implementiert werden und entlasten die Verarbeitungsressourcen des Anbieters. Dies ist insbesondere für ressourcenarme, mobile Geräte von entscheidender Bedeutung. Der Inhaber des Handelsunternehmen könnte so zum Beispiel bei Eintreten eines entsprechenden Ereignisses direkt über sein Mobiltelefon informiert werden, dass die TV-Geräte während des Transports zu stark erschüttert wurden, und sofort einen Mechaniker rufen, der die Geräte noch im Lagerhaus begutachtet. Notwendige Vorraussetzung hierfür sind jedoch weitere Verarbeitungsschritte, die ein komplexes Ereignis in ein entsprechendes Anwendungsereignis überführen. Neben einem komplexen Ereignismuster kann das Handelsunternehmen also noch zusätzlich einen Verarbeitungs-Workflow vorgeben, der diese Überführung vollzieht. Ein einfaches Beispiel eines solchen Workflows ist in Abbildung 2 dargestellt.

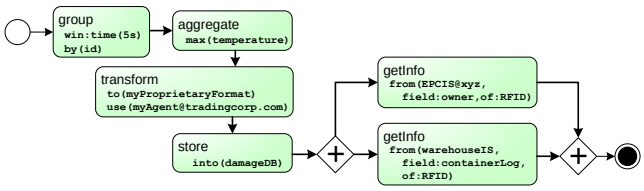


Abbildung 2. BPMN-Diagramm eines Verarbeitungs-Workflows

Dieser Beispiel-Workflow, welchen das Handelunternehmen zusammen mit obigem Ereignismuster bei der Middleware registrieren würde, beschreibt, wie die Middleware eine entsprechende komplexe Ereignisinstanz verarbeiten soll. Im gegebenen Beispiel sollen alle erzeugten komplexen Ereignisse zunächst in einem Zeitfenster von fünf Sekunden gruppiert und anschließend bzgl. der maximalen Temperatur aggregiert werden. Bevor im übernächsten Schritt das Resultat in einer Datenbank für Archivierungszwecke gespeichert werden soll, verlangt das Handelsunternehmen noch die Transformation des komplexen Ereignisses in ein proprietäres Format. Da das Lagerhaus einen entsprechenden Dienst nicht bietet, wird das Ereignis von der Middleware an einen externen Agenten des Handelsunternehmens geschickt. Erst nach dessen Transformation und Rückantwort wird das Ergebnis in der Datenbank gespeichert. Alle bisherigen Schritte mussten sequentiell ausgeführt werden, die nachfolgenden Schritte jedoch sind unabhängig und können parallel ausgeführt werden. In diesen Schritten werden zusätzliche Informationen über die mit

RFID-Chips versehenen TV-Geräte von einem externen *EPC Information Service* abgerufen und gleichzeitig das Sensor-Logbuch des intelligenten Containers an das komplexe Ereignis angehängt.

Ist der Verarbeitungs-Workflow erfolgreich abgearbeitet worden, werden die Resultate vom *Query Manager* als Anwendungsereignis zurück an das Handelsunternehmen oder einen anderen gewünschten Endpunkt gesendet.

IV. Implementierung

Die Implementierung basiert auf der *Jadex Event Stream Processing Architecture* [1] (JESPA), einer von uns entwickelten Architektur zur verteilten Verarbeitung von Ereignisströmen. Diese Architektur zeichnet sich u.a. durch ihre Flexibilität und Erweiterbarkeit aus, weshalb sie in großen Teilen den Kern der hier vorgestellten Middleware darstellt.

Wesentliche Teile des beschriebenen Konzeptes sind bereits umgesetzt. Auf unterster Ebene wurden bspw. Adapter zur Kommunikation mit Ereignisquellen implementiert. Ein Adapter für das *Nokia 6131 NFC*-Telefon [3] erlaubt die Verarbeitung von RFID-Daten, *SunSPOT*-Sensoren [4] liefern Helligkeits-, Feuchtigkeits- und Beschleunigungswerte, ein *Android*-Adapter [5] stellt aktuelle Aktivitäten eines Nutzers zur Verfügung und ein erster Prototyp-Adapter für die *MagicMap*-Software [6] erlaubt die Verarbeitung von Positionsschätzungen realer Objekte, welche MagicMap aus Signalstärkemessung diverser Funktechnologien errechnet.

Für die Dienstorchestrierung zur Vor- und Nachverarbeitung von Ereignissen in der Middleware sorgen spezielle Dienstagenten, welche mittels des *Jadex v2*-Agentensystems [7] implementiert wurden. Jadex unterstützt in der v2-Version nicht nur das Programmieren von 'intelligenten' *BDI*-Agenten (BDI=Belief, Desire, Intention) für komplexe Aufgaben, sondern auch das Erzeugen leichtgewichtiger Agenten, die einfache Dienstaufgaben übernehmen und für die dedizierte Verarbeitung einzelner Ereignisinstanzen verantwortlich sind.

Die *Esper*-Laufzeitumgebung [2] schließlich sorgt für die Identifizierung und Erzeugung komplexer Ereignisse basierend auf Strömen primitiver Ereignisse von Sensornetzen und liefert mit seiner *Event Pattern Language* die Basis zur Spezifikation von Ereignismustern für Nutzer von Sensordaten.

Die Implementierung konnte jedoch bisher nicht ausreichend evaluiert werden, da zur Umsetzung komplexer Szenarien noch wesentliche Bausteine der Architektur realisiert werden müssen.

V. Fazit

In dieser Arbeit wurde eine Middleware zur verteilten Abfragebearbeitung von Sensordaten vorgestellt. Ziel der Middleware ist es, Nutzer einen transparenten Zugriff auf relevante Sensordaten zu erlauben, indem die Heterogenität der Hard- und Software durch die Middleware verborgen wird. Der Zugriff auf Sensordaten kann über verschiedene Wege, entweder durch Identifikation komplexer Ereignismuster in Ereignisströmen, der Abfrage von Sensor-Historien aus Datenbanken oder der durch die Middleware vermittelten Kommunikation mit dem Sensornetzwerk selbst, erfolgen. Anbieter und Nutzer von Sensordaten haben die Möglichkeit durch Angabe von BPEL-basierten Verarbeitungs-Workflows die Vor- und Nachbearbeitung von Ereignissen durch die Middleware zu veranlassen, um so z.B. Ressourcen der Endgeräte (z.B. Sensoren, Mobiltelefone etc.) auf beiden Seiten der Verarbeitungskette zu entlasten.

In weiteren Schritten sollen bereits existierende Standards, z.B. *SensorML* [8] zur Beschreibung von Sensoren und deren Messungen, in das System integriert werden. Weiterhin sollen Gemeinsamkeit mit anderen Projekten, z.B. dem *COUGAR Sensor Database Project* [9] etc. und Möglichkeiten der Interoperabilität - sofern möglich - untersucht werden.

Literatur

[1] D. Bade, "Towards an extensible agent-based middleware for sensor networks and rfid systems," in *Inproceedings of the 3rd Int. Workshop on Agent Technology for Sensor Networks (ATSN-09)*, 2009.

[2] EsperTech, "Esper - event stream intelligence," esper.codehaus.org, 2009.

[3] Nokia, "Nokia 6131 nfc," http://www.nokia.de/A4420858, May 2009.

[4] Sun, "Sunspot," http://www.sunspotworld.com/, May 2009.

[5] Google, "Android," http://developer.android.com/, May 2009.

[6] S. Bruning, J. Zapotoczky, P. Ibach, and V. Stantchev, "Cooperative positioning with magicmap," March 2007, pp. 17–22.

[7] A. Pokahr and L. Braubach, "From a research to an industrial-strength agent platform: Jadex v2," in *Business Services: Konzepte, Technologien, Anwendungen - 9. Internationale Tagung Wirtschaftsinformatik (WI 2009)*, H.-G. F. Hans Robert Hansen, Dimitris Karagiannis, Ed. Österreichische Computer Gesellschaft, 2 2009, pp. 769–778.

[8] O. Open Geospatial Consortium, "Sensorml," http://www.opengeospatial.org/standards/sensorml, May 2009.

[9] Y. Yao and J. Gehrke, "The cougar approach to in-network query processing in sensor networks," *SIGMOD Rec.*, vol. 31, no. 3, pp. 9–18, 2002.

Priority aware Resource Management for Real-Time Operation in Wireless Sensor/Actor Networks

Marcel Baunach
Department of Computer Engineering, University of Würzburg, Germany
Email: baunach@informatik.uni-wuerzburg.de

***Abstract*—Increasing complexity of today's WSAN applications can rapidly result in reduced real-time capabilities of the underlying sensor nodes. Using preemptive operating systems is one way to retain acceptable reactivity within highly dynamic environments but commonly leads to severe resource management problems. We outline our *dynamic hinting* approach for maintaining high system reactivity by efficient combination of preemptive task scheduling and cooperative resource allocation. With respect to task priorities, our technique significantly improves classical methods for handling priority inversions under both short- and long-term resource allocations. Furthermore, we facilitate compositional software design by providing independently developed tasks with runtime information for yet collaborative resource sharing. In some cases this even allows to improve blocking delays as otherwise imposed by bounded priority inversion.**

I. Introduction

The ever increasing size, pervasiveness and demands on today's wireless sensor/actor networks (WSAN) significantly boosts the complexity of the underlying nodes. Thus, modular hardware and software concepts (e.g. service oriented programming abstractions) are more and more used to manage design and operation of these embedded systems. Then, adequate interaction between the various modules is essential to avoid typical compositional problems. Beside task scheduling, directly related issues comprise resource sharing or even real-time operation. Concerning this mixture, we find that current WSN research is still too restricted to static design concepts. As already stated in [1], next generation embedded systems will be more and more used as reactive real-time platforms in highly dynamic environments, where the true system load varies and can not be predicted a priori. In fact, we also expect a clear focus shift from pure sensing in classic WSNs towards additional pro-activity in WSAN applications (e.g. integrated control systems, precise on demand measurements, etc.). Then, preemptive and prioritized tasks are required for reliable and fast response on various events.

We present the *dynamic hinting* approach for cooperative resource sharing and real-time operation within preemptive operating systems. As often suggested [2], we take advantage of the resource manager's enormous runtime knowledge about the system's current resource requirements. This information is carefully selected and forwarded to those tasks, which currently block more relevant tasks by a resource allocation. In turn, these so called *hints* allow blocking (and even deadlocked) tasks to adapt to current resource demands and finally to contribute to the system's overall reactivity and stability.

II. Motivation and Requirements

Resource assignment in complex, modular systems with concurrently running tasks is hard to manage during development and runtime. This is particularly true, if tasks are allowed to allocate virtually any resource mix in any order or if long-term allocations collide with sporadic and time critical on demand allocations. During our research we found that reactivity and pro-activity in modern WSAN applications requires quite sophisticated real-time and smart adaptive resource concepts. We'll just give a short example from a real-world application:

A radio protocol task commonly requires long-term allocation of the used transceiver in combination with relatively short but sporadic access to the interconnection bus. Obviously, both resources need specific configuration and thus are non-preemptive. Using the bus becomes time critical when radio transmission slots must be obeyed or when a receive buffer must be read and cleared quickly to allow the reception of further radio packets. Concurrent to this communication task, sensor tasks often use exactly the same bus for continuous data streaming. Again both resources are non-preemptive but this time, the bus is also locked in a long-term allocation. The resulting compositional problem is already hard to solve. Even if task priorities can be selected carefully to indicate the desired relevance of each task, their compliance can not be guaranteed. Instead, knowledge about the overall system load (including further tasks) must be incorporated manually into the code. The regular release of a long-term resource could be one solution. However, this strategy might impose considerable overhead when deallocation and re-allocation are expensive in time and energy. Where data streams often require explicit termination (trailers) and initiation (headers), resources might require a time-consuming (de)initialization procedure upon each (de)allocation. Using server tasks or stateful libraries for managed operation of such resources is also no universal solution since this would just relocate the problem and cause additional overhead.

III. Related Work in WSN/WSAN Systems

Non-preemptive systems with run-to-completion tasks are very common in the WSN domain and prevent some resource conflicts implicitly since task executions can't be interleaved. Yet, this often causes bad reactivity to sporadic events. Therefore, e.g. TinyOS [3] and Contiki [4] support preemptive extensions but then lack priorities and resource management entirely. *Preemptive systems* potentially provide much better

reactivity since a task can be preempted for a more important action implemented in another task. Yet, preemption yields no instant advantage if the action requires a shared resource which is exclusively held by a less important task. Resulting problems like bounded or unbounded priority inversion [5] might lead to thwarting of high priority tasks and even deadlocks may occur. In any case, the task priorities defined by the developer are not obeyed as desired. To cope with some of these issues, well studied protocols like *priority ceiling*, *highest locker* or *priority inheritance* [6] are found in some embedded operating systems. We selected the priority inheritance protocol (PIP) as basic technique for our approach. Again, preemptive WSN operating systems like MANTIS [7] or RETOS [8] do not consider real-time or resource related problems at all.

IV. Resource Management and Dynamic Hinting

The central objective of our approach is to allow tasks $t \in T$ the collaborative sharing of exclusive resources. At the same time, it supports them to closely comply with their intended *base priorities* P_t. The basic idea behind *dynamic hinting* might be applied as integral concept for many real-time operating systems if these support three central features:

1) truly preemptive and prioritized tasks,
2) non-preemptive (i.e. exclusive) resources,
3) temporally limited resource requests (e.g. via deadline).

While the first two features can be found quite often, the last requires a special timing concept. Then however, tasks can request resources which are still held by other tasks. In this case, a requester is suspended until the resource is released (and handed over) or until the timeout occurs (and the request is denied). Though this allows to cope with allocation failures, it can also induce long resource request chains (Fig. 1, 2a). In case of infinite timeouts, even deadlocks may occur (Fig. 2b). These are already critical if two tasks mutually request a resource which is held by the other one, respectively.

Thus, many conservative resource management systems try to avoid deadlocks by simply refusing a resource request *immediately* if it would cause an allocation cycle. Others accept at least resource chains and simply suspend each requester task until it can be served. In our opinion, both methods are not satisfying since exactly the just rejected or suspended task h alone has to cope with the situation. This is especially annoying if h is truly more important than at least one other task l in the just averted cycle or extended chain. It also results in a violation of base priorities ($P_l < P_h$). Furthermore, resources are usually indispensable when requested and thus, tasks tend to retry infinitely until the allocation succeeds. The resulting (active) loops or long timeouts might not only block other tasks but even worse, they simply shift the problem back from system level to task level.

Indeed, task-resource-dependencies are highly dynamic and depend on the system wide allocation order during runtime. Hence, another task might react much better than h if it knew about the situation. Unfortunately, tasks are commonly not aware about their spurious influence and so the allocation

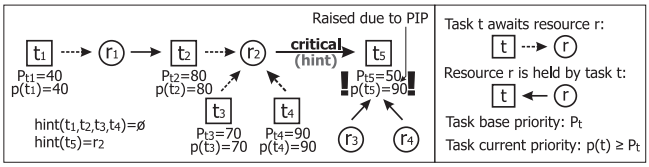


Figure 1. Example for Priority Inheritance and *Dynamic Hinting*

chains are commonly reduced successively, beginning at their very end. This is exactly where *dynamic hinting* applies.

Our approach provides runtime information for each task about which resource it should release to improve the overall system reactivity and liveliness. Considering these so called *hints* is always optional for each task. But if followed, it definitely reduces direct, chained or deadlock blocking of at least one higher priority task (→Fig. 1, 2).
Therefore, two preconditions must be fulfilled:

1) An ongoing resource allocation must never prevent any task from requesting any resource. Otherwise, our approach lacks knowledge about the system requirements.
2) A spurious task must receive the time and opportunity to react on a hint.

In our case, PIP provides the necessary possibility and priority (1) and the limited waiting of other tasks provides the time (2). PIP adjusts task priorities dynamically at runtime and according to the current resource assignment situation. It selects each task's l *current priority* $p(l) \geq P_l$ to be at least as high as the current priority $p(h)$ of the highest prioritized task h it currently blocks by virtue of a resource allocation.

Then, the first step for determining hints is to identify the *critical resources* for each task l. These currently define $p(l) \gneq P_l$ and thus, they directly or indirectly cause the blocking of more important tasks with base priority truly above P_l:

$$crit(t) := \{r \in R | r \text{ defines } p(t) \text{ by PIP}\} \quad (1)$$

In turn, t can reduce the blocking of at least one task by releasing any $r \in crit(t)$. Yet, our approach always selects the hint as follows (→Fig. 1, 2):

$$hint(t) := r \in crit(t),\ r \text{ was requested last.} \quad (2)$$

Then, if l releases its hinted resource r, it is directly passed to its first requester, w.l.o.g h. Next, $p(l)$ is updated by PIP and h is scheduled promptly. This is true since then h holds the highest priority of all tasks in ready state and l did let h pass by. As soon as l is scheduled again, it can immediately re-request r to continue its operation quickly. In any case, the untimely release of a hint resolved a priority inversion and accounted for the intended task base priorities.

The example in Figure 1 shows $crit(t_5) = \{r_2\}$ since $p(t_5) > P_{t_5}$ was defined by t_4's request for r_2. Releasing r_2 would instantly relax $p(t_5) := P_{t_5}$. Then t_4 is served and scheduled since it is indeed the task with highest priority but currently blocked by t_5. The allocation timeout t_4 specified for r_2 grants t_5 the time to cooperate as described. If t_5 follows its hint r_2 prior to its regular release, it indeed improves the bounded priority inversion toward t_4. Furthermore, t_5 also improves the reactivity of t_2 and t_3 since these tasks are also

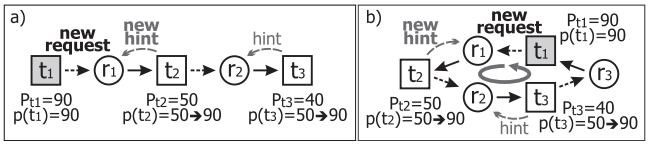


Figure 2. Dynamic Hinting Examples: a) Chain, b) Deadlock

more relevant ($P_{t_2} > P_{t_3} > P_{t_5}$) and will receive r_2 right after t_4.

Next, we'll describe two exemplary ways in which a task may receive and handle its hints: First, an *explicit query* can be done at distinct points in time or at code positions where its handling would be possible at all. Then however, a task can never react as long as it is in *waiting state*. Yet, this is exactly the case upon deadlocks and during many long-term allocations, where tasks e.g. wait for some events/interrupts while holding a resource. Beside this severe weakness, the manual effort and code pollution would be immense.

Thus, we recommend a much better strategy called *early wakeup*. When enabled, all functions by which a task suspends itself may return early upon a new or changed hint. Then, a dedicated return value will indicate this special situation. This way, coping with hints can be done instantly and it is entirely limited to the cases when they really occur. The use of *early wakeup* can be selected and tuned individually by each task t and for each self-suspension. Therefore, we extended the involved functions by an additional threshold parameter φ:

```
result_t sleep(deadline | timeout, φ)
result_t waitEvent(event, deadline | timeout, φ)
result_t getResource(resource, deadline | timeout, φ)
```

Then, a self-suspending function will only return early if

$$(\varphi \neq 0 \ \wedge \ p(t) > P_t \ \wedge \ p(t) \geq \varphi), \tag{3}$$

i.e. if priority inheritance raised the caller's priority $p(t)$ to at least the specified threshold φ. In particular, these functions will also return right after calling if a hint is already available.

E.g. both new requests in Fig. 2a,b will immediately resume t_2 if it has *early wakeup* enabled. Then, its request for r_2 is withdrawn. Otherwise, or if t_2 refuses to release its hint r_1 and simply requests r_2 again, t_3 may wake up early. Obviously, a single cooperative task in a chain or cycle is already sufficient to improve or recover from this situation.

Of course, priority thresholds are not the only useful metric for a task to decide between cooperative or egoistic behavior. Thus, beside the hinted resource r, we grant each task t access to some further information: Its current (raised) priority $p(t)$, a flag indicating that a deadlock situation might persist if the hint is not followed, and the absolute time at which the hint r expires due to the latest request timeout:

```
Resource* getHint(Priority_t* p, boolean* DL, Time_t* TO);
```

The latter is of special interest for applying time-utility-functions [9]. These allow to relate the remaining allocation time to the still remaining timeout. Another option is to introduce a real-time priority threshold by initially defining φ equal for all tasks. This inherently limits the potential co-operativeness to situations where tasks (directly or indirectly) block any real-time task t_R with $P_{t_R} \geq \varphi$.

V. Real-World Applications and Test Bed

For analyzing our approach of combining temporally limited resource requests, the priority inheritance protocol and *dynamic hinting*, we extended the *Smart*OS [10] kernel as described since it is available for several sensor nodes, provides appropriate task, timing and resource basics, and thus allowed an easy integration. The implementation was done for Texas Instrument's MSP430 family of microprocessors, since these are found on a large variety of sensor nodes. Requiring $4\,\mathrm{kB}$ of ROM and $150\,\mathrm{B}$ of RAM for the whole kernel, the typically low computational performance and small memory of sensor nodes was considered carefully to leave sufficient room for the actual application.

Our test bed considers a quite frequent problem we also had in one of our real WSAN control applications: A task S continuously transfers some data over a shared bus b to an external device. The stream is rather long (or even infinite) but can be suspended and resumed at any time for more important communication over the same bus. Therefore, it always needs some bus setup plus a complex header/trailer for proper initiation/termination. During the transfer, S needs exclusive access to b. A common solution is to split the stream payload into atomic packets. Then, S would terminate the stream and release the bus temporarily after each packet. This way other tasks may receive the bus regularly. However, since S does not really know if it currently blocks a more relevant task, the temporary interruption might be completely unnecessary. Furthermore, the selected packet length defines the duration of potentially resulting priority inversions. By using short (long) packets, the overhead increases (decreases) while improving (degrading) the reactivity of higher prioritized tasks when these request b. Commonly, a fixed length is selected during development with regard to the individual application requirements. These must be known exactly, then. Using a server task for coordinating the bus access might even result in slightly worse performance due to client-server communication overhead. The mentioned problems remain the same but are concentrated at the server which also commonly creates atomic packets or grants exclusive bus reservations.

Dynamic hinting provides two options for improvements. Since our approach knows about pending bus requests, S could query its current blocking state periodically and react only if necessary. Though the query interval must still be selected carefully, the overhead for useless stream interruption is already avoided! The additional use of *early wakeup* finally improves the reactivity as it hints S *instantly* and *only* if it blocks a task with truly higher base priority.

For the concrete application we had to stream $8\,\mathrm{bit}$ ADC data sampled at $10\,\mathrm{kHz}$ over an SPI bus. The overhead for each header and trailer was 1 byte. Beside, a radio transceiver R and a motor controller M shared the same SPI bus (at different settings) for short communication. Yet, both associated tasks had to process sporadic events (av. inter-arrival time: $\approx 5\,\mathrm{ms}$) and were much more time and safety critical since especially failures in the motor control were disastrous. So, we defined

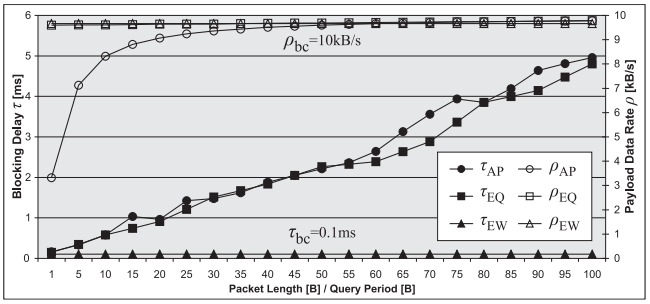


Figure 3. Streamtest: Packet Oriented vs. *Dynamic Hinting*

$P_S < \varphi = 100 < P_R < P_M$. To reduce CPU load we used a DMA channel between ADC and the bus controller. Thus, S simply had to allocate and configure the bus resource for a new stream. After starting the DMA transfer, S did sleep until an event signaled to finalize the stream or a hint occurred. The following example code shows the relevant implementation details for S when using *early wakeup*:

```
1  void streamData() {
   int stop = 0;
3  /* start stream */
   getResource(&SPI, INFINITE, 0);
5  cfgBus(); header(); startDMA();
   while (stop != 1) {       // 1 indicates stop event
7    /* Wait infinitely for the stopStream event.
        Enable early wakeup if raised >= φ = 100. */
9    stop = waitEvent(&stopStream, INFINITE, φ);
     if (stop == -1) {       // hint received!
11     Resource_t *hint = getHint(NULL, NULL, NULL);
       if (hint == &SPI) {   // conditional hint handling
13       /* stop stream and release resource quickly */
         stopDMA(); trailer(); releaseResource(hint);
15 /* --- THE TASK WILL BE SUSPENDED HERE SINCE AT ---
     --- LEAST ONE OTHER TASK WAITS FOR THE HINT ---- */
17       /* continue stream as soon as possible */
         getResource(hint, INFINITE, 0);
19       cfgBus(); header(); startDMA(); } } }
   /* stop stream */
21 stopDMA(); trailer(); releaseResource(&SPI); }
```

Streaming data simultaneously to sporadic but highly reactive tasks might already cause extreme system load for low performance embedded systems like sensor nodes. Yet, the testbed results show, that our approach can still gain good reactivity and high throughput without manual task tuning. First, we implemented the application with atomic fixed-length packets (AP), then we used *dynamic hinting* with *explicit querying* (EQ) and finally we activated *early wakeup* (EW). Fig. 3 shows the results in terms of the average blocking delay τ of the real-time tasks and the achieved payload data rate ρ of the streaming task. Due to the fixed trailer length and sampling rate, the best case values are τ_{bc}=100 ns and ρ_{bc}=10 kB/s.

As expected for the packet oriented design, its throughput ρ_{AP} improves while the blocking delay τ_{AP} degrades rapidly with increasing packet length. When using *dynamic hinting* with periodic explicit querying, ρ_{EQ} remains nearly constant and close to the achievable maximum. However, the blocking delay τ_{EQ} almost matches τ_{AP} and is also not satisfying for long periods (for short ones, the task causes higher CPU load). When using *early wakeup*, the data rate is still held high while the blocking delay is kept extremely low. Indeed, $\rho_{EW} \approx \rho_{bc}$ and $\tau_{EW} \approx \tau_{bc}$. For better comparability, ρ_{EW} and τ_{EW} are visible as horizontal lines in Fig. 3. Yet, *early wakeup* is independent from any packet length or query period.

VI. Conclusion and Outlook

In this paper, we outlined the *dynamic hinting* approach for cooperative resource sharing among preemptive tasks in reactive systems. In particular, the individual task base priorities are considered carefully to keep each task's performance close to its intended relevance. Therefore we analyze emerging task-resource dependencies at runtime and provide spurious tasks with information about how they could increase the reactivity of more relevant tasks or to recover from deadlocks. Thereby, tasks can collaborate even without explicit knowledge of each other. Nevertheless, each one can decide dynamically between cooperative or egoistic behavior with respect to its current conditions and other tasks' requirements.

While our approach is not necessarily limited to the WSAN domain, our implementation and test bed showed, that using compositional software design and prioritized tasks allows to create quite reactive systems even on small embedded devices.

At present we research more sophisticated concepts for adjusting the acceptance of hints to the task and system situation. In particular, we want to improve the hint selection and the application of TUFs. Also, we plan to evaluate the use of *dynamic hinting* for remote resource management in distributed systems. Concerning real-world applications, we just integrated our approach into a WSAN based indoor localization and car control system, where we achieved a considerably higher localization frequency and path precision.

References

[1] Giorgio C. Buttazzo, "Real-Time Scheduling and Resource Management," in *Handbook of Real-Time and Embedded Systems*, I. Lee, J. Y.-T. Leung, and S. H. Son, Eds. CRC Press, 2007.

[2] N. Audsley, R. Gao, A. Patil, and P. Usher, "Efficient OS Resource Management for Distributed Embedded Real-Time Systems," in *Proceedings of Workshop on Operating Systems Platforms for Embedded Real-Time applications*, Dresden, Germany, Jul 2006.

[3] U. Berkeley, "TinyOS," Web site http://www.tinyos.net/, 2004.

[4] Adam Dunkels, Bjorn Gronvall, and Thiemo Voigt, "Contiki - A Lightweight and Flexible Operating System for Tiny Networked Sensors," in *LCN '04: 29th IEEE International Conference on Local Computer Networks*. IEEE Computer Society, 2004.

[5] O. Babaoğlu, K. Marzullo, and F. B. Schneider, "A formalization of priority inversion," *Real-Time Syst.*, vol. 5, no. 4, pp. 285–303, 1993.

[6] L. Sha, R. Rajkumar, and J. P. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization," *IEEE Trans. Comput.*, vol. 39, no. 9, pp. 1175–1185, 1990.

[7] Shah Bhatti, James Carlson, Hui Dai, Jing Deng, Jeff Rose, Anmol Sheth, Brian Shucker, Charles Gruenwald, Adam Torgerson, and Richard Han, "MANTIS OS: an embedded multithreaded operating system for wireless micro sensor platforms," *Mob. Netw. Appl.*, vol. 10, no. 4, 2005.

[8] Hojung Cha, Sukwon Choi, Inuk Jung, Hyoseung Kim, Hyojeong Shin, Jaehyun Yoo, and Chanmin Yoon, "RETOS: resilient, expandable, and threaded operating system for wireless sensor networks," in *IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks*. New York, NY, USA: ACM, 2007.

[9] Peng Li, Binoy Ravindran, and E. Douglas Jensen, "Adaptive Time-Critical Resource Management Using Time/Utility Functions: Past, Present, and Future," *Computer Software and Applications Conference, Annual International*, vol. 2, pp. 12–13, 2004.

[10] M. Baunach, R. Kolla, and C. Mühlberger, "Introduction to a Small Modular Adept Real-Time Operating System," in *6. Fachgespräch Sensornetzwerke*. RWTH Aachen University, 16.–17. Jul. 2007.

Exploiting Platform Heterogeneity in Wireless Sensor Networks for Cooperative Data Processing

(Extended Abstract)

Andreas Reinhardt, Ralf Steinmetz
Multimedia Communications Lab, Technische Universität Darmstadt
Merckstr. 25, 64283 Darmstadt, Germany
Email: {andreas.reinhardt, ralf.steinmetz}@kom.tu-darmstadt.de

***Abstract*—In wireless sensor networks, nodes are often fitted with low-power components to allow for a long node lifetime when operated on batteries. However, these available resources can be insufficient to perform sophisticated data processing on a local scale, necessitating the transmission of all sensor readings to an external sink. These transmissions are expensive both in terms of delay and energy, and thus undesirable. To alleviate the situation, we propose the use of a heterogeneous sensor network with higher-capacity processing nodes that allow to perform more complex data processing operations within the sensor network. Estimates of the energy consumptions confirm that employing a heterogeneous sensor network can preserve energy and thus lead to an extended lifetime of the network.**

I. Introduction

Nodes in wireless sensor networks (*WSNs*) are generally designed with energy considerations in mind to allow for long lifetimes when operated on batteries [1]. These energy savings however often come at the cost of low-power microcontroller units (*MCUs*) with reduced computational capabilities and low clock frequencies. With only a few kilobytes of RAM and program Flash storage, the possible complexity of applications is additionally limited. These tight resource limits of sensor node platforms (*motes*) disallow some operations to be performed within the sensor network, and commonly require the transmission of data to external nodes which perform the resource-intensive processing tasks. Commonly, the situation is resolved by transmitting all collected data (often only slightly processed, if at all) to an external sink node which performs the processing tasks.

This data forwarding process is however expensive in terms of energy, as especially in the case of multi-hop transmissions in large networks, delay and energy demand increase linearly with the number of hops. A common way to alleviate the number of transmissions in the multi-hop case is data aggregation ([2], [3]), where packets that share the same route are merged on their way to the destination. Aggregation can thus lead to an overall reduction of the number of packets sent, although further data-specific processing is generally not performed.

As a high volume of traffic might still be present in the network, we propose to move the data processing into the network by deploying dedicated processing nodes. These processor nodes provide greater MCU power (and ideally, larger memory sizes) than the deployed nodes to allow for tasks with greater complexity to be performed within the network. By forming a sensor network, which is heterogeneous in terms of computational power, demanding processing operations can be performed within the network, and thus the amount and size of packet transmissions to a base station significantly reduced.

To show the feasibility of this approach, we exemplarily discuss three application scenarios that would significantly benefit from processing the data inside the network. Concisely, we evaluate the demands of *data compression*, *cryptography*, and *high data-rate sample processing*. To provide computational resources for in-network processing, we exemplarily assume TelosB [4] and SunSPOT [5] devices, as they are present in our TWINS.KOM testbed [6]. However, other combinations of motes are possible as well.

After presenting the related work in Sec. II, we present our vision of collaborative data processing in Sec. III, and show a theoretical energy analysis in Sec. IV. We conclude this paper in Sec. V, where we summarize our results and present the next steps.

II. Related Work

Existing *hybrid* sensor network architectures target to reduce the number of hops a packet requires to reach its destination by supplementing a WSN by additional connections over a secondary, often wired, medium.

Sharma and Mazumdar have investigated the use of *limited infrastructure*, i.e. networks with a number of wired connections between sensor nodes, in [7]. Their approach establishes a small-world graph utilizing wired links between a subset of nodes to reduce the overall energy demand as well as the different energy consumption rates of participating nodes. The additional efforts required for the wiring however make it suited for long-term deployments of sensor networks only.

Hu et al. have built a hybrid network from Mica2 motes and Stargate devies for detecting cane toads in northern Australia [8]. Similar to our proposed system, a two-tiered sensor network structure with low-power motes and higher-power processing nodes is given. However, the Stargate's comparably high energy consumption of 4 watts leads to a quick depletion of its battery and thus renders the solution unsuited for long-term autonomous operation.

Wagenknecht et al. also propose to deploy nodes with higher computational capabilities within a WSN to act as cluster-

heads for *sensor subnetworks*, i.e. partitions of the sensor network [9]. They use embedded systems with a 233 MHz clock frequency and 128 megabytes of RAM as the backbone to interconnect the sensor subnetworks through a wireless mesh network. Although deploying additional gateways allows for shorter multi-hop routes, the energy savings are possibly counterbalanced by the greater energy requirements of the gateways, which are not analyzed in detail in the paper.

A different approach to shift computational tasks into the network is the use of mobile agents. In such networks, data is not forwarded to an external sink, but instead, the processing application (the *mobile agent*), including its state variables, are sent to the node and executed locally [10]. As all process context data are contained within the agent, it can be supplied with input data at one node, while the processing can be performed at a different, more powerful, system. We thus consider it a well-suited supplement to migrate tasks between nodes.

Further dimensions of heterogeneity have been analyzed, such as heterogeneous link qualities ([11], [12]), or energy level heterogeneity [11]. The authors however focus on means to alleviate heterogeneity rather than exploit it.

III. Collaborative Data Processing

Often, motes are too weak to perform computationally intensive tasks locally, or do at least not provide sufficient energy budgets to perform the demanding operations numerous times during their battery lifetime. Especially in the presence of low-power 8-bit microcontroller units (*MCUs*), operations to process 32-bit data require many more instructions than when executed on a native 32-bit platform.

To overcome this limitation of many existing sensor networks, we propose deploying a heterogeneous set of nodes with two distinct levels of computational capability. Like other WSNs, low-power sensor nodes perform sensing and basic processing tasks, and an external sink node acts as data collector. However, additional dedicated nodes are present within the network to perform data processing operations, thereby alleviating the energy-consuming multi-hop data transport to the sink, while overcoming resource limits of low-power platforms. We indicate a sample network topology with nine low-power sensor nodes, two processor nodes, and a single external sink in Fig. 1.

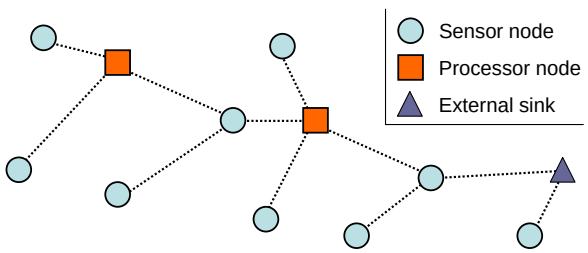


Fig. 1. Exemplary heterogeneous sensor network topology

It is essential to distinguish our system from related work where nodes with secondary network interfaces, peripheral ports, and power consumptions of several watts are used. Instead, we propose the use of low-power embedded systems with support for duty-cycled operation, such as SunSPOTs or Intel's Imote2 nodes. Both resemble native 32-bit architectures with greater computational capabilities, but also greater power consumptions than the low-power motes. We briefly compare both platforms to two common mote platforms in Table I.

TABLE I
Comparison of mote platforms

	Mica2[13]	TelosB[4]	SunSPOT[5]	Imote2[14]
RAM size	4 kB	10 kB	512 kB	32 MB
System clock	7.37 MHz	8 MHz	180 MHz	104 MHz*
Sleep current	15 µA	1 µA	31 µA	820 µA
Active current	8 mA	1.8 mA	80 mA	66 mA
Word size	8 bits	16 bits	32 bits	32 bits

* The Imote2 can dynamically adjust its clock frequency between 4 and 416 MHz

In the following, we present how three common application scenarios for WSNs can be supported by our proposed heterogeneous WSN architecture.

A. Data Compression

As both route length and packet size of multi-hop radio transmissions in WSNs have a dependency on the overall energy consumption of the transfer, data compression (also referred to as *source coding*) is a viable approach to compact data (e.g. [15]) prior to sending it. However, the limited amount of memory present on motes is often insufficient to store complex models or code tables, and thus leads to degraded compression gains.

As we have determined in [16], compressing data on a per-packet basis often leads to no improvements over the uncompressed data size at all, while knowledge about the temporal history of data can achieve significant size reductions. Supplementing the source coding by means of in-network processing, such as data aggregation (cf. [2]), can even lead to further savings, but also has higher computational demands on nodes that perform the processing operation. This makes the presented data compression scenario well suited for the proposed heterogeneous networks.

Applied to the scenario depicted in Fig. 1, the processing nodes should integrate within the tree rooted at the external sink node. When configured to aggregate packets on their way to the sink, and compress the results, their presence can lead to energy savings resulting from an overall smaller number of transmissions.

B. Cryptography

High computational demand is also present in area of cryptography, where many algorithms rely on heavy use of the modulo operation. Especially, when the used key length exceeds a platform's native word size, additional operations to emulate the corresponding operations are required, which come at a significantly increased time and thus energy consumption. Emulating these instructions is however often necessary to ensure sufficiently large key lengths.

Measurements on real hardware, performed by Gura et al. in [17], have shown that both ECC and RSA-1024 require more than 4.5 seconds to execute on an 8-bit microcontroller clocked at 14.7 MHz. When more powerful processing nodes are integrated with the sensor network, their greater computational capabilities allow them to perform strong cryptography within reasonable time limits.

Especially, the 32-bit word size and the significantly increased RAM size of the processor nodes reduce the need for instruction emulations and expensive data buffering on external memory, and can thus reduce the required execution time. When the processor nodes act as in-network terminals to provide secured links to the sink node, low-power sensor nodes can employ the AES-128 support of their CC2420 radio transceiver [18] to establish encrypted connections to the processor nodes with a low hop count.

C. High Data-Rate Sample Processing

When sensors that generate high-volume data (such as image or audio sensors) are present within the network, their samples cannot be processed by the sensor node at all times, but are instead forwarded to the sink for further processing. The lack of hardware multipliers in many embedded systems also limits the use of algorithms with many multiplication and addition operations, such as the Fast Fourier Transform (*FFT*). Transferring all data to the sink however leads to a significant volume of traffic in the network.

If instead, a heterogeneous set of nodes is present in the network, resource-demanding tasks can be performed in less time when configuring the processor nodes to specialize on these tasks and request the sensor nodes to transmit their data there. Due to their higher clock frequency and the larger RAM size, the processor platforms inherently consume more energy in all operation modes. However, their reduced processing time improves both transmission delays and power demand, and thus counterbalances the higher energy consumption.

IV. Theoretical Energy Analysis

When considering the current consumption values quoted in Table I, it is obvious that both platforms suited as processors (SunSPOT and Imote2) have a significantly greater energy demand in both active and deep sleep modes than the two sensor node platforms (Mica2 and TelosB). However, the clock frequencies differ by one order of magnitude, hence many more operations can be performed on a processor node within the same amount of time. For the sake of simplicity, we assume an identical number of instructions required to perform the same task on all platforms, although sophisticated features and special extensions to the instruction set present in the processing nodes may lead to deviations.

TABLE II
Execution time and power consumption of the demo method

	Mica2	TelosB	SunSPOT	Imote2
Execution time	13.6 ms	12.5 ms	0.55 ms	0.96 ms
Energy per call	327 µJ	67.6 µJ	82.7 µJ	98.4 µJ
Average power	3.3 mW	0.68 mW	0.85 mW	1.37 mW

A. Execution Duration

To visualize the impact of the clock speed, we have assumed a demo method of 100,000 instructions and evaluated the time and energy required to execute it. The corresponding results for a single call are shown in Table II. Additionally, the table contains the results from our analysis of the overall power consumption when calling the method 10 times per second and immediately putting the MCU into sleep mode when the method has finished.

Although both processor node platforms require between 22 and 45 percent more energy to perform the operation, their benefit of a 32-bit architecture and the corresponding reduced emulation demand for complex algorithms is expected to counterbalance the additional energy requirements. Additionally, the average power consumption of the SunSPOT is only 25% higher than the TelosB's when duty-cycling the node, and put into perspective when considering the achievable savings in terms of the overall network traffic.

B. Node Lifetimes

Having determined a comparable energy demand to perform the same algorithms on the more powerful processing platforms, it has become clear that a WSN can benefit from the use of heterogeneous nodes. However, to ensure a long network lifetime, processors should not deplete their batteries faster than the remaining nodes in the network. When continuously operating SunSPOT nodes with a battery capacity of 750 mAh, their lifetime is limited to around nine hours. In contrast, when assuming a duty cycle of only 10% (i.e. spending 90% of the time in sleep mode), lifetime increases to 93 hours, and when activity phases are limited to 2%, the overall node lifetime extends to 16.5 days. It is thus mandatory to find algorithms which achieve a tradeoff between energy and delay constraints, considering the costs of local computation, in-network processing, or the transfer to the external sink in their decision process.

V. Conclusion and Outlook

In this paper, we have presented the benefits of heterogeneous sensor networks, comprising nodes with different computational capabilities. By adding nodes with higher computational performance to a WSN, complex tasks can be performed within the network instead of transferring all data to an external sink node. Although the faster processor nodes exhibit an increased energy consumption, we have theoretically shown that energy savings can be achieved by deploying processor nodes, as their greater energy consumption is counterbalanced by reduced execution times and less traffic in the network.

A. Future Work

In successive work, we target to investigate deployment strategies for the processor nodes and conduct practical experiments with heterogeneous sensor networks, based on our TWiNS.KOM testbed, which integrates TelosB and SunSPOT devices [6]. We also intend to evaluate the applicability of the developed algorithms on networks that are heterogeneous in terms of energy.

ACKNOWLEDGEMENTS

We would like to thank Parag S. Mogre for the fruitful discussions and his constructive contributions to this paper.

REFERENCES

[1] G. J. Pottie and W. J. Kaiser, "Wireless Integrated Network Sensors," *Communications of the ACM*, vol. 43, no. 5, 2000.

[2] B. Krishnamachari, D. Estrin, and S. Wicker, "The Impact of Data Aggregation in Wireless Sensor Networks," in *Proceedings of the International Workshop on Distributed Event-Based Systems (DEBS)*, 2002.

[3] T. Arici, B. Gedik, Y. Altunbasak, and L. Liu, "PINCO: A Pipelined In-Network COmpression Scheme for Data Collection in Wireless Sensor Networks," in *Proceedings of the 12th International Conference on Computer Communications and Networks (ICCCN)*, 2003.

[4] *TelosB Datasheet*, Crossbow Technology, http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/TelosB_Datasheet.pdf.

[5] Sun Microsystems Inc., "Project SunSPOT - Sun Small Programmable Object Technology," Online: http://www.sunspotworld.com, 2008.

[6] A. Reinhardt, M. Kropff, M. Hollick, and R. Steinmetz, "Designing a Sensor Network Testbed for Smart Heterogeneous Applications," in *Proceedings of the Third IEEE International Workshop on Practical Issues in Building Sensor Network Applications (SenseApp)*, Oct 2008.

[7] G. Sharma and R. Mazumdar, "Hybrid Sensor Networks: A Small World," in *Proceedings of the Sixth ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, 2005.

[8] W. Hu, V. N. Tran, N. Bulusu, C.-T. Chou, S. Jha, and A. Taylor, "The Design and Evaluation of a Hybrid Sensor Network for Cane-toad Monitoring," in *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks*, 2005.

[9] G. Wagenknecht, M. Anwander, T. Braun, T. Staub, J. Matheka, and S. Morgenthaler, "MARWIS: A Management Architecture for Heterogeneous Wireless Sensor Networks," in *Proceedings of the 6th International Conference on Wired/Wireless Internet Communications (WWIC)*, 2008.

[10] X. Wang, A. Jiang, and S. Wang, *Advances in Intelligent Computing*. Springer Berlin/Heidelberg, 2005, vol. 3645/2005, ch. Mobile Agent Based Wireless Sensor Network for Intelligent Maintenance.

[11] M. Yarvis, N. Kushalnagar, H. Singh, A. Rangarajan, Y. Liu, and S. Singh, "Exploiting Heterogeneity in Sensor Networks," in *Proceedings of the 24th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, 2005.

[12] G. Smaragdakis, I. Matta, and A. Bestavros, "SEP: A Stable Election Protocol for Clustered Heterogeneous Wireless Sensor Networks," in *Proceedings of the 2nd International Workshop on Sensor and Actor Network Protocols and Applications (SANPA)*, 2004.

[13] *MICA2 Datasheet*, Crossbow Technology, http://www.xbow.com/products/Product_pdf_files/Wireless_pdf/MICA2_Datasheet.pdf.

[14] L. Nachman, J. Huang, J. Shahabdeen, R. Adler, and R. Kling, "IMOTE2: Serious Computation at the Edge," in *Proceedings of the Wireless Communications and Mobile Computing Conference (IWCMC)*, 2008.

[15] C. M. Sadler and M. Martonosi, "Data Compression Algorithms for Energy-Constrained Devices in Delay Tolerant Networks," in *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems (SenSys)*, 2006.

[16] A. Reinhardt, M. Hollick, and R. Steinmetz, "Stream-oriented Lossless Packet Compression in Wireless Sensor Networks," in *Accepted for Publication in: Proceedings of the Sixth Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*, 2009.

[17] N. Gura, A. Patel, A. Wander, H. Eberle, and S. C. Shantz, "Comparing Elliptic Curve Cryptography and RSA on 8-Bit CPUs," in *Cryptographic Hardware and Embedded Systems (CHES)*. Springer Berlin/Heidelberg, 2004.

[18] Texas Instruments Inc., "CC2420: 2.4 GHz IEEE 802.15.4 / ZigBee-Ready RF Transceiver (Rev. B)," 2007. [Online]. Available: http://www.ti.com/lit/gpn/cc2420

Enhanced Performance by Sub-1 GHz WSN Solutions based on IEEE 802.15.4-2006

Andreas C. Wolf
Managing Director
Dr. Wolf Wireless GmbH
Teltow, Germany
aw@dw-w.com

***Abstract*—PSSS (Parallel Sequence Spread Spectrum) [1] technology is the basis for the PHY of the new IEEE802.15.4-2006 standard with the enhancement for the data rate from 20 kbps to 250 kbps for the European area. Even robustness against multipath fading is enhanced and makes the sub 1 GHz PHY high attractive, even due to less inference probability, compared to 2.4GHz solutions and lower attenuation in the transmission path.**

I. INTRODUCTION

The sub 1 GHz PHY's of IEEE 802.15.4-2003 standard offer only 20 kbps for Europe/ETSI (European Telecommunications Standards Institute) and 40 kbps for the FCC region. Compared to the 250 kbps the data rate was unattractive, especially for WSN (Wireless Sensor Networks) with many subscribers. For the ETSI region has t be taken into account, that there is a duty cycle limitation of 1%. That causes average data rate of not more then 200 bps for the IEEE 802.15.4-2003 PHY. The peak data rate for the sub 1 GHz IEEE820.15.4-2006 PHYs (ETIS/FCC) is 250 kbps common to the 2.4 GHz PHY.

The coverage is for sub 1GHz bands less then for the 2.4 GHz band. Simulations with a ray tracing tool underline that fact. In figure 1 is shown the received power for a 2.4 GHz transmission for a LOS (Line of Sight) and a NLOS (No Line of Sight) area.

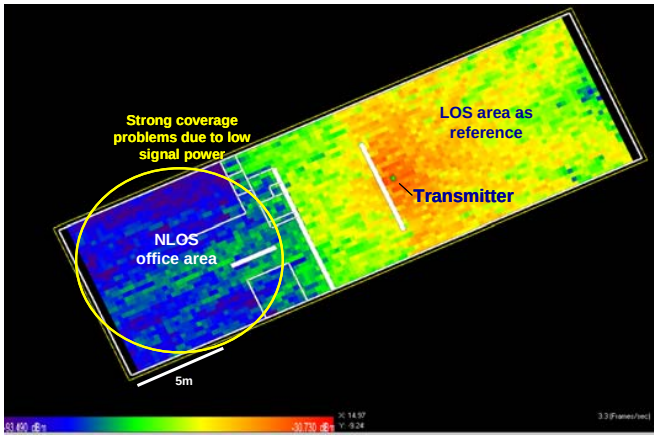


Figure 1. Coverage 2.4 GHz in LOS and NLOS area. Received power: blue -93,5 dBm, red -30 dBm

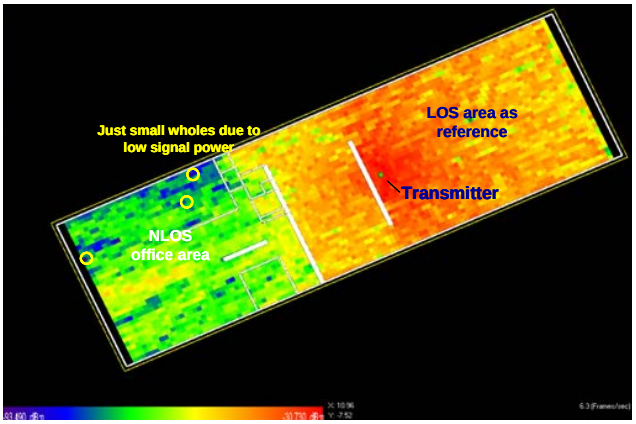


Figure 2. Figure 1. Coverage 868 MHz in LOS and NLOS area. Received power: blue -93,5 dBm, red -30 dBm

The 868 MHz example in figure 2 shows that the received power is significant higher. Even the expected interference is in the sub 1 GHz better then in the 2.4 GHz band. WLAN and Bluetooth are occupying the 2.4 GHz band.

The motivation for enhancing the data rate for the sub 1 GHz PHY in the IEEE 802.15.4-2006 standard was to combine the attractive coverage of the sub 1 GHz band and the low interference with the high data rate of the 2.4 GHz PHY. Especially for the ETSI area with the 1 % duty cycle limitation the increased data rate was necessary.

II. PSSS TECHNOLOGY

A. *Basic*

PSSS uses for the encoding m-sequences in parallel. Equitation (1) describes the base m-sequence ms_1.

$$ms_1 = (m_{11}, m_{21}, ..., m_{M1}) \quad (1)$$

The coding table is given by EN and contains cyclic shifted m-sequences of ms_1.

$$EN = \begin{bmatrix} m_{11} & \dots & m_{1N} \\ m_{21} & \dots & m_{2N} \\ m_{M1} & \dots & m_{MN} \end{bmatrix} \quad (2)$$

For the encoding the data D (3) is multiplied with EN (2).

$$D^T = (d_1, d_2, ..., d_N) \quad (3)$$

$$S = EN \bullet D \quad (4)$$

Each data bit of D is spread with a cyclic shifted m-sequence. The spreaded bit are the added column wise. The decoding can be reached by cyclic cross correlating the PSSS-Symbol S with the base m-sequence ms_1. This operation is similar to using a matrix DE for decoding.

$$DE = EN^T \quad (5)$$

$$CCF = S \bullet DE \quad (6)$$

CCF presents the cyclic cross correlation between the PSSS symbol S and the decoder matrix DE. The reconstruction is done by threshold decision as described in (7).

$$d'_n(ccf_n) = \begin{cases} d'_n = 0;\ ccf_n \le (Max\{CCF\} + Min\{CCF\} \div 2) \\ d'_n = 1;\ ccf_n > (Max\{CCF\} + Min\{CCF\} \div 2) \end{cases} \quad (7)$$

$d'_n(ccf_n)$ is the reconstructed data word. Depending on implementation targets of PSSS different threshold algorithms are available.

For reducing the PAPR (Peak Average Power Ratio) and the DC component of the PSSS symbol S precoding could be used. The precoding of one symbol is executed independent of the precoding of any other symbol with the two steps described mathematically as follows:

$$S'(m) = S(m) + \frac{(Max + Min)}{2}$$

where S(m) is the current PSSS symbol and S'(m) is the aligned symmetric to zero PSSS symbol and Max and Min are the maximum and minimum chip amplitudes within the symbol respectively and

$$p''(m) = \frac{p'(m)}{A}$$

where A = (Max' – Min') and Max' and Min' are the maximum and minimum chip amplitudes within the aligned symmetric to zero PSSS symbol p'(m) respectively.

Precoding reduces the PAPR and therefore the demanded linearity of the power amplifier.

B. PSSS for the IEEE 802.15.4-2006 sub 1 GHz PHYs

Target for the new standard was [2] to reach 250 kbps even for the sub 1 GHz PHY. For the PHYs as base m-sequence was selected a 31 chip log sequence. From the resulting encoding matrix only a subset has been selected. Available are 31 cyclic shifted sequences. For FCC only five and for ETSI twenty sequences have been selected. That causes that for the given chip rate a data rate of 250 kbps is realized, both FCC and ETSI version of the PHY.

Selecting a subset of EN causes even that the distance between the correlation peaks of CCF (6) increases, which could be used for enhanced multipath fading robustness. The delayed multipath fading parts of the received signal are between the correlation peaks and don't cause ISI (Inter Symbol Interference), if the delay spread is shorter the distance between the CCF peaks.

For avoiding hurting the cyclic correlation for the decoder due to multipath fading the PSSS symbol S is cyclic extended similar to the cyclic extension of OFDM symbols. The extend PSSS symbol contains 32 chips.

III. PERFORMANCE OF PHY IMPLEMENTATIONS AND AVAILABLE PLATFORM

For the ESTI an FCC PHYs of IEEE802.15.4-2006 are discrete FPGA based implementations available that have a sensitivity of less then -100dBm for 1% PER. The available link budget is about 120 dB or even more. Even in [3] a implementation as single chip is described.

IV. FUTURE STEPS

PSSS as technology is usable for WSN due to low power consumption and low cost implementations combine with unique data rate of 250 kbps for the ETSI region of the IEEE 802.15.4-2006 standard. The low complexity of PSSS implementations is opening the path to high data rate solutions, where OFDM implementations are limited in the reachable data rate. PSSS can be combined with well know technologies like deconvolution and MIMO for enhancing the multipath fading robustness. Even combination of PSSS and OFDM seam to be promising [4].

REFERENCES

[1] A. Wolf, "Patentschrift zu PSSS", 2003.

[2] H. van Leeuwen and A. Wolf, IEEE15-04-0121-03-004b, March 2004.

[3] Trung-Kien Nguyen et al., "Low-Power Direct Conversion Transceiver for 915 MHz Band IEEE 802.15.4b Standard Based on 0.18 µm CMOS Technology", ETRI Journal, Volume 30, Number 1, February 2008, pp. 33-45.

[4] Paulo Isagani M. Urriza and Joel Joseph S. Marciano Jr., "A Flexible OFDM Spread Spectrum System Using Parallel Sequences", http://eee.upd.edu.ph/urriza/research.html.

The development of the discrete prototypes is granted by Europäische Union, Europäischer Fond für regionale Entwicklung. Investition in Ihre Zukunft. Allgemeine Informationen unter www.efre.brandenburg.de.

-SomSed-
The Evolution of an Experimental Wireless Sensor Network Towards a Research Platform

Felix Hackbarth[1], Thomas Meyerhoff[2], Harald Sauff[3], Bryce T. Bradford[4], Luis Torres[5], Helge Klimek[6], Björn Gressmann[6], Christian Renner[6], Martin Stemick[2], Sebastian Georgi[2], Christoph Weyer[6]
[1]Institute of Automation, [2]Institute of Telecommunications, [3]Institute of Security in Distributed Applications, [4]Institute of Microsystems Technology, [5]Institute of Communication Networks, [6]Institute of Telematics, Hamburg University of Technology
Email: https://somsed.tu-harburg.de/junior/phdstudents

***Abstract*—The exploratory focus of the SomSed research field is the interdisciplinary research on self organizing mobile sensor and data networks. Since the founding of SomSed in 2007, great progress in scientific research has been achieved and much practical knowledge has been gathered using a prototype network permanently installed. This prototype network, from hereon referred to as CampusNet, is the basis for further investigations and offers the possibility to perform long term measurements in a large scale and real environment. The scope of this paper is to outline the current status of the SomSed research field and to briefly discuss future developments.**

I. Introduction

Self organizing mobile Sensor and data networks (SomSed) is a research field at the Hamburg University of Technology. While the cooperation among the professors of the institutes is common, doing research in cooperation between undergraduates and Ph.D. students of different institutes is rare. As announced in a previous paper [1] the institutes work together in a matrix like organization structure on topics concerning wireless sensor networks. In doing so, the institutes can concentrate on their core competences concerning this research field. The unique collaboration of several institutes forms a broad basis for research.

The Ph.D. students branch of SomSed focuses on their own special research topics and implemented a wireless sensor network on the campus of the Hamburg University of Technology. The cooperation on undergraduate and Ph.D. student level also profits from this approach and leads to additional synergy effects and knowledge transfer between the collaborating institutes.

The institutes themselves use the knowledge gained in SomSed. For example, experiences gained in SomSed are used to build up sensor networks for cruise and container vessels, and doing feasibility studies of using 2.4GHz applications in these environments [2]. Another approach is to investigate multi-coverage based broadcasting in order to increase reliability in a wireless sensor network, as presented in [3]. In this approach an Integer Linear Program (ILP) has been applied to multimedia data transmission inside an aircraft passenger cabin. The solution provides compact routing and scheduling of the relaying nodes.

II. CampusNet

During the last year SomSed-Active developed and deployed an experimental wireless sensor network on the campus referred to as CampusNet. The CampusNet consists of 26 fixed nodes of type IRIS from the company Crossbow Technology [4]. The nodes are based on an ATmega1281 microprocessor with an integrated 2.4 GHz IEEE 802.15.4 radio transceiver. The nodes run the open source operating system TinyOS version 2.x.

Before the construction of the CampusNet started a series of open field measurements of connectivity and signal strength of the IRIS nodes have been carried out. The results of these measurements were used to find an adequate placement in terms of connectivity for the participating nodes. The placement of the nodes is shown in Fig. 3 using small circles.

The software for the CampusNet can be divided into three parts: The sensor node firmware, which is responsible for routing, tree construction, sensing, power management and data buffering, and the frontend and backend software. The backend software just persists incoming data from the sensor network into a database and is the initiator of regularly occurring tree constructions. The frontend software is used for analysis and visualization of the stored information. The routing mechanism and the frontend software are described in more detail in the following sections.

A. Routing

The purpose of this network was at first to collect data about signal strengths and link quality for validation of a wireless sensor network of this dimension. Since this stage only measurements are to be obtained, a tree routing structure was implemented (see Fig. 3). Therefore there is only one sink in the network that writes all received packets into a database.

In order to fulfill the need of a long living or real time network, the CampusNet supports two operation modes: one active and one passive mode. In active mode measured data

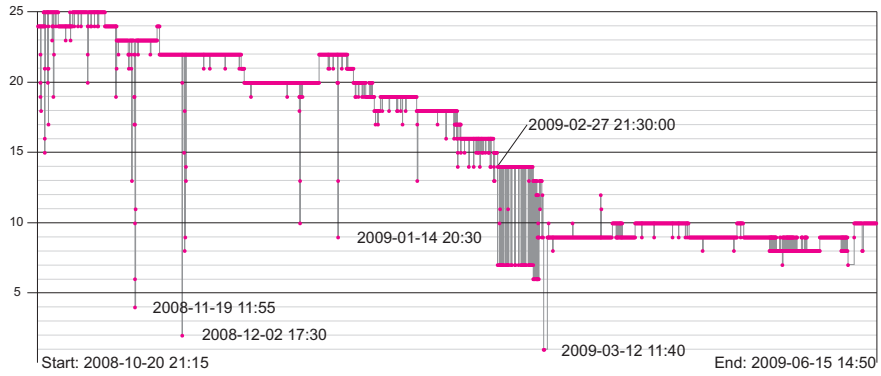


Fig. 1. Nodes participating in the tree over time.

will be forwarded on continuously in real time. This results in high energy consumption, because the transceivers of the nodes will be permanently switched on. In passive mode the participating nodes perform wake-sleep intervals of 15 minutes. After waking up, they send their own data and forward the packets received from other nodes to nodes closer to the sink. Then they go back to sleep mode in order to preserve battery power to extend their lifetime.

In either mode, a routing tree is generated every hour to determine the best route to the sink in terms of signal strength and link quality. This is necessary, because environmental conditions may change rapidly. For example, around lunch times the canteen is densely populated and many people are using wireless LAN. This can cause a reduction of the link quality between nodes nearby and has to be compensated by the routing algorithm. The generation of a new routing tree is initiated by a tree construction packet sent by the sink. This packet contains the type of the actual mode, a timestamp during which the new tree is valid and the local time of the data sink. The nodes receiving this packet use it to adjust their clock and to determine their parent node. If a node receives several tree construction packets sent by different nodes, the node with the highest received signal strength indicator (RSSI) is chosen as parent. By receiving several tree construction packets the nodes are capable of repairing a tree if one connection fails. If this is the case, an alternative parent is chosen. The connection parameters like link quality and signal strength and the clock synchronization offset of each node are recorded into the database. The collected data is currently being analyzed.

B. Frontend

In order to have a useful visualization and to store data measured by the CampusNet a web-based frontend was developed. This web-based frontend of the CampusNet is accessible via `http://www.sva.tu-harburg.de/~somsed/website/`. The Google-maps-API is used to show the positions of the nodes and the actual routing tree. A screenshot of the map showing the CampusNet is shown in Fig. 3. When a specific element is selected, an information frame shows all measured data of that selected element. For example, selecting a node displays several graphs showing its measured data, while selecting a tree branch displays its link quality indicator (LQI) and received signal strength indicator and packet success rate (PSR). Using a time shift function different points in the lifetime of the CampusNet can be visited and the different created trees can be seen in time lapse. If mobile nodes are participating in the network, their motion paths are also shown on the map.

The basis of the frontend is a database where all the measured data and statistics are stored.

Fig. 2. Solar power module for an iris node with emergency battery, solar panel, electronics with node adapter and rechargeable lead batteries.

III. Hardware Developments

In the last year several hardware developments have been made by SomSed-Active that added functionality to the CampusNet.

A. Solar Node

The solar node developed was subject of a diploma thesis [5]. The goal was the design of a cost effective and robust solar module, which is able to power the whole sensor node even under unfavorable weather conditions.

Therefore, a solar module with high efficiency and power rating had to be found. In order to bridge the gap of the solar power supply during nights and bad weather conditions, an additional battery charging system has been developed, too.

As a result of [5], a prototype of the solar node has been built and was successfully integrated into the CampusNet. The concept demonstrated its robustness over the past eight months until today, in spite of seasonal low solar radiation during the winter months. The developed module for the node is shown in Fig. 2.

Because of the success of the solar modules and the advantages of an independent power supply, ten more solar nodes are currently being built. Their deployment will result in a small energy autonomous sensor network, which will be subject of further investigations.

B. Over the Air Programming

Sensor nodes are usually programmed using an interface device which is connected to a computer. Once the sensor nodes are programmed and deployed at their locations, it is often difficult to install new software. Sensor nodes may be deployed in high altitudes, hazardous areas, or out of reach for other reasons. In these cases, programming sensor nodes becomes a significant challenge. Over the Air Programming (OTAP) is a technique which aims at eliminating this problem. It reduces the effort of maintaining a wireless sensor network, because the nodes do not have to be physically accessible for programming or maintenance.

An own OTAP module was developed for the CampusNet, as described in [6]. It provides an intuitive user interface and offers a set of important features:

- Several images on a node: Every sensor node is able to carry different program images from which one can be chosen for execution.
- ID of image name and version: IDs allow to identify and distinguish program images.
- Check OTAP compatibility: Safeguards have been implemented in order to prevent non-OTAP images to be written to a node.
- Golden image: In case of error, a fall-back image will be selected for execution, providing basic maintenance capabilities.
- Easy setup of nodes for OTAP: Once a node has been prepared with the initial image no further wired communication is necessary.
- Support for heterogeneous networks: The OTAP capabilities can simply be integrated in different programs.

The capability to store more than one image on a node provides the opportunity to easily switch between different software functionality. The configuration of the whole network can be changed with a minimum of effort. In case of an erroneous image, the golden image will be automatically selected as the working image to reestablish OTAP-functionality for reprogramming.

C. 4-Sensor-Board

The 4-Sensor-Board is an extension to the IRIS node. It measures temperature, ambient light intensity, relative humidity, and air pressure. An integrated solar power generation module keeps the on board rechargeable batteries charged. The task was to perform continuous measurements of the described parameters in the CampusNet. The board was designed to meet these measurement requirements, while also operating with very low power consumption. As an example: while sampling the environment once every ten seconds and transmitting the data every 40 seconds the average power consumption is 174μW, which corresponds to a battery life of more than four years.

IV. Experience gained with the CampusNet

Since the deployment of the CampusNet among others, almost 360.000 database entries containing measured sensor data have been stored. More than 820.000 entries containing neighborhood information have been recorded, each consisting of link quality indicator, received signal strength indicator, packet success rate, the number of duplicate packets, the amount of missed packets and the number of received packets. The data traffic at the sink was also documented. Given the recorded information, it is possible for any time in the past to reconstruct the trees that were created, their link properties and the sensor data. As already mentioned, this data is currently being analyzed with focus on the clockdrift-temperature and link quality-environment dependencies.

During the last months several lessons have been learned: The tree routing structure caused congestion of data packets near the sink. The reason is the increased channel utilization

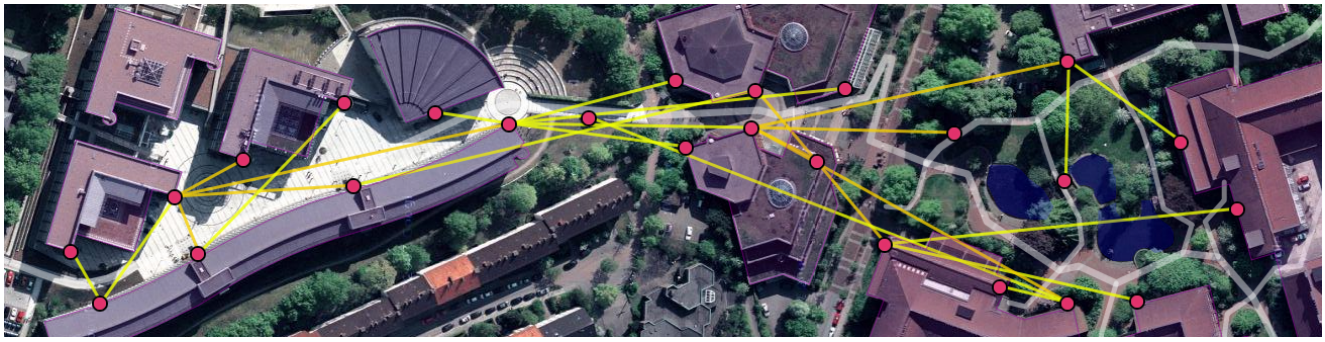


Fig. 3. Routing tree on the CampusNet on Nov. 1st 2008 0:00 am.

while data packets are traveling from a leaf node through the tree.

Another aspect is the asymmetry of links between nodes. More precisely, the tree is constructed taking the highest quality links downwards from the sink to the leaves. But the link quality upwards the same connection is not necessarily of the same quality. This may result in packet loss.

There were also problems with the casing of the nodes. These had the protection category of IP55. As it turned out, the bushings were not sealed against water intrusion. To add additionally waterproofing to the case, silicone was used as gasket. But acetic acid was exhaling from the silicone causing corrosion of some of the electronics. Moreover the temperature and humidity variation in combination with direct sunlight on the case and nightly temperature drops caused condensation that dripped onto the nodes electronics. Over the last months, the water in the cases only caused some node losses. After drying, these nodes were still fully functional.

For the next long-term deployment these problems have been solved.

Fig. 1 shows the nodes and node losses over time and also the stability of the trees. For example from February 27th on, the number of nodes in the tree varied from 14 to 7. This is a result of the elongated region that is covered by the CampusNet. If regions are connected through few bottleneck nodes, the tree is very sensitive to their failure. In the CampusNet, this bottleneck caused the cutoff of the lower campus (in Fig. 3 right part of the network).

V. Work in Progress

Currently the CampusNet is deployed and collecting data. The results of the evaluation of the collected data will be used to set up the next generation of the CampusNet.

The long term goal for the next generation wireless sensor network is to develop a platform for scientific research for students and staff of the university. Consequently, a modular design is currently being developed, which allows fast realization of research projects.

This design consists of the following modules:

- Extended OTAP-Module
- MAC-Layers
- Network-Layers
- TinySec-Security-Module
- Sensor-Interface-Module

The different modules form building blocks which are used to combine them into a node firmware which provides the infrastructure for individual research projects.

In general, it is desirable to not congest the network tree with additional data during experiment runs. On the other hand, it is often necessary to collect logging and performance data for later evaluation.

To meet these requirements, OTAP is enhanced with data persistence functionality. The new functionality enables a node to collect logging data for debugging. Aside from logging, the module can be used for structured data persistence of other data as well. Relatively large amounts of data can be stored on the node for later processing.

OTAP will use the large memory space which is also used to store program images to store its data. This allows for flexible assignment of nodes, depending on whether data collection assignments are expected.

The collected data will be transferred wirelessly like program images once experiment runs have finished. It is also planned to provide the developer with tools that help translating logging data, stored in compact form on the node, to human readable text.

A nodes firmware can be built to choose out of a set of MAC- and Network-Layer modules.

Additionally, a security layer can be included to enable secure communication.

Finally, a generalized sensor module interface provides uniform access to sensors attached to the nodes.

Since the network configuration can be switched fast from one application to another, easy sharing of the resource CampusNet among the participating institutes is possible.

References

[1] S. Georgi, C. Weyer, M. Stemick, C. Renner, F. Hackbarth, U. Pilz, J. Eichmann, T. Pilsak, H. Sauff, L. Torres, K. Dembowski, and F. Wagner, "Somsed: An interdisciplinary approach for developing wireless sensor networks," 7. GI/ITG KuVS Fachgespräch Drahtlose Sensornetze, Berlin, Germany, Tech. Rep. B 08-12, 2008.

[2] T. Pilsak and J. ter Haseborg, "Emc feasibility study of the use of 2.4-ghz-wlan applications on bridges of cruise and container vessels," in *Electromagnetic Compatibility, 2008. EMC 2008. IEEE International Symposium on*, Detroit, MI, USA, Aug. 2008, pp. 1–6.

[3] L. Torres and U. Killat, "Routing and scheduling for short-range wireless inflight multimedia networks," in *Proceedings of the 2nd International Workshop on Aircraft System Technologies (AST 2009)*, O. von Estorff and F. Thielecke, Eds. Hamburg, Germany: Shaker Verlag, Aachen, Mar. 2009, pp. 337–346.

[4] "Crossbow Technology," http://www.xbow.com/Home/wHomePage.aspx.

[5] C. Lange, "Energiegewinnung für drahtlose Sensorknoten," Master's thesis, Hamburg University of Technology, Hamburg, Germany, Oct. 2008.

[6] M. Stemick, A. Boah, and H. Rohling, "Over-the-air programming of wireless sensor nodes," 7. GI/ITG KuVS Fachgespräch Drahtlose Sensornetze, Berlin, Germany, Tech. Rep. B 08-12, 2008.

Wireless Sensor Networks and the Internet of Things: Selected Challenges

Delphine Christin, Andreas Reinhardt, Parag S. Mogre, Ralf Steinmetz
Multimedia Communications Lab, Technische Universität Darmstadt
Merckstr. 25, 64283 Darmstadt, Germany
{*delphine.christin, andreas.reinhardt, parag.mogre, ralf.steinmetz*}*@kom.tu-darmstadt.de*

***Abstract*—Wireless sensor networks (*WSNs*) are increasingly gaining impact in our day to day lives. They are finding a wide range of applications in various domains, including health-care, assisted and enhanced-living scenarios, industrial and production monitoring, control networks, and many other fields. In future, WSNs are expected to be integrated into the "Internet of Things", where sensor nodes join the Internet dynamically, and use it to collaborate and accomplish their tasks. However, when WSNs become a part of the Internet, we must carefully investigate and analyze the issues involved with this integration. In this paper, we evaluate different approaches to integrate WSNs into the Internet and outline a set of challenges, which we target to address in the near future.**

I. Introduction

The future Internet, designed as an "Internet of Things" is foreseen to be "a world-wide network of interconnected objects uniquely addressable, based on standard communication protocols" [1]. Identified by a unique address, any object including computers, sensors, RFID tags or mobile phones will be able to dynamically join the network, collaborate and cooperate efficiently to achieve different tasks. Including WSNs in such a scenario will open new perspectives. Covering a wide application field, WSNs can play an important role by collecting surrounding context and environment information. However, deploying WSNs configured to access the Internet raises novel challenges, which need to be tackled before taking advantage of the many benefits of such integration.

The main contributions of this paper can be summarized as follows: We look at WSNs and the Internet holistically, in line with the vision where WSNs will be a part of an Internet of Things. Thereby, we identify representative application scenarios for WSNs (see Section II) from the multidimensional WSN design space [2], in order to obtain insights into issues involved with the integration. These representative application scenarios open up different schemes for integrating the WSNs into the Internet, which we present and compare in Section III. A closer investigation of the integration possibilies then helps us identify critical challenges (see Section IV), which need to be addressed if the full potential of the integration of WSNs and the Internet has to be realized. Finally, in Section V we summarize our discussion, giving pointers for possible solutions to address the identified challenges while regarding the resource limitations present in common WSN nodes.

II. Selected WSN Applications

The wide wireless sensor network application field can be divided into three main categories according to [3]: Monitoring space, monitoring objects and monitoring interactions between objects and space. The proposed classification can be extended by an additional category monitoring human beings.

One example of the first category is environmental monitoring. WSNs are deployed in particular environments including glaciers [4], forests [3], and mountains [5] in order to gather environmental parameters during long periods. Temperature, moisture or light sensor readings allow analyzing environmental phenomena, such as the influence of climate change on rock fall in permafrost areas [5].

The second category centers on observing particular objects. Structural monitoring is one of the possible illustrations of this category. By sensing modes of vibration, acoustic emissions and responses to stimuli, mechanical modifications of bridges [6] or buildings [7] indicating potential breakages of the structure may be detected.

Monitoring interaction between objects and space is the combination of both previous categories and includes monitoring environmental threats like floods [8] and volcanic activities [9].

Presenting an extension to the presented classification, the last category focuses on monitoring human beings. Worn close to the body, the deployed sensors can gather acceleration information and physiological parameters like heart beat rate. Especially in applications in the medical area, such deployments may help diagnosing bipolar patients [10] and monitoring elderly people in a home care scenario [11].

The proposed classification, and particularly the selected deployments, illustrate the high diversity of WSN applications in term of monitored subjects and environments. Beneficial for the Internet of Things, this important scenario diversity must however be taken into account by considering suitable approaches for the WSN integration into the Internet.

III. Integration Approaches

Connecting WSNs to the Internet is possible in the three main approaches mentioned by [12], differing from the WSN integration degree into the Internet structure. Currently adopted by most of the WSNs accessing the Internet, and

presenting the highest abstraction between networks, the first proposed approach (Fig. 1) consists of connecting both independent WSN and the Internet through a single gateway.

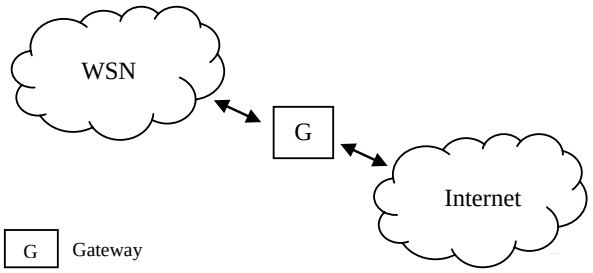


Fig. 1. Independent network

Showing an increasing integration degree, the second approach (Fig. 2) forms a hybrid network, still composed of independent networks, where few dual sensor nodes can access the Internet.

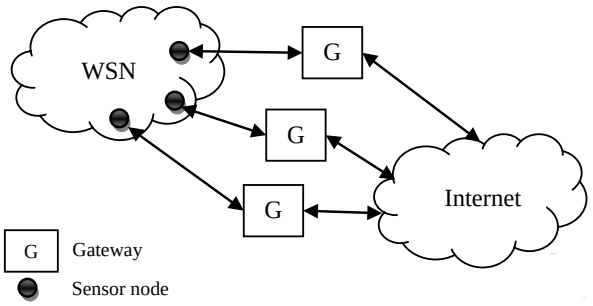


Fig. 2. Hybrid network

Illustrated by Fig. 3, the last approach is inspired from current WLAN structure and forms a dense 802.15.4 access point network, where multiple sensor nodes can join the Internet in one hop.

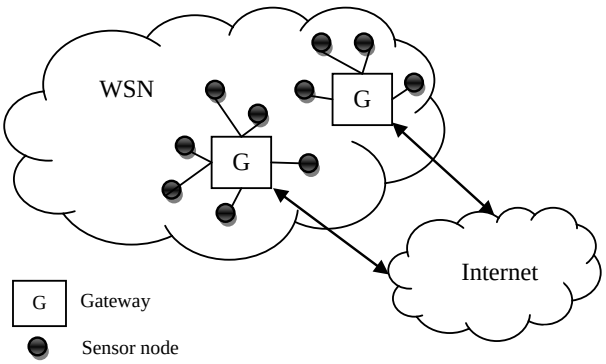


Fig. 3. Access point network

It is obvious that the first approach presents a single point of failure due to the gateway uniqueness. Gateway dysfunction would break down the connection between both WSN and the Internet. With several gateways and access points, the second and third scenarios do not present such weakness. To ensure network robustness, they would consequently be preferred, if the application supports this type of network structure.

The choice between both remaining integration approaches is influenced by the WSN application scenario. Allowing to cover important distances, the second approach can be envisaged for WSNs organized in mesh topology. Accordingly, this approach would be particularly adapted to deployments belonging to the first "Monitoring space" and the second "Monitoring interactions between objects and space" categories previously introduced in the proposed application classification. By offering Internet access in one-hop, the third and last approach can be adopted by WSN applications requiring low latency and therefore direct connections. Presenting mainly star topologies, WSNs can maintain such organization by having a central gateway instead of a common base station without Internet access. By considering the previous WSN application classification, this third approach can be suitable for monitoring of object and human beings, and may be employed in the [6], [7], [10], [11] deployments for example.

It is important to remark that both second and third integration approaches only support static network configurations. Indeed, each new device wanting to join the Internet requires time-consuming gateway reprogramming. Therefore, the flexibility wanted by the future Internet of Things cannot be achieved by both approaches in their current form.

To fulfill the flexibility expectation, adopting the "IP to the Field" paradigm [13] may be appropriate. In the considered paradigm, sensor nodes are expected to be intelligent network components, which will no more be limited to sensing tasks. By transferring the intelligence to the sensor nodes, the gateway's functionalities would be restricted to forwarding and protocol translation. Consequently, gateway reprogramming operations would no more be required and dynamic network configuration could be attained. Additionally, this shift of intelligence will open new perspectives including geographic-based addressing for example.

IV. Challenges for WSNs in an Internet of Things

The formerly introduced "IP to the Field" paradigm involves assigning additional responsibilities to sensor nodes in addition to their usual sensing functionality. To highlight and discuss the challenges emerging from such novel responsibility assignment, we selected three potential tasks that the sensor nodes would have to accomplish: Security and quality of service *(QoS)* management, and network configuration.

A. Security

In common WSNs without Internet access, the sensor nodes may already play an important role to ensure data confidentiality, integrity, availability and authentication depending on the application sensitivity. However, the current identified attack scenarios require a physical presence near the targeted WSN in order to jam, capture or introduce malicious nodes for

example. By opening WSNs to Internet, such location proximity will no more be required and attackers would be able to threaten WSNs from everywhere. In addition to this novel location diversity, WSNs may have to address new threats like malware introduced by the Internet connection and evolving with the attacker creativity. Most current WSNs connected to the Internet are protected by a central and unique powerful gateway ensuring efficient protection. However, a direct reuse of such existing security mechanisms is made impossible by the scarce energy, memory, and computational resources of the sensor nodes. In fact, common Mica2 motes offer 7.3 MHz 8-bit microcontrollers with 128 Kbytes of reprogrammable flash memory, 4 Kbytes of RAM and 4 Kbytes of EEPROM [14]. At last, many services on the Internet make use of cryptography with large key lengths such as RSA-1024, which are not currently supported by sensor nodes. Consequently, innovative security mechanisms must be developed according to the resource constraints to protect WSNs from novel attacks originating from the Internet.

B. Quality of Service

With gateways acting only as repeater and protocol translators, sensor nodes are also expected to contribute to quality of service management by optimizing the resource utilization of all heterogeneous devices that are part of the future Internet of Things. Not considered as a weakness, the device heterogeneity opens new perspectives in terms of workload distribution. In fact, resource differences may be exploited to share the current workload between nodes offering available resources. Improving the QoS, such collaborative work is consequently promising for mechanisms requiring high amount of resources like security mechanisms. Nevertheless, the existing approaches ensuring QoS in the Internet are not applicable in WSNs, as sudden changes in the link characteristics can lead to significant reconfiguration of the WSN topology. It is therefore mandatory to find novel approaches towards ensuring delay and loss guarantees.

C. Configuration

In addition to security and QoS management, sensor nodes can also be required to control the WSN configuration, which includes covering different tasks, such as address administration to ensure scalable network constructions and ensuring self-healing capabilities by detecting and eliminating faulty nodes or managing their own configuration. However, self-configuration of participating nodes is not a common feature in the Internet. Instead, the user is expected to install applications and recover the system from crashes. In contrast, the unattended operation of autonomous sensor nodes requires novel means of network configuration and management.

V. Conclusion

In this first analysis step to integrate WSNs into the Internet of Things, we have considered selected application scenarios presenting a high diversity in terms of monitored subjects and environments. By taking into account their main characteristics, we have analyzed three integration approaches and demonstrated that they were inappropriate in their current state to allow sensor nodes joining dynamically the Internet of Things.

We consider applying the IP to the Field paradigm, which implies assigning additional responsibilities to the sensor nodes as an adequate solution to integrate WSNs with the Internet. We have selected three important task assignments in order to highlight the challenges emerging from the paradigm adoption: Security, QoS, and configuration management. Their analysis revealed that the solutions currently deployed in the Internet are not suitable for the limited sensor node resources and consequently, novel mechanisms have to be developed to adapt to the capabilities and constraints of WSNs. We plan to investigate existing approaches and find suitable modifications for resource-constrained sensor platforms to tackle these challenges.

Acknowledgment

This work was supported by CASED (www.cased.de). The authors would like to thank Matthias Hollick for the fruitful discussions.

References

[1] "Internet of Things in 2020: Roadmap for the Future," 2008, online, http://www.smart-systems-integration.org/public/internet-of-things.
[2] K. Römer and F. Mattern, "The Design Space of Wireless Sensor Networks," *Wireless Communications, IEEE*, vol. 11, no. 6, 2004.
[3] D. Culler, D. Estrin, and M. Srivastava, "Guest Editors' Introduction: Overview of Sensor Networks," vol. 37, no. 8, 2004.
[4] K. Martinez, R. Ong, and J. Hart, "Glacsweb: a Sensor Network for Hostile Environments," in *Proceedings of the Sensor and Ad Hoc Communications and Networks Conference (SECON)*, 2004.
[5] I. Talzi, A. Hasler, S. Gruber, and C. Tschudin, "PermaSense: investigating permafrost with a WSN in the Swiss Alps," in *Proceedings of the workshop on Embedded networked sensors (EmNets)*, 2007.
[6] R. Lee, K. Chen, S. Chiang, C. Lai, H. Liu, and M.-S. Wei, "A Backup Routing with Wireless Sensor Network for Bridge Monitoring System," in *Proceedings of the Communication Networks and Services Research Conference(CNSR)*, 2006.
[7] P. Katsikogiannis, E. Zervas, and G. Kaltsas, "A Wireless Sensor Network for Building Structural Health Monitoring and Seismic Detection," *physica status solidi (c)*, vol. 5, 2008.
[8] D. Hughes, G. Blair, G. Coulson, P. Greenwood, B. Porter, P. Smith, and K. Beven, "An Adaptable WSN-based Flood Monitoring System," in *Proceedings of the European Conference on Smart Sensing and Context (EuroSSC)*, 2007.
[9] W. Werner-Allen, K. Lorincz, M. Ruiz, O. Marcillo, J. Johnson, J. Lees, and M. Welsh, "Deploying a Wireless Sensor Network on an Active Volcano," *IEEE Internet Computing*, vol. 10, no. 2, 2006.
[10] M. H. Teicher, "Actigraphy and Motion Analysis: New Tools for Psychiatry," *Harvard Review of Psychiatry*, vol. 3, 1995.
[11] A. Wood, G. Virone, T. Doan, Q. Cao, L. Selavo, Y. Wu, L. Fang, Z. He, S. Lin, and J. Stankovic, "ALARM-NET: Wireless Sensor Networks for Assisted-living and Residential Monitoring," 2006.
[12] R. Roman and J. Lopez, "Integrating Wireless Sensor Networks and the Internet: a Security Analysis," *Internet Research: Electronic Networking Applications and Policy*, vol. 19, no. 2, 2009.
[13] Smart Energy Alliance, online, http://www.smart-energy-alliance.com/solutions/ip-to-the-field/.
[14] Crossbow Technology, online, http://www.xbow.com.

MoNoTrac: A Mobile Node Trace Generator

Bastian Blywis Felix Juraschek Mesut Güneş Christian Graff
Computer Systems and Telematics
Institute of Computer Science
Freie Universität Berlin, Germany
{blywis, jurasch, guenes, graff}@inf.fu-berlin.de

***Abstract*—Mobility is a core feature of future networks, e.g., wireless sensor, wireless mesh, and mobile ad-hoc networks. Thus the ability to generate accurate traces of mobile nodes is an important aspect for wireless network research. Many publications regarding wireless networks rely on simulations. The applied mobility models are often highly abstract and emulate human behavior poorly. Graph-based approaches try to restrict the area of movement to street-like structures yet they do not model real environments. MoNoTrac is a work-in-progress framework to create mobility traces based on real maps provided by *OpenStreetMaps*. Its plug-in architecture allows the usage of custom mobility models and provides simplified access for research in the domain of mobile networks.**

***Index Terms*—Mobile Ad-Hoc Network (MANET), Mobility Model, Trace, Generator**

I. Motivation

Wireless networks are in an emerging state since the last decade. Wireless mesh networks (WMN) and wireless sensor networks (WSN) are two of the most prominent examples. Although often only considered as static networks, mobile nodes can be part of these network architectures. In WMNs the clients are usually mobile and roam from one mesh router to another. Depending on the definition, the wireless mesh routers can also show at least some limited mobility. High mobility is introduced when clients extend the network by offering a routing service. WSNs used for animal tracking [1], [2], to monitor elderly [3], or for localization tasks [4] are only some of the many example applications that also possess this property. The disctinction into disjunct groups of WMNs, WSNs, and mobile ad-hoc networks (MANETs) is becoming blurred as novel applications are evolving traditional network architectures. The future Internet and the often named "Internet of Things" will likely possess mobile components. Thus the simulation of mobile networks is still an aspect of current and future scientific and industrial research. Many publications are based on simulations [5]. However, simulations make assumptions and have limitations that often result in conclusions that cannot be transfered to real networks as they abstract from the reality in many aspects [6]. Besides the used radio propagation model, mobility models play a significant role in experiments and their scientific soundness. To evaluate real world applications the simulated nodes have to move as close as possible like their real counterparts [7]. Yet many of the commonly used mobility models emulate human or vehicular behavior poorly. Additionally, often times an empty and rectangular movement area is considered or a simple grid-like graph. Although more sophisticated models for urban and suburban mobile networks exist, these approaches have limitations. In this paper we introduce the *Mobile Node Tracer* (MoNoTrac), a work-in-progress tool to generate mobility traces based on geographical data provided by the *OpenStreetMap* project. A plug-in interface allows the customization and adaptation to the requirements for specific experiments.

The remainder of the paper is organized as follows. In Section II the related work is discussed. Subsequently in Section III, MoNoTrac is introduced and its features elaborated. Section IV lists some features for future versions of MoNoTrac that are up for discussion. The paper closes with a conclusion in Section V.

II. Related Work

Several mobility models have been defined to generate traces as input to other applications or alternatively to directly control mobile nodes in a simulation environment. The most simple mobility model *random walk* resembles a Brownian motion but no human behaviour. This applies also to the *random waypoint* mobility [8] and *random direction* models that show sharp angles in the paths of the nodes. The *Gauss-Markov* model considers recent moves of a node for the next waypoint and generates a smoother human-like path. Animal movement can be modeled by the Levy Walk model although it also represents some human walk patterns [9].

None of these models often used in simulations restricts the area of movement to real world like structures. With the *Manhattan grid* model [10] the area of movement is limited to a grid-like graph resembling some central city areas. *Freeway* models try to simulate traffic flows on roads, whereas *city area* models combine densely populated city centers with suburban and rural areas. One tool making use of these models is the *Communication Scenario and Mobility Scenario Generator* (CosMos) [11] which allows to create zones where different mobility models are applied to mobile nodes. The nodes can move from one zone to another which is modeled as a Markov-Chain.

Besides the artificial generation of mobile node traces, real data is often preferred. Yet these data sets are hard to get and require large expenses to distribute, maintain, and operate tracking devices and the required infrastructure. Privacy concerns aside, GSM and UMTS data is glady used if available [12]. Only the date and time of a communication and cell location information are available but no detailed

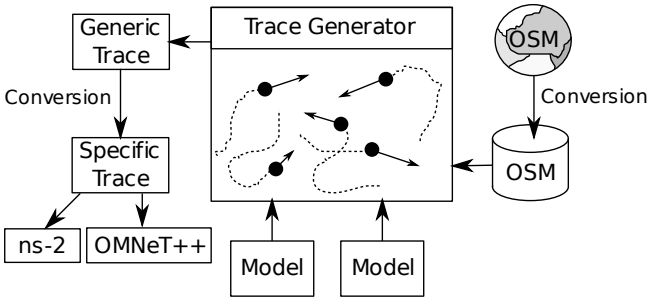


Figure 1: System model of MoNoTrac.

information about the traveled path. The mobility data is only coarse grained (macroscopic level of mobility). Simulations of mobile networks require more detailed position information. Further on, for the simulation of WSNs, WMNs, and MANETs a different user behaviour and mobility can be assumed than the one in cellular systems.

The CCC Sputnik project [13] published several traces gathered by RFIDs from conferences, e.g, the Chaos Communication Camp in 2007. The project wants to demonstrate the problems, threats, and benefits that have to be considered by tracking and data mining. Their movement data is limited to conference buildings.

One of the major problems with real traces is that they cover a limited time span that cannot be extended at will. For the sound simulation of mobile networks multiple traces of mobile nodes in the same environment are required.

Several applications try to provide the user with more realistic traces based on real maps. Street maps in the MapInfo MIF/MID-format are used by the *random waypoint city* model [14] to simulate vehicular movement. Saha and Johnson [15] used maps from the *Topologically Integrated GEographic Encoding and References* (TIGER) database provided by the *US Census Bureau* for their mobility model of vehicular traffic. The *Mobility model generator for VEhicular networks* (MOVE) [16] is a tool that uses user or randomly generated maps as well as maps from the TIGER database. It is built on top of SUMO a micro-traffic simulator [17]. Generated traces can be imported by ns-2 or Qualnet. The TIGER database seems to be a popular source of maps but it only comprises US street maps and provides limited information, e.g., no speed limits or no one-way roads.

III. MoNoTrac

MoNoTrac is a work-in-progress framework to generate mobility traces based on mobility models and real map data. Scenario descriptions, which consist of a movement area, mobile nodes, mobility models, and a simulation time are used as input for MoNoTrac to generate mobility traces. The application is based on Java 6 and published under the GPL.

Users create a scenario by selecting a region with streets, roads, and public transportation stations from a repository of geographic data that serves as movement area of mobile nodes. In a next step, the number and type of mobile nodes are added to the scenario description. A mobility model is applied to each participating node. The implemented mobility models are available as plug-ins. This software architecture allows the user to extend MoNoTrac with custom mobility models. Finally a runtime is specified for the scenario.

Based on the scenario description, MoNoTrac generates mobility traces in a meta format based on XML. This format is designed in a way, that a translation script can automatically convert the generated mobility traces to input formats of popular simulation environments such as ns-2 and OMNeT++. Thus, mobility traces based on real map data can be easily integrated into custom wireless network simulation scenarios. The system model and workflow of MoNoTrac is shown in Figure 1. With the focus on the usability and simplicity to define scenarios and generate mobility traces, a rapid trace generation is achieved, which we missed in other tools.

The maps of the OpenStreetMap project are used for the real world map data sets. The vector format based OpenStreetMap data are parsed and transformed to a graph representation and stored in a relational database[1]. The main window of MoNoTrac is shown in Figure 2. On top of the rendered OpenStreetMap map data the extracted graph is drawn. The user can draw any polygon to specify the movement area of the mobile nodes. The movement area can be bounded or boundless. Thus nodes can leave, enter, and reenter the simulation area if required.

The number, type, and distribution of mobile nodes can be specified. Currently, the types *pedestrian* and *car* are supported. These types of nodes are limited in their movement to the appropriate edges of the graph based on the values provided by the OpenStreetMap data. The nodes move according to the specified mobility model. At the time being, the *random walk*, *random waypoint*, and a variant of the *manhattan grid model* are implemented as plug-ins for MoNoTrac.

IV. Outlook and Topics of Discussion

A first release candidate of MoNoTrac will be soon available at http://des-testbed.net. The application is under heavy development. Currently the map of the Berlin area has been preprocessed and stored in a database.

In the next step an online processing is required to make all areas available for MoNoTrac. Also the possibility to visualize the generated mobility traces will be implemented. Several extensions are envisioned. First of all, different models shall be usable for particular zones with inter-zone mobility similar to CosMos. Temporal mobility characteristics, e.g., rush hours are considered. While they can be part of a loaded mobility model, the framework should provide support to dynamically modify the configuration. We also like to incorporate the public transportation system to provide further “roads” for pedestrians. While extracting the public transportation lines from the maps is feasible, there is no general and easy way to provide schedules for the public transportation vehicles. MOVE supports bus timetables but they have to be specified by the user which is a labour-intensive task. To increase the accuracy of simulations based on the generated traces additional information could be

[1]The graph presentation is kindly provided by the *Databases and Information Systems* research group at *Freie Universität Berlin*. Thanks to Joos-Hendrik Böse and Jürgen Bross.

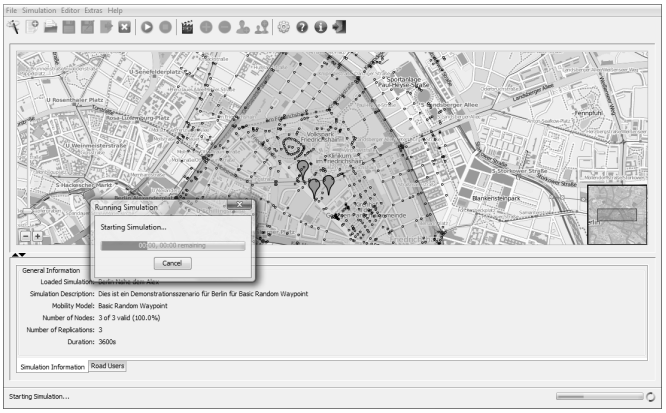


Figure 2: Mainwindow of MoNoTrac. Currently a trace is generated.

provided based on the OpenStreetMap data, such as the road type and according speed limits. To adapt the radio propagation to the environment several environmental parameters could be extracted, e.g., where living, industrial, and office areas are.

V. Conclusion

Simulation environments are one of the most important tools of scientific research of mobile networks. For life-like simulation of real world applications the most often used simple mobility models do not generate data resembling human movement patterns. As real traces of mobile nodes are of limited availability, their generation based on map data has been of interest in the last years. In this publication we introduced MoNoTrac a framework to create traces of mobile nodes. Data from the *OpenStreetMaps* project is used together with user supplied mobility models via a plugin infrastructure to create input for various simulation environments.

VI. Acknowledgments

This work presented has been partially sponsored and supported by the European Union OPNEX project.

References

[1] T. Liu, C. M. Sadler, P. Zhang, and M. Martonosi, "Implementing software on resource-constrained mobile sensors: experiences with impala and zebranet," in *MobiSys '04: Proceedings of the 2nd international conference on Mobile systems, applications, and services*. New York, NY, USA: ACM, 2004, pp. 256–269.

[2] O. Landsiedel, J. A. Bitsch Link, K. Wehrle, J. Thiele, and H. Mallot, "Rat watch: Using sensor networks for animal observation," in *ACM Workshop on Real-World Wireless Sensor Networks (RealWSN) in conjunction with ACM MobiSys*, 2006, (Poster and Abstract).

[3] C. Secombe, R. Steele, and W. Brookes, "Perceptions of the elderly on the use of wireless sensor networks for health monitoring," in *OZCHI '06: Proceedings of the 18th Australia conference on Computer-Human Interaction*. New York, NY, USA: ACM, 2006, pp. 55–62.

[4] L. Hu and D. Evans, "Localization for mobile sensor networks," in *MobiCom '04: Proceedings of the 10th annual international conference on Mobile computing and networking*. New York, NY, USA: ACM, 2004, pp. 45–57.

[5] S. Kurkowski, T. Camp, and M. Colagrosso, "MANET simulation studies: the incredibles," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 9, no. 4, pp. 50–61, 2005.

[6] D. Cavin, Y. Sasson, and A. Schiper, "On the accuracy of MANET simulators," in *POMC '02: Proceedings of the second ACM international workshop on Principles of mobile computing*. New York, NY, USA: ACM Press, 2002, pp. 38–43.

[7] M. Günes and M. Wenig, *Models for Realistic Mobility and RadioWave Propagation for Ad-hoc Network Simulations*. Springer, 2009, ch. 11, pp. 255–280. [Online]. Available: http://www.springer.com/computer/communications/book/978-1-84800-327-9

[8] J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, and J. Jetcheva, "A performance comparison of multihop wireless ad hoc network routing protocols," *Proceedings of the Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'98)*, pp. 85–97, 1998.

[9] I. Rhee, M. Shin, S. Hong, K. Lee, and S. Chong, "On the levy-walk nature of human mobility," in *Proc. INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, 2008, pp. 924–932.

[10] J.-K. Chen, C. Chen, R.-H. Jan, and H.-H. Li, "Expected link life time analysis in manet under manhattan grid mobility model," in *MSWiM '08: Proceedings of the 11th international symposium on Modeling, analysis and simulation of wireless and mobile systems*. New York, NY, USA: ACM, 2008, pp. 162–168.

[11] M. Günes and J. Siekermann, "Cosmos – communication scenario and mobility scenario generator for mobile ad-hoc networks," in *The Second International Workshop on Mobile Ad Hoc Networks and Interoperability Issues (MANETII'05), International Conference on Wireless Networks (ICWN'05)*, Las Vegas, Nevada, USA, 27-30 June 2005.

[12] ——, "Are ad-hoc networks able to substitute cellular networks? - a performance comparison of ad-hoc network routing protocols in realistic scenarios," in *International Workshop on Wireless Ad-hoc Networks (IWWAN'05)*, London, UK, May 2005.

[13] "Ccc sputnik at openbeacon.org," last visit: May 2009. [Online]. Available: http://www.openbeacon.org/ccc-sputnik.0.html

[14] J. Kraaier and U. Killat, "- the random waypoint city model -: user distribution in a street-based mobility model for wireless network simulations," in *WMASH '05: Proceedings of the 3rd ACM international workshop on Wireless mobile applications and services on WLAN hotspots*. New York, NY, USA: ACM, 2005, pp. 100–103.

[15] A. K. Saha and D. B. Johnson, "Modeling mobility for vehicular ad-hoc networks," in *VANET '04: Proceedings of the 1st ACM international workshop on Vehicular ad hoc networks*. New York, NY, USA: ACM, 2004, pp. 91–92.

[16] F. Karnadi, Z. H. Mo, and K.-c. Lan, "Rapid generation of realistic mobility models for vanet," in *Proc. IEEE Wireless Communications and Networking Conference WCNC 2007*, 2007, pp. 2506–2511.

[17] D. Krajzewicz and M. Behrisch, *SUMO - Simulation of Urban MObility - User Documentation*. [Online]. Available: http://sumo.sourceforge.net/docs/gen/sumo_user.pdf

A Prototype Implementation of the TinyOS Hardware Abstraction Architecture for Ferroelectric RAM

Sebastian A. Bachmaier
Material Testing Institute
Universität Stuttgart
Stuttgart, Germany
sebastian.bachmaier@mpa.uni-stuttgart.de

***Abstract*—This report presents a hardware driver for the Ramtron Ferroelectric RAM (FRAM, FeRAM) chips for use in TinyOS according to TinyOS' Hardware Abstraction Architecture. FRAM is a replacement for flash memory, suitable for usage in Wireless Sensor Networks (WSNs) for its properties. The properties of FRAM and flash are shortly depicted and compared. The design of the driver implementation is described, including a chip clustering method to circumvent the capacity limitation. The driver offers the DirectStorage interface and the BlockStorage interface for usage by applications. Comments on the suitability of the provided interfaces, intended for flash memory originally, for FRAM are given.**

***Index Terms*—TinyOS, FRAM, driver, hardware abstraction architecture**

I. INTRODUCTION

In this report, TinyOS, TinyOS drivers and FRAM are introduced shortly. Then, the architecture of the driver implementation is shown, the clustering of several chips is described and finally the suitability of existing TinyOS flash storage abstractions is assessed.

A. TinyOS and its Driver Model

TinyOS is an operating system and library of code components for sensor networks. The further development is done by working groups and by user contributions. Working groups can issue TinyOS Extension Proposals (TEPs), specifying best practices for new code contributions.

TEP2 [1] is one of the central TEPs for TinyOS 2.0. It describes a *hardware abstraction architecture* (HAA). The HAA specifies a three-layered architecture for driver implementations. The three layers comprise the *hardware presentation layer* (HPL), which exposes the hardware's capabilities directly, the *hardware adaptation layer* (HAL) which abstracts the hardware and allows to maintain states in software, and the *hardware interface layer* (HIL) which offers a standardized platform-independent interface for applications, irrespective of the underlying hardware components. Drivers reside in a *chip* directory by convention, with some additional code in a *platform* directory where code is placed which states the platform specifics, like specific hardware pins.

B. Ferroelectric RAM (FRAM)

FRAM is a relatively new memory technology which combines the best from static RAM memory (fast, energy efficient) and flash memory (non-volatile). It is based on a ferroelectric material which retains its state even when currentless. FRAM is a suitable replacement for flash.

TABLE I. shows a simplified comparison between flash and FRAM memory properties. Please note that the values for a specific application have to be taken from the actual datasheet of the actually used chip. Values may vary greatly, especially for the energy consumption per stored bit as this depends not only on the used chip, but also on the calculation model, e.g. the assumptions made with respect to read-write cycle times and the assumed bus speed. Hence, the data is only given to stress some main differences. These are: (1) the durability in terms of write cycles. This is irrelevant for many applications however. (2) The capacity which is in favor of flash memory since being in a later stadium of the development cycle and the smaller manufacturing processes. (3) The energy effort to store a bit. This is an important quantity in WSNs for the power limitations imposed to the system owing to the desired autonomous operation over long time periods.

TABLE I. SIMPLIFIED FLASH–FRAM COMPARISON

Comparison with respect to …	**Type of Non-Volatile Memory**	
	flash	***FRAM***
Available Interfaces	SPI/I2C/Parallel	SPI/I2C/Parallel
Sleep Mode Current	1 µA	1 µA
Data Retention	> 10 a	> 10 a
Write Cycles	~ 10^5	~ 10^{10}
Capacity[a]	≤ 32 Gibit (parallel) ≤ 128 Mibit (SPI)	≤ 4 Mibit (parallel) ≤ 2 Mibit (SPI) ≤ 1 Mibit (I2C)
Energy Consumption[b]	90 nJ/bit	1.1 nJ/bit
Write Speed/byte[c]	~ 10 µs	~ 400 ns

a. Development is making rapid progress. This is a snapshot view only. Capacity varies with physical chip/die size.

b. These values differ greatly with the usage model used for calculation and the actual chip.

c. Depending on bus speed, data unit size and others.

Ramtron, Colorado Springs, CO offers FRAM chips with SPI bus which are meant to replace serial flash memory. The SPI protocol used is similar to the one of flash chips. Pin compatibility is also given. It is therefore easy to replace flash by FRAM. The realization here is for the FM25H20 type.

This work was funded by the 7th Framework Programme of the European Commission.

II. IMPLEMENTATION

The implementation follows that of the STM25P flash chip by Hui [2]. The components should reside in tinyos-2.x/tos/chips/fm25h and platform/<platform-name>/chips/fm25h. However, the contribution resides at tinyos-2.x-contrib/ustutt where it can be retrieved from.

A. Hardware Presentation Layer (HPL)

The HPL offers no *erase* and *pageProgram* commands, but offers a *write* command instead. In FRAM writes are possible without prior erase. The write command operates on data units of down to single bytes. Flush is not implemented since data is always written through. Sleep mode support is available.

B. Hardware Adaptation Layer (HAL)

Two HAL implementations are offered: one simpler HAL for single chip mode and a ClusterHAL component for clustered operation of several chips under a flat, continuous address space. Standard wiring uses the single chip HAL. The single chip HAL is similar to the STM25P implementation.

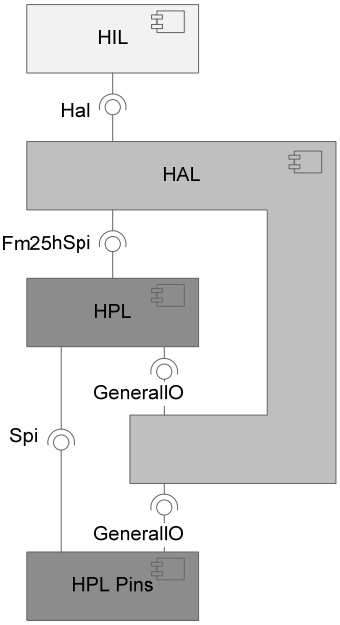


Figure 1. HAA of the clustered operation: the HPL is ignorant of the actually used chip as this is dispatched by the HAL (GeneralIO interface)

C. ClusterHAL

While the FRAM offers some advantages over flash memory it still offers less capacity. This is due to the smaller packing density, which is caused by the larger manufacturing process sizes and the ferroelectric material properties.

We therefore had to bundle several chips to get a memory size comparable to the 1 MiB of the TelosB which were used as reference. The resulting cluster was desired to act like one big memory under a unified address space. This means a dispatcher has to handle accesses to the unified address space and direct them to the corresponding chip. The dispatcher is provided on the HAL layer. This has the advantage of having the HPL unchanged for cluster or single chip operation. Furthermore, HPL can be stateless and "present" just the operations the FRAM offers. However, the layering in this approach is not strictly adhered to as for clustered operation the IO pins are handled by the HAL, transparently to the HPL (see Figure 1). The HPL just accesses the chip select (CS) to activate the large virtual chip (of which the HPL is ignorant of) and the HAL activates the appropriate physical chip determined by the memory address that is accessed. Memory accesses across chips are split into several separate operations. Other approaches are conceivable and can be implemented later. E.g., a strictly layered architecture would access several individual HPLs, but code is replicated then.

D. Hardware Interface Layer (HIL)

For a description of two prototype implementations of HIL refer to chapter III. The implementation here follows the STM25P implementation.

III. SUITABILTY ASSESMENT OF FLASH ABSTRACTIONS

A. BlockStorage Interface (TEP103)

TEP103 [3] standardizes three fundamental storage abstractions found in typical sensor network applications: BlockStorage for program memory, ConfigStorage for little chunks of configuration data and LogStorage for data logging application. TEP103 aims solely at flash memory and incorporates specialties of flash memory. However, the flash functionality is a subset of FRAM functionality, i.e. FRAM has fewer restrictions to consider. It should therefore be possible to realize these storage abstractions for FRAM. For workload restriction, of the three storage abstraction of TEP103, only the BlockStorage was implemented exemplarily.

B. DirectStorage Interface (TEP128)

TEP 128 ([4], draft version) describes an interface for direct access to non-volatile storage. It offers read, write, erase, flush and crc commands. It differs primarily in two points from the abstractions of TEP103: (1) the interface is an application indifferent general purpose interface, and (2) the implementation (in conjunction with the VolumeSettings interface) is platform independent. TEP 129 describes a new set of BlockStorage, ConfigStorage and LogStorage which resides above the platform-independent intermediate DirectStorage interface.

There are some comments to the DirectStorage interface which occurred during implementation of the interface. Due to the missing sector size of FRAM, the sector size can artificially be set to either an arbitrary size (for easier programming a fraction of a power of two) or it can be set to 1 which is the natural sector size of FRAM. This leads to two consequences: (1) the volume information structure *fm25h_volume_info_t* which is set in the tool *tos-storage-fm25h* should define both *base* and *size* as *uint32_t* instead of *unit8_t* to accommodate the larger size numbers (this is internal to the chip specific tool chain and has no consequences to the interfaces), and (2) the *erase* command's parameter *eraseUnitIndex* (and the corresponding *eraseDone* event's) should likewise be *uint32_t*, instead of *uint16_t*. This results from the erase unit size which is of size 1 as well. However, here the advantage of an artificially introduced larger erase unit size becomes obvious. For erasing larger memory regions less function calls were necessary then.

A last comment is given on the DirectModify interface signature. It is perhaps preferable to name all completion events in the same manner and so rename the completion event of *modify(...)* to *modifyDone(...)*.

IV. CONCLUSION

FRAM can replace flash memory when low-power operation in combination with short write access times is of importance. Drawback is the smaller maximum capacity per chip. If the capacity is a limiting factor, several chips can be clustered as proposed. Several HIL interfaces originally intended for flash memory were successfully implemented. Recommenda-

tions were made to the DirectStorage and DirectModify interfaces, both being still in draft status and open for changes.

ACKNOWLEDGMENT

S. B. likes to thank Helmut Ernst for his support in the hardware development of the sensor nodes used for this work.

REFERENCES

[1] V. Handziski, J. Polastre, J.-H. Hauer, C. Sharp, A. Wolisz, D. Culler, D. Gay, "TEP2 – Hardware Abstraction Architecture". http://www.tinyos.net/tinyos-2.x/doc/html/tep2.html. Date of access: 2009-05-25

[2] J. Hui, "Implementation of the TEP103 for the ST M25P serial code flash". http://tinyos.cvs.sourceforge.net/viewvc/tinyos/tinyos-2.x/tos/chips/ stm25p. Date of access: 2009-05-28

[3] D. Gay, J. Hui, "TEP103 – Permanent Data Storage (flash)". http://www.tinyos.net/tinyos-2.x/doc/html/tep103.html. Date of access: 2009-05-25

[4] D. Moss, J. Du, P. Dutta, D. Ganesan, K. Klues, A. Martin, G. Mathur, "TEP128 - Platform Independent Non-Volatile Storage Abstractions". http://www.tinyos.net/tinyos-2.1.0/doc/html/tep128.html. Date of access: 2009-05-25

tinyMoBot: A Platform for Mobile Sensor Networks

Torsten Stremlau, Christoph Weyer, and Volker Turau
Institute of Telematics
Hamburg University of Technology
Hamburg, Germany
{torsten.stremlau, c.weyer, turau}@tu-harburg.de

***Abstract*—In many application scenarios of wireless sensor networks parts or the whole network consist of mobile sensor nodes. Currently, no common platform is available. This paper describes a project that has developed a mobile sensor node, based on standard components: a Crossbow IRIS mote and LEGO MINDSTORMS NXT components, i.e., motors and sensors.**

I. Introduction

Wireless sensor networks consisting of mobile nodes are currently a growing research area [1], [2], [3]. In these scenarios often only few nodes of the network are mobile. Therefore, it is important that the mobile nodes are able to communicate with static nodes. Since a general mobile sensor network platform is not available, often self-developed mobile sensor nodes are used [4]. A standard sensor node is attached via an adapter hardware to an existing robot platform. In most cases the robot platform or the adapter hardware has an additional micro processor, which increases the complexity of software development. In this paper the mobile sensor network platform tinyMoBot is presented. A circuit board without a micro controller is used as an adapter. Enabling a Crossbow IRIS mote to directly operate with LEGO MINDSTORMS NXT motors and sensors. This decreases the complexity of the resulting mobile platform since software needs to be developed for one target platform only. Other benefits of this solution are the low hardware complexity and the relatively low cost of the adapter. The rich arsenal of available LEGO components make this an attractive platform.

II. The tinyMoBot Adapter

Figure 1 provides an overview of the main components of the tinyMoBot adapter and how they are connected. The ATMega1281 on the IRIS mote is connected via the 51-pin socket with the adapter. Since the adapter is mounted on a robot, removing the IRIS mote for reprogramming or debugging is not feasible. Therefore, the adapter provides appropriate interfaces so that there is no need to disconnect the IRIS mote. An ISP header is available for programming the IRIS mote while it is connected. The adapter also provides an USB connector to enable serial communication with the IRIS mote, which is convenient for debugging purposes.

To be flexible in its use, the adapter provides almost the same connectivity capabilities as the LEGO MINDSTORMS NXT brick. The adapter consists of six sockets for connecting up to three motors and three sensors. It also supports every

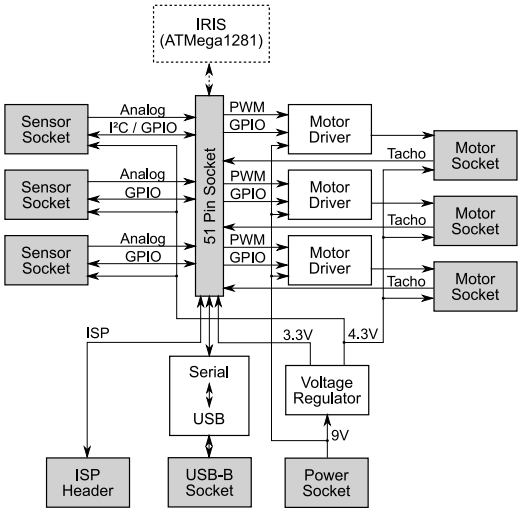


Fig. 1. Components of the tinyMoBot Adapter

type of LEGO sensors available. Currently, there are three types of sensors: active sensors, passive sensors, and digital sensors. Active and passive sensors are sensors that provide an analog voltage, representing their measurements. Digital sensors a controlled via the I^2C protocol. Additional circuit is needed in order to support the different sensor types. As seen in Fig. 1 the different components of the robot and the IRIS mote require different supply voltages. Since it is not feasible to carry multiple sets of batteries, voltage regulators are used to create the different voltages from the standard LEGO voltage of 9 V.

III. Realization of the Adapter

In this section the realization of the tinyMoBot adapter is briefly described. The board was developed during a project work at Hamburg University of Technology [4]. EAGLE from Cadsoft was used for developing the circuit diagram. TinyOS components are available to handle the different hardware components. Figure 2 shows the layout of the finished adapter.

Because of limitations of the ATMega1281 and to simplify the design of the adapter its supports only digital sensors on the first sensor socket. Active and passive sensors are supported on all three ports. Analog sensors provide an analog output voltage between 0 V and 5 V. By using a voltage divider this

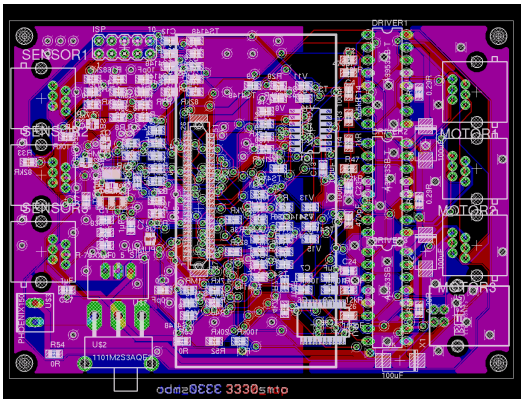


Fig. 2. Final Layout of the Adapter

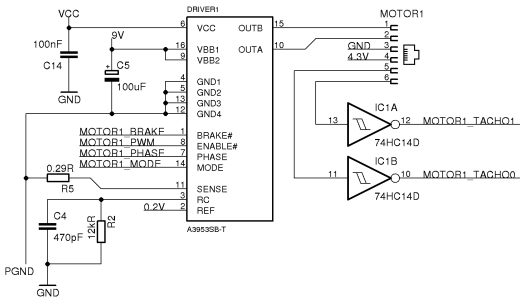


Fig. 3. Schematic of a Motor Port

output voltage is limited to a maximum of 3.3 V, in order to use the sensors directly with the ADC of the IRIS mote.

LEGO MINDSTORMS NXT motors are DC motors and the speed is controlled utilizing PWM. The adapter consists of motor driver ICs to enable the ATMega1281 to control the motors directly. The LEGO MINDSTORMS NXT motors also provide a tacho signal that delivers an accurate measurement of the current speed. These signals are connected to interrupt lines of the ATMega1281.

Figure 3 shows the schematic of the first motor port, which is same for all three ports. DRIVER1 is the motor driver IC used to drive the motors. The driver IC also limits the maximum current of the motors for preventing damages. The IC does this by temporarily switching off the current. The length of this period is determined by the time constant of the RC circuit consisting of C4 and R2. The IC measures the actual current, by using the voltage drop over R5. The lines MOTOR1_BRAKE, MOTOR1_PWM, etc. are directly connected to the ATMega1281 and are used to control the motor. As the tacho signals of the motor use 5 V, they cannot directly be connected to the ATMega1281, which uses 3.3 V. Inverting Schmitt triggers IC1A and IC1B are utilized therefore.

A switching voltage regulator is used to provide 5 V from the supply voltage of 9 V, then a linear LDO voltage regulator is used to provide 3.3 V. The 4.3 V are created by using the 5 V and the voltage drop of a diode. The adapter works in a voltage range between 6 V and 9 V. As the motors are driven directly by the supply voltage, the speed of the motors will be slowed down when the voltage drops.

TABLE I
ENERGY CONSUMPTION WITH A SUPPLY VOLTAGE OF 7.5 V

Action	Measurement
no movement, sleep mode (calculated)	25 mA
no movement, radio enabled	30 mA
straight driving	270 mA
turn driving	300 mA
maximum speed	0.28 $\frac{m}{s}$

IV. EVALUATION

Table I shows the energy consumption of the completely assembled robot. Under the assumption of batteries with 2000 mAh the robot should be able to drive for about 6.6 h. During that time the robot would travel a distance of 6.72 km.

The adapter draws a relatively high current of 25 mA even in sleep mode. This relative high consumption is due to the following effects. A fixed consumption of 7 mA for each motor (in this case 14 mA) that is plugged must be assessed. The reason for this may be some internal details of the LEGO motors. The motor driver ICs are another source for this high consumption. Each of the ICs have a leakage current of 1 mA. Unfortunately, the voltage regulator that provides the 5 V has a very low efficiency at low current output, so that is takes a few mA even if no current is needed at the output. Additionally, the voltage dividers used on the board have a current consumption of around 1 mA. In a next version of the board these leakages must be considered in more detail.

The adapter board was successfully used in a student project to develop a remote control for the tinyMoBot. The software of the remote control is executed on a separate IRIS mote equipped with an acceleration sensor. Based on gesture recognition the remote control creates movement commands. These are transmitted wirelessly to the tinyMoBot.

V. CONCLUSION

The tinyMoBot is currently used in programming courses at Hamburg University of Technology. In the near future a new hardware revision should resolve the existing problems, especially the high current consumption in sleep mode.

REFERENCES

[1] A. Kansal, A. Somasundara, D. Jea, M. Srivastava, and D. Estrin, "Intelligent Fluid Infrastructure for Embedded Networks," in *Proceedings of the Second International Conference on Mobile Systems, Applications, and Services (MobiSys'04)*, Boston, MA, USA, 2004.

[2] K. Dantu, M. Rahimi, H. Shah, S. Babel, A. Dhariwal, and G. S. Sukhatme, "Robomote: Enabling Mobility in Sensor Networks," in *Proceedings of the Fourth International Symposium on Information Processing in Sensor Networks (IPSN'05)*, Los Angeles, CA, USA, 2005.

[3] M. Rahimi, H. Shah, G. S. Sukhatme, J. Heideman, and D. Estrin, "Studying the Feasibility of Energy Harvesting in a Mobile Sensor Network," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'03)*, Taipei, Taiwan, 2003.

[4] T. Stremlau, "Entwicklung einer mobilen Plattform für drahtlose Sensornetze," Project Work, Hamburg University of Technology, Institute of Telematics, 2008.

Clock Synchronization of TinyOS-based Sensor Networks with DCF77

Lars Niemann, Marcus Venzke, Christian Renner, and Volker Turau
Institute of Telematics
Hamburg University of Technology
Hamburg, Germany
venzke@tu-harburg.de

***Abstract*—The paper presents an approach of applying DCF77 time radio signals to provide a clock with global time in sensor networks based on TinyOS. Some nodes containing DCF77 receiver hardware reliably decode time signals even if these are distorted to some extent. The clock with global time is provided by compensating clock drift. Time is distributed in the network with a protocol generating timestamps on the MAC layer.**

I. INTRODUCTION

Sensor network nodes contain their own local clocks. Frequently these are synchronized in the network or between neighboring nodes as required for several algorithms (internal synchronization). In most cases synchronization is not performed to global time because this would require a suitable time source. On the other hand, global time is required if data is measured on several nodes, aggregated, and should be interpreted outside the network.

This paper thus analyses how clocks in sensor networks can be synchronized to global time (external synchronization). It assumes the use of radio time signals as time source, which are transmitted in several countries. It discusses how these signals can be evaluated reliably in hardware and software, how local clocks can be synchronized to it, and how time can be distributed in the network. TinyOS and IRIS nodes are assumed as platform for a demonstrator.

II. CLOCK SYNCHRONIZATION IN SENSOR NETWORKS

Two important protocols for time synchronization in sensor networks are the Flooding Time-Synchronization Protocol (FTSP) [1] and the TimeSync Protocol for Sensor Networks (TPSN) [2]. Both select a leader node whose local clock acts as time source. Its time is periodically broadcasted in packets also containing identification information. Other nodes receive and store it in conjunction with their local reception time and estimate the global time. With FTPS each node immediately retransmits the packets. In contrast, with TPSN the local estimation of global time is sent to other nodes in a tree structure.

TinyOS implements its local clock in software incrementing an integer from system start with 1kHz, 32kHz, or 1MHz. It implements clock synchronization in module TimeSyncC [3] as a combination of TPSN and FTSP. A tree of nodes is established using the node with lowest ID as root. Timestamps are generated at MAC layer when packets are sent or received [4]. This makes transmission time for timestamps small and deterministic, resulting in a higher accuracy clock synchronization, since the delay for waiting for a free channel is not included.

III. DCF77 RECEIVER HARDWARE

To implement a clock with global time, a sensor network needs a source of the global time to be distributed in the network. Candidates are satellite systems (e.g. GPS) as well as terrestrial radio transmitters available in several countries. The latter requires receiver hardware with less complexity and lower energy consumption and is more suitable for sensor networks. The paper thus assumes the use of Germany's terrestrial long wave time transmitter DCF77. Its utilization requires receiver hardware contained in one, some, or all sensor network nodes.

The DCF77 time signals are transmitted on 77.5 kHz by the Physikalisch Technische Bundesanstalt close to Frankfurt. Its transmission range of 2000 km spans most of central Europe. Date and time are transmitted in packets with 1 bit/s. Bits and packets are exactly aligned to seconds and minutes of global time respectively. It is modulated with amplitude (AM) and phase modulation (PM). The latter permits decoding with higher accuracy and reliability. On the other hand, decoding AM allows much simpler hardware and three orders of magnitude less energy. It is thus more suitable for sensor networks [5].

The DCF77 receiver hardware module for IRIS nodes shown in Fig. 1 has been developed using AM demodulation [5]. It is based on the receiver module EM6 DCF with a ferrite

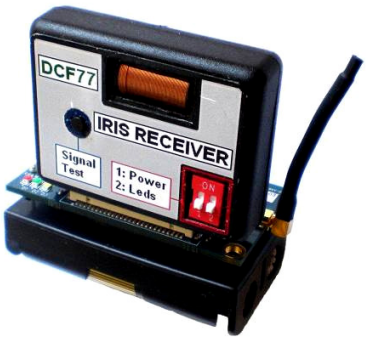


Figure 1. DCF77 receiver hardware developed for IRIS nodes

antenna of HKW-Elektronik requiring 250 µA in a voltage range between 1.1 V and 3.6 V. The module is directly connected to the IRIS node's 51-pin expansion connector without voltage regulator. An IO pin is used as power supply allowing power-down to save energy. An interrupt line reads the demodulated, digital DCF77 signal. Two more switches connected to IO pins are used to enable test software for debugging and adjusting antenna. The antenna has to be mounted at least 1cm from the IRIS node to not obstruct reception.

IV. Decoding Distorted Signals with High Accuracy

The receiver's digital signals have to be decoded by the sensor node's controller in software. Signals can be distorted, if radio reception is poor, e.g. due to unfavorable orientation of the antenna. The decoder needs to gracefully handle distortion and still regenerate time with high accuracy.

A combination of interrupt driven edge detection and periodic sampling should be applied to detect the edges of the receiver's distorted, digital signals with high accuracy and low processing time. Distortion results in quick erroneous changes between high and low as shown in Fig. 2 depicting two bits. Detecting all edges with interrupts would lead to a high interrupt rate requiring a lot of processing time. On the other hand, sampling with a reasonable rate reduces accuracy of edge detection and thus clock accuracy. The combination applies interrupts to detect falling edges marking the exact start of a second of global time. Afterwards a low sampling rate is used to determine the intended rising edge distinguishing 1 and 0 bits.

Distortion and processing in the receiver can shift the falling edge's position reducing clock accuracy. The latter is a property of the receiver, leading to a delay of up to 3 ms for module EM6 DCF in case of good signal quality. Distortion leads to varying delays. Measurements have shown that it can be reduced to 1 ms using Exponentially Weighted Moving Average (EWMA).

V. A Global Time Clock for TinyOS

Node time synchronization is performed using timestamps received every minute and the falling edges of the DCF77 signal marking every second. Local node time depends on the internal or external oscillator of the microprocessor, so that clock drift is inferred. The latter has been observed to be as large as multiple seconds per day on the IRIS platform. Furthermore, clock drift is not a constant scaling factor, but varies among nodes and depends on the temperature.

As a result, we estimate the clock drift by comparing the local and global times elapsed since the last synchronization, i.e., timestamp generation via the DCF77 module. Smoothing of the estimation is performed using EWMA. An update-parameter $\alpha=0.8$ has been empirically determined and used in

$$c = \alpha \cdot c_{old} + (1-\alpha) \cdot c_{measure} \quad (1)$$

Besides updating the clock drift compensation factor c, the synchronization module stores the local time t_{sync} and the global time T_{sync} of the current synchronization process and discards

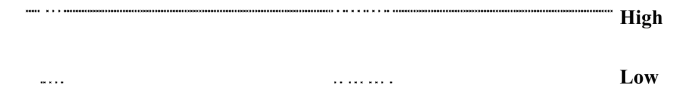


Figure 2. Digital signal with two distorted bits: 0 (left) and 1 (right)

earlier values. Global time T_t can thus be estimated at any (local) time $t \geq t_{sync}$ via

$$T_t = T_{sync} + c \cdot (t - t_{sync}) \quad (2)$$

VI. Distributing Time in the Sensor Network

The global time must be distributed in the sensor network by radio if some nodes do not have a DCF77 receiver or cannot receive the time signal, e.g. due to bad reception conditions. In TinyOS this can be implemented based on the available module TimeSyncC (see section II). The module must be adapted to forward received packets as described for FTSP. In case of several nodes having access to external global time, the node with the best signal quality should be selected as leader.

Energy restrictions make leader selection an awkward issue. Nodes should remain in sleep mode and with radio receiver switched off as much as possible to save energy. Thus the reception of packets cannot be guaranteed. Dynamic changes in the network topology, such as the announcement of a new leader, are therefore hard to treat. An algorithm can deal with the issue in two ways. It can ignore energy issues leaving the radio transceiver in operation all times. Alternatively it can allow power save operation assuming a fixed leader sending timesync packets in fixed time intervals among which receiving nodes can sleep.

VII. Conclusion

DCF77 and similar radio time signals are an advisable global time source for sensor networks. Some nodes containing low power receivers listen to the omnipresent signals free of charge and distribute time to other nodes by radio. Good accuracy and low processing time are feasible even if signals are distorted to some extent. Simple arithmetic models allow synchronizing local clocks by adjusting clock drift.

References

[1] M. Maróti, B. Kusy, G. Simon, and Á. Lédeczi, "The Flooding Time Synchronization Protocol," In Proc. of the 2nd international Conference on Embedded Networked Sensor Systems, pp. 39-49, ACM, Nov. 2004.

[2] K. Römer, P. Blum, and L. Meier, "Time Synchronization and Calibration in Wireless Sensor Networks," In: Ivan Stojmenovic (ed.): Handbook of Sensor Networks: Algorithms and Architectures, John Wiley & Sons, pp. 199-237, 2005.

[3] "Component: tos.lib.ftsp.TimeSyncC," In TinyOS 2.0.2 Documentation of Interfaces and Components, University of California, Berkeley, July 2007. http://www.tinyos.net/tinyos-2.x/doc/nesdoc/telosa/

[4] M. Maroti, and J. Sallai, "Packet-level time synchronization," Version 1.1, Draft Documentary, TinyOS Core Working Group, May 2008. http://www.tinyos.net/tinyos-2.x/doc/pdf/tep133.pdf

[5] L. Niemann: Zeitsynchronisation mit DCF77 in TinyOS basierten Sensornetzen. Projekt work. Hamburg University of Technology, 2009.

Mission Statement: Applying Self-Stabilization to Wireless Sensor Networks

Andreas Lagemann, Jörg Nolte
Distributed Systems/Operating Systems
Brandenburg University of Technology Cottbus
Cottbus, Germany
Email: {ae,jon}@informatik.tu-cottbus.de

Christoph Weyer, Volker Turau
Institute of Telematics
Hamburg University of Technology
Hamburg, Germany
Email: {c.weyer, turau}@tu-harburg.de

***Abstract*—Long living and unattended deployments of wireless sensor networks requires fault-tolerant solutions. Self-stabilizing algorithms are providing these properties in an elegant and verifiable way. Recently, a lot of research has been performed to determine appropriate means to apply these promising technique to wireless sensor networks. In this paper the current state of the art in this field is given. Additionally, three major challenges are presented for achieving self-stabilizing sensor networks.**

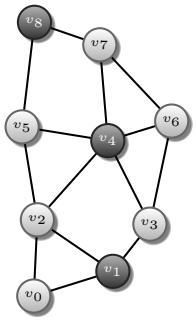
I. Introduction

Wireless sensor networks (WSNs) are operating under inherently instable conditions. The notoriously unreliable wireless communication facilities and environmental influences lead to highly dynamic conditions that in turn lead to frequent communication topology changes and other disturbances. The fact that WSNs usually are intended to run unattended for several months or years necessitates facilities that handle these dynamics in a self-acting manner. The concept of self-stabilization, introduced by Dijkstra in 1974 [1], comprises many properties that are helpful in this context. In fact a self-stabilizing system has embedded mechanisms to react to disturbances and faults in a self-controlled way. These mechanisms provide for the ability of the system to move from any faulty state to a safe one in bounded time. This property is called *convergence* and is augmented by *closure*, which additionally guarantees, that no move made by a self-stabilizing algorithm may lead to a faulty state. For an existing algorithm it can be shown formally, that it possesses both properties and thus is self-stabilizing. Self-stabilization can be regarded as a completely novel approach to fault-tolerance. Instead of specifying faults that may occur and creating algorithms that are robust against these particular faults, simply the set of desired system states is specified and algorithms are formulated such that these states are eventually reached after a fault occurred. This kind of fault-tolerance is also called non-masking fault tolerance, because faults are not (and need not be) detected as such and thus no measures to mask their effects can be taken.

In the following section we will introduce the concept of self-stabilizing algorithms and their application in WSNs. After that we will present three major challenges still to be met for applying self-stabilizing algorithms to WSNs. Finally, we will conclude by giving an outlook of work to be done.

II. Self-Stabilization in WSNs

Figure 1 shows a simple example for a self-stabilizing algorithm: it constructs a maximal independent set (MIS) as described in [2], which can be used as a basis for clustering. The membership of a node in the independent set is indicated by setting the variable `in` to true. Such an algorithm typically consists of a set of rules, which in turn consist of a *guard* (left of $\rightarrow$) and a *statement* (right of $\rightarrow$) that shall be executed when the guard evaluates to true.



$$v.in = false \wedge \forall\ w \in N(v):\ (w.in = false) \rightarrow v.in := true$$
$$v.in = true \wedge \exists\ w \in N(v):\ (w.in = true) \rightarrow v.in := false$$

Fig. 1. Maximal Independent Set [2]

Obviously it is necessary to define the exact semantics of these rules. There are two main questions that come to mind when looking at the algorithm in Fig. 1. The first thing to notice is, that the rules access variables of neighboring nodes. For the formal model often a special shared memory model is used. First of all the variables can be divided in public and private ones. Private variables can solely be accessed by the corresponding node. It is assumed, that each node can read the public variables of its direct neighbors but only the owner of a public variable can write it. To achieve this in WSNs, it is mandatory to provide the nodes with a reasonably stable view of their neighborhood. Therefore, this view has to hide the continuous fluctuations of the wireless link's quality from the algorithm. Mahalle [3] is a neighborhood protocol that achieves this and was developed especially for self-stabilizing algorithms. When this view is established, nodes can exchange the contents of their public variables on a regular basis with their neighbors and maintain a cache for each neighbor. The guards are then evaluated based on the cached values. This

communication model was introduced by Ted Herman [4] who called it *cached sensornet transformation*.

The second question is: when are the guards evaluated and the statements executed? First of all execution is assumed to take place in rounds, which seems to imply a synchronous system (which was indeed assumed at first) but the concept of rounds can also be extended to asynchronous systems, when the algorithm does not depend on knowledge about these rounds. Many approaches of self-stabilizing algorithms require, that no two neighboring nodes can execute a statement concurrently. That greatly simplifies the algorithm analysis and the proofs of the self-stabilizing property. The requirements are embodied by an abstract entity called *daemon*. The so called *central daemon* selects exactly one node per round and thus trivially achieves mutual exclusive execution of neighbors. This execution model is of course too restrictive to be applied in a realistic environment. Therefore, a more flexible model, the *distributed daemon* is introduced. Here in each round a subset of size N is selected to make a step concurrently. This model covers the *central daemon* (for $N = 1$) as well as the *synchronous* model where all nodes execute in each round (when N is set to the total number of nodes in the network). A node is called *enabled*, if one of its guards resolves to *true*. When an enabled node is selected, it executes the corresponding statement.

Algorithms developed for a central daemon often do not stabilize under a distributed or synchronous daemon due the concurrent execution within the neighborhood. The MIS algorithm depicted in Fig. 1 is an example of such an algorithm. To solve this so called *transformations* have been proposed [4]–[7]. They convert algorithms designed for such abstract models into semantically equivalent algorithms that stabilize under weaker assumptions.

III. Major Challenges

Applying self-stabilizing algorithms in the field of WSNs to increase the fault-tolerance is currently an active research area [4], [7]–[11]. In this section we will present the most essential challenges for utilizing the benefits of self-stabilization for WSNs.

A. Appropriate Programming Abstractions

One major concern is an appropriate programming abstraction that preserves the simplicity of the algorithms as well as the self-stabilizing properties. It is thereby very desirable to keep the algorithm *description language* independent of WSN specific details like the model transformation applied or the neighborhood protocol used. One major step in this direction is SelfWISE a programming abstraction designed for applying self-stabilizing algorithm in WSNs. It consists of the SelfWISE framework that is the runtime environment for executing self-stabilizing algorithm and a language to express those algorithms. Figure 2 depicts the MIS algorithm presented above in the SelfWISE language. For a more complete description of the SelfWISE framework and language see [10].

SelfWISE allows the application of self-stabilizing algorithms to WSNs by generating appropriate C/C++ code that can be run in a framework. Additionally, transformations that allow to conserve the self-stabilizing properties when communication takes place in a wireless ad-hoc network can be integrated. To fully guarantee preservation of the self-stabilizing properties it will be also necessary to investigate the influence of the compiler and the operating system. Here the development of techniques that preserve self-stabilization is mandatory for accomplishing the goal of completely fault-tolerant WSNs.

Self-stabilizing algorithms use some notion of neighborhood, which is not always merely the 1-hop communication neighborhood but may also span 2-hop communication distance or be defined by other means than communication neighborhood. A programming language for self-stabilizing algorithms must provide appropriate abstractions for such neighborhood notions. These abstraction must be designed such that they allow for efficient implementations with a low memory footprint. One could imagine communication models that provide access to the state of nodes in such alternative neighborhoods. The transformation that implements such an abstraction has to make a trade off between flexibility and energy consumption. To give developers control over this crucial performance aspects the programming abstraction must provide means to specify the desired trade off.

B. Efficient and Scalable Model Transformations

When concerning the communication model, the aforementioned cached sensornet transformation from Herman is widely regarded as appropriate for WSNs. For the execution model several proposals exist for transforming algorithms written for the central daemon such that they can be run in WSNs while self-stabilization is preserved. Transformations of the execution model ensure the exclusive execution within each neighborhood under the distributed or synchronous daemon. The idea behind these transformations is to break the symmetry by using unique identifiers or randomization. A *strict transformation* converts the algorithms in such a way that the execution of the resulting algorithms is equivalent to an execution under the central daemon. An algorithm $\mathcal{A}$ is transformed into $\mathcal{A}'$ such that only one node in each neighborhood performs a move of $\mathcal{A}$ concurrently. Examples for strict transformations are the deterministic conflict manager (CMD) [5] that uses unique node identifiers and BitToss [6]. The latter elects a neighbor by a Bernoulli trial until solely a single node is enabled. The main drawback of these strict transformations is the limited concurrent activity, exactly one node within each

```
algorithm MaximalIndependentSet;
    public bool in;
rule R1:
     in = false and forall(Neighbors v : v.in = false) -> in := true;
rule R2:
     in = true and exists(Neighbors v : v.in = true) -> in := false;
```

Fig. 2. Maximal Independent Set [2]

neighborhood executes its statement. Often this limitation is too restrictive and a higher degree of concurrency is needed. Algorithms converted by a *weak transformation* produce an execution that may not be possible under a central daemon. The reason for this is the fact that nodes may perform a move within a neighborhood concurrently. The idea is that potential deceptive statement executions are resolved after some time, but with the advantage of a faster convergence. Examples for weak transformations are the randomized conflict manager (CMR) [5] and the randomized transformation introduced by Turau and Weyer [7], which both lead to a probabilistic convergence. The latter reference also proposed a transformation that is even self-stabilizing in the case of occasional message losses.

A good metric for the quality of transformations is the average convergence time they yield for different algorithms. It could be shown that the transformation from [7] performs best with respect to average convergence time [12]. Nevertheless, all transformations regarded in [12] rather impose an abstract model on top of a WSN instead of integrating the algorithm more tightly. A rewarding goal would be a lightweight transformation which utilizes the characteristics of the wireless channel to increase efficiency.

C. Application Field of Self-Stabilization in WSNs

Another major concern is to find fields of application in WSNs where the benefits of self-stabilization show to advantage best. The fault tolerance added by self-stabilizing algorithms is the very first property that comes to mind. This alone is not necessarily a sufficient argument for their use, because the gain in fault tolerance must be carefully compared with other approaches to decide if self-stabilization is the method of choice. Methods for assessing the fault tolerance measure of self-stabilization are currently studied and developed (e. g., see [13]).

But there is more to self-stabilization than fault tolerance. The inherent flexibility of self-stabilizing algorithms seems to be especially well suited to deal with the dynamics of the wireless medium. Due to this the network topology is bound to the changes over time. The knowledge of the network topology is often needed to achieve efficient message transport. The efficiency suffers greatly from topology changes, since with each change the topology must be built newly. Here self-stabilizing algorithms could help to maintain the topology information even in the presence of changes. It is the inherent locality of these algorithms (only operating on their own and their neighbor's states) that promises fast adaptation.

Another interesting aspect is the convergence property of self-stabilizing algorithms. It is shared by other lightweight approaches that aim for achieving scalability. For instance the concept of eventual consistency is a lightweight consistency model first introduced for distributed databases. It does not give guarantees about consistency of copies at every time instance but merely assures that all copies will eventually be consistent, when the time between updates is long enough again. As Gustavsson and Andler point out [14] this approach has several similarities to self-stabilization.

IV. Conclusion

Application scenarios for wireless sensor networks require long living and unattended deployments. These characteristics necessitate fault-tolerant solutions. Self-stabilizing algorithms are an elegant way to develop such applications. We presented the major challenges for the near future, that need to be tackled rendering their utilization in real sensor networks useful. A suitable programming abstraction is vital for allowing a wide range of developers to create self-stabilizing applications. More efficient and lightweight transformations are needed for integrating those algorithms seamlessly into WSNs. Exploiting self-stabilizing properties in other fields than fault tolerance, for achieving a benefit from this elegant paradigm, is another important issue. The greatest challenge will be the integration of self-stabilization into real applications. Therefore, new ideas and experiences with self-stabilizing algorithms in WSNs are tremendously important.

References

[1] E. W. Dijkstra, "Self-stabilizing Systems in Spite of Distributed Control," *Commun. ACM*, vol. 17, no. 11, pp. 643–644, 1974.

[2] S. M. Hedetniemi, S. T. Hedetniemi, D. P. Jacobs, and P. K. Srimani, "Self-Stabilizing Algorithms for Minimal Dominating Sets and Maximal Independent Sets," *Computers and Mathematics with Applications*, vol. 46, no. 5–6, pp. 805–811, Sep. 2003.

[3] C. Weyer, S. Unterschütz, and V. Turau, "Connectivity-aware Neighborhood Management Protocol in Wireless Sensor Networks," in *Proc. 7th GI/ITG KuVS Fachgespräch "Drahtlose Sensornetze" (FGSN'08)*, Berlin, Germany, Jul. 2008.

[4] T. Herman, "Models of Self-Stabilization and Sensor Networks," in *Proc. 5th Int. WS on Distr. Comp. (IWDC'03)*, Dec. 2003.

[5] M. Gradinariu and S. Tixeuil, "Conflict Managers for Self-Stabilization without Fairness Assumption," in *Proc. 27th Int. Conf. on Distr. Comp. Systems (ICDCS'07)*, Jun. 2007.

[6] W. Goddard, S. Hedetniemi, D. Jacobs, and P. K. Srimani, "Anonymous Daemon Conversion in Self-stabilizing Algorithms by Randomization in Constant Space," in *Proc. 9th Int. Conf. on Distr. Comp. and Networking (ICDCN'08)*, Jan. 2008.

[7] V. Turau and C. Weyer, "Fault Tolerance in Wireless Sensor Networks through Self-Stabilization," *Int. Journal of Communication Networks and Distr. Syst.*, vol. 2, no. 1, pp. 78–98, 2009.

[8] H. Kakugawa and T. Masuzawa, "Convergence Time Analysis of Self-Stabilizing Algorithms in Wireless Sensor Networks with Unreliable Links," in *Proc. 10th Int. Symposium on Stabilization, Safety, and Security of Distr. Syst. (SSS'08)*, Nov. 2008.

[9] N. Mitton, E. Fleury, I. Lassous, and B. Sericola, "Fast Convergence in Self-Stabilizing Wireless Networks," in *Proc. 12th Int. Conf. on Parallel and Distr. Syst. (ICPADS'06)*, Jul. 2006.

[10] C. Weyer and V. Turau, "SelfWISE: A Framework for Developing Self-Stabilizing Algorithms," in *Proc. 16th ITG/GI - Fachtg. Komm. in Vert. Syst. (KiVS'09)*, Mar. 2009.

[11] Y. Yamauchi, T. Itou, G. Nishikawa, F. Ooshita, H. Kakugawa, and T. Masuzawa, "Clustering Algorithm for Mobile Ad-Hoc Networks to Improve the Stability of Clusters," in *Proc. IASTED Int. Conf. on Sensor Networks (SN'08)*, Sep. 2008.

[12] C. Weyer, V. Turau, A. Lagemann, and J. Nolte, "Programming Wireless Sensor Networks in a Self-Stabilizing Style," in *Proc. Third Int. Conf. on Sensor Technologies and Applications (SENSORCOMM'09)*, Jun. 2009.

[13] N. Müllner, A. Dhama, and O. Theel, "Derivation of Fault Tolerance Measures of Self-Stabilizing Algorithms by Simulation," in *Proc. 41st Annual Simulation Symposium (ANSS'08)*, April 2008.

[14] S. Gustavsson and S. F. Andler, "Self-Stabilization and Eventual Consistency in Replicated Real-Time Databases," in *Proc. First Workshop On Self-Healing Systems (WOSS'02)*, Nov. 2002.

Solar Power Harvesting - Modeling and Experiences

Daniela Krüger, Stefan Fischer
Institute of Telematics
University of Lübeck
Ratzeburger Allee 160
23538 Lübeck, Germany
{krueger|fischer}@itm.uni-luebeck.de

Carsten Buschmann
coalesenses GmbH
Röntgenstr. 28
23562 Lübeck, Germany
buschmann@coalesenses.com

***Abstract*—As battery capacities are a key limiting factor of wireless sensor networks, harvesting energy from the environment is very attractive. For outdoor applications, solar power seems to be the best suited energy source. However, the amount of energy delivered from the sun changes significantly over the year, which makes the dimensioning of the panel difficult. In this paper we discuss the most important impact factors and introduce a model that predicts the harvested solar power and the battery charge over the year. In addition, we present experimental results of the first six month of our long term experiments for validating our model.**

I. Introduction

Traditionally wireless sensor networks are powered by primary batteries, which limits their lifetime or leads to high maintenance costs induced by exchanging drained batteries. In addition, the limited power source urges extremely low duty cycles, which introduces additional difficulties into the design protocols and applications.

Hence, different power supplies have been discussed - in particular systems that continuously harvest energy from the environment. An overview of potential power sources for wireless sensor networks such as air flow, pressure variation, vibrations, human power and solar energy is given in [1].

We explored solar-powered sensor nodes in the context of the FleGSens project [2], where a prototypic sensor network consisting of 200 iSense sensor nodes [3] for the surveillance of critical areas and properties is designed and set up. The FleGSens project concentrates on ensuring integrity and authenticity of generated alarms caused by trespassers, on robustness against attackers who may compromise a limited number of sensor nodes as well as on assuring availability over a reasonable period of time independent of season or weather. In order to achieve the intended network-lifetime, each node is equipped with a solar cell and a rechargeable battery.

However, solar cells provide energy dependent on their size, orientation to the sun and temperature of the solar module, their output varies heavily over the year. In this paper we present the design considerations we made during our work and summarize our observations to a practical design guide for solar powered systems.

We also present first experimental results to verify our prediction model and show how much energy different panel types yielded and to what extend their output power is influenced by the seasons.

The remainder of this paper is organized as follows. The next section presents related work. In Section III we discuss different impact factors influencing the efficiency and derive a model for predicting the monthly harvested solar energy. Section IV shows our experimental results and discusses their similarity to the model predictions. Finally, we conclude the paper with a summary and directions for future work.

II. Related Work

Much research has yet been done in order to develop energy efficient protocols for sensor networks, but most publications do not consider harvesting technologies. Now that more and more harvesting systems exist researchers increasingly take into account the provided energy when designing protocols. The authors of [4] present a routing protocol for harvesting systems, while [5] describes a statistic-based approach to schedule tasks onto hardware and software. In [6] a real-time scheduling method is discussed that jointly handles constraints from both energy and time domain.

Based on heuristic techniques Kansal et.al. show in [7] and [8] how nodes can learn about their energy environment and use this information for task sharing among nodes. They use an exponentially weighted moving-average (EWMA) as an energy prediction model and adopt they duty cycle in case of over- or underestimation.

In contrast, the authors of [9] investigate in which way the duty cycle should be adapted when the harvested energy is not predictable.

Apart from the aforementioned publications, other authors focus on how energy harvesting systems should be designed. As mentioned above, [1] gives an overview over potential power sources, but discusses each source only briefly without considering different sizes or orientation of solar panels. Furthermore, it shows the differences between secondary battery chemistries like Lithium, NiMHd and NiCd. The authors of [10] discuss advantages and disadvantages of energy storage technologies, too.

Technical issues are also considered in [11] and [12]. The first introduces a power transferring circuit for optimally conveying solar energy into rechargeable batteries. The latter presents a multi-stage energy transfer system using two buffers for energy storage.

III. Expected Power Estimation

The difficulty in deciding which kind of solar panel to choose for powering a sensor node is that the panel manufacturers only provide information on how much energy the panel can deliver under defined laboratory light conditions. These so called standard test conditions (STC) especially include a lighting energy of $100mW/cm^2$. However, usually no indication is given how much solar radiation arrives at the panel over the year.

A. Impact Factors

The main corner stone when modeling the solar power that can be harvested over the year is data regarding the average monthly solar radiation R arriving at the surface.

Figure 1 shows the according data for Hamburg, Germany, stated in mWh/cm^2 per month. It was measured on a surface

Month	Daily solar radiation [mWh/cm^2]	Days in month	Monthly solar radiation [mWh/cm^2]
Jan	85	31	2635
Feb	155	28	4340
Mar	255	31	7905
Apr	360	30	10800
May	440	31	13640
Jun	490	30	14700
Jul	440	31	13640
Aug	430	31	13330
Sep	330	30	9900
Oct	205	31	6355
Nov	105	30	3150
Dec	50	31	1550
Σ		365	101945

Fig. 1. Average monthly solar radiation for Hamburg ($R(M)$).

Month	Temperature corridor exceedance loss
Jan	25%
Feb	10%
Mar	0%
Apr	0%
May	10%
Jun	25%
Jul	25%
Aug	10%
Sep	0%
Oct	0%
Nov	0%
Dec	10%

Fig. 2. Assumed energy loss due to temperature exceedance ($L(M)$).

tilted by $45°$ towards south, yielding a yearly cumulative radiation of $1.020 kWh/m^2$ [13]. The monthly radiation must then be multiplied with the solar panel size A to get the monthly received radiation.

However, only a fraction of the solar radiation can be converted into electrical power. This is due to a number of impact factors that reduce the harvested energy.

First of all, each solar panel features a specific efficiency, i.e. a reduction factor e_{panel} that accounts for the fact that the panel converts only a fraction of the received solar energy into electric power must be introduced.

Second, the radiation angle reduces the harvested energy. While the standard test conditions assumes that the solar radiation hits the panel orthogonally, this is unrealistic for real deployments as the sun moves over the day as well as over the year. Hence, the factor $a = \cos(\alpha)$ must be included, where α is the angular deviation from orthogonal radiation.

Third, if the harvested electric power is passed through a voltage regulator or used for charging a battery, losses will occur here as well, yielding a reduction factor e_{el} accounting for the efficiency of the electronics.

For most WSN applications, the sensor nodes operate in alternating phases of activity and low power sleep modes. During the sleep phases, the nodes dissipate hardly any power, the harvested energy cannot directly be consumed but must be stored. A common way is to use a rechargeable battery, as it can accommodate large amounts of energy. However, an additional difficulty arises when considering charging: common battery technologies exhibit temperature limits to the charge process. For example, lithium-ion batteries can neither be charged below $0°$C nor above $45°$C.

As a result, there will be times during winter when solar power is available but cannot be stored in the battery because it is too cold. The same holds for the summer, when temperatures in the enclosure can exceed the temperature limits especially at noon. Both effects result in a typical monthly temperature corridor exceedance loss L. However, it must be admitted that the influence of the factor is not well-explored yet, the values we assumed for our model are listed in Figure 2.

Finally, the battery capacity deserves some attention. Assuming that the sensor node dissipates more energy than the solar panel can deliver during winter (especially during December, January and February), this deficit can be compensated by energy stored in the battery before (at times when the panel supplied more energy than spent by the node). The larger the battery capacity C, the longer periods of insufficient solar power can be sustained, and the more power can be dissipated during these periods.

B. Model

Considering the impact factors (c.f. Figures 1 to 3) discussed above, we designed a model for predicting the energy that can be harvested with a solar panel as well as for estimating the battery charge development over the year under the condition of a given power dissipation of the sensor node.

Description	Symbol	Value	Unit
Battery capacity	C	21120	mWh
Panel size	A	170	cm^2
Panel Efficiency	e_{panel}	0.07	
Electrical loss	e_{el}	0.7	
Angular loss	a	0.7	
Duty cycle	d	0.179	
Sleeping node power dissipation	P_{sleep}	0.165	mW
Maximum node power dissipation	$P_{running}$	148.5	mW
Average node power dissipation	P_{node}	26.72	mW
Starting month	t_{start}	6	

Fig. 3. Constant parameters with example values.

The harvested solar energy $E_{solar}(M)$ in a certain month $M \in \{1, ..., 12\}$ can be predicted as

$$E_{solar}(M) = (1 - L(M))\, e_{el}\, e_{panel}\, A\, a\, R(M)$$

by considering the temperature exceedance loss of the particular month M, the electrical efficiency, the panel efficiency, the panel size, the loss due to the radiation angle and the amount of solar radiation during M.

Let's assume that the sensor node exhibits a power dissipation of $P_{running}$ at full operation and of P_{sleep} when sleeping. Then, if the node is running at a duty cycle of $d \in [0.0; 1.0]$, i.e. if the node is awake $100\,d$ per cent of the time and sleeps during the rest, the average power dissipation P_{node} is

$$P_{node} = d\, P_{running} + (1 - d)\, P_{sleep}$$

The energy dissipated by the node in a certain month M can then be approximated by

$$E_{dissipate}(M) = P_{node}\, 24\, DiM(M)$$

where DiM yields the number of days in month M.

Now that that all input values are defined, the energy stored in the battery over the course of time can be calculated.

Given that $E(0)$ is the initial battery charge, the energy $E(t)$ at the end of a month can then be estimated by

$$\begin{aligned} E(t) &= min\{C, E(t-1) + E_{solar}(M(t)) - E_{dissipate}(M(t))\} \\ M(t) &= ((t - 2 + t_{start})\ mod\ 12) + 1 \end{aligned}$$

where $t \in \mathbb{N}$ indicates the months since which the system is running and $t_{start} \in \{1, ..., 12\}$ is the starting month of the estimation. The helper function $M\colon \mathbb{N} \Rightarrow \{1, ...12\}$ converts the monotonously growing t into the proper month index according to the starting month t_{start}.

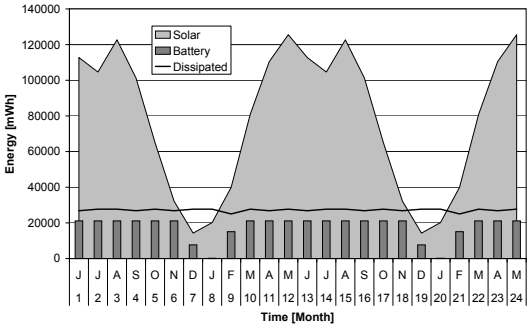
The below Figure shows an example run of the battery energy model. It uses the values given in Figures 1 to 3. Highlighted cells in Figure 4(a) indicate that the node dissipated

more energy than the solar cell harvested, i.e. that it drained the battery. Note that the duty cycle was set to 25% (c.f. Figure 3), which is the maximum that can be sustained over the winter months. Further increasing it would lead to negative values in column three of Figure 4(a), indicating that the sensor node ran out of battery in the according month.

The table data is additionally visualized in Figure 4(b). It becomes obvious that the monthly power dissipation stays more or less constant (and varies only slightly due to the different number of days per month), while the harvested power heavily varies over the year. During times when less power is harvested than dissipated, the battery is drained. Its charge goes down to about 2000mWh in January because of the low harvesting power during winter. As the battery capacity is 21120mWh, the charge graph never exceeds this threshold.

t	$((t-2+t_{start}) \bmod 12)+1$	E(t) [mWh]	E_{solar} [mWh]	$E_{dissipate}$ [mWh]
0		21120		
1	6	21120	104958	24683
2	7	21120	97390	25506
3	8	21120	114211	25506
4	9	21120	94248	24683
5	10	21120	60500	25506
6	11	21120	29988	24683
7	12	8895	13280	25506
8	1	2203	18814	25506
9	2	16350	37185	23038
10	3	21120	75256	25506
11	4	21120	102816	24683
12	5	21120	116868	25506
13	6	21120	104958	24683
14	7	21120	97390	25506
15	8	21120	114211	25506
16	9	21120	94248	24683
17	10	21120	60500	25506
18	11	21120	29988	24683
19	12	8895	13280	25506
20	1	2203	18814	25506
21	2	16350	37185	23038
22	3	21120	75256	25506
23	4	21120	102816	24683
24	5	21120	116868	25506

(a) Table representation



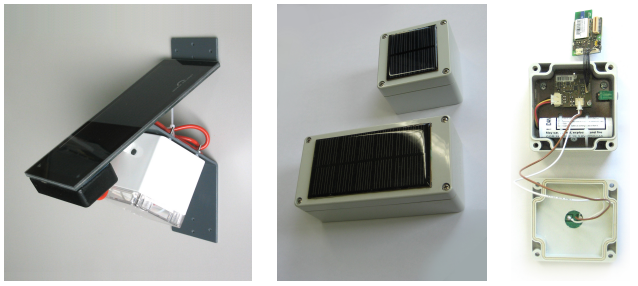
(b) Graph representation

Fig. 4. Battery energy prediction over a 24 month period.

IV. Experimental Results

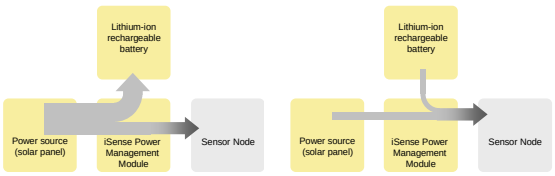
To validate the model, we started an experimental evaluation in December 2008. We used iSense sensor nodes [3] that were connected to three different types of solar cells (Figure 5).

As shown in Figure 5(c), the nodes were equipped with a special power management module, a lithium ion rechargeable battery and a solar panel. The power management module distributes the power provided by the solar panel in an intelligent way. If the panel can deliver more power than the sensor node requires, it charges the lithium ion battery (c.f. Figure 6(a)). Otherwise, it reduces the battery drainage by supplying the node with the solar power (c.f. Figure 6(b)) as much as possible and drawing the rest from the battery.



(a) Large. (b) Medium, Small. (c) Inside view.

Fig. 5. Panel types and node setup



(a) Charging (b) Reducing discharge

Fig. 6. Energy Flows.

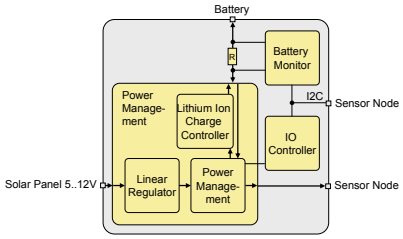


Fig. 7. Power Management Concept.

Figure 7 shows a conceptual view of the solar power management module. The Solar power is fed into the power management component through a linear regulator. For charging the battery, a charge controller in integrated as well. The battery current flows into and out of the battery are monitored and logged, the battery monitor also accumulates the currents during charging and discharging cycles, and hence provides precise information about the energy currently stored within the battery.

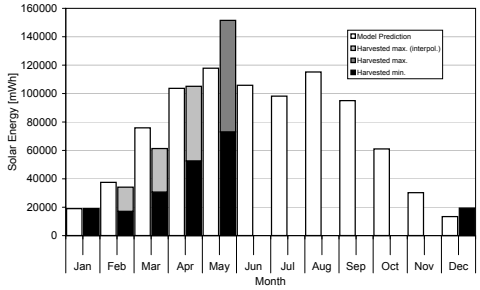
	Large Panel	Medium Panel	Small Panel
Panel efficiency	0.09	0.12	0.11
Panel size	170	81.25	37.05
Open circuit voltage at MPP	6	9	5
Short circuit current at MPP	250	109	81
Electrical efficiency	0.8	0.4	0.63
Radiation angle efficiency	0.8	0.7	0.7

Fig. 8. Technical cell data and model settings.

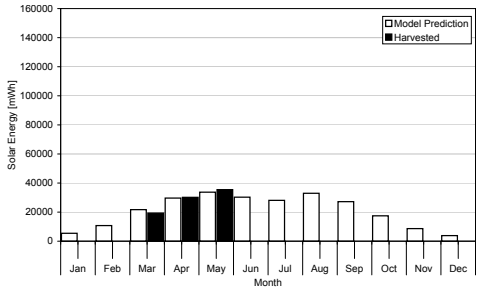
The table in Figure 8 summarizes some technical data of the solar panels used as well as the model settings used below.

Figure 9 shows both predicted and measured harvested energy for the three panel types. The values predicted by the model are always indicated by white bars, while the dark bars indicate the energy harvested in reality. Note that so far, real-world data is available for a small number of months only.

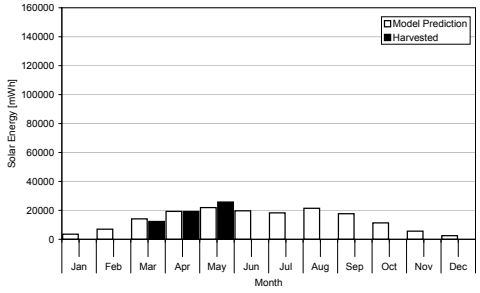
The different dark bars in Figure 9(a) need some further explanation. The black bars indicate the harvested energy by our first test node equipped with a large solar panel. It can



(a) Large Panel



(b) Medium Panel



(c) Small Panel

Fig. 9. Experimental Results.

be seen that prediction and measured values highly resemble during the first two months of our experiment - December and January. After that - from February to May - the according harvesting results are pretty disappointing. In April we found the reason for this: Because the battery was fully charged most of the time, only a fraction of the available solar energy could be harvested.

In order to find out how much energy could really be harvested, we employed additional sensor nodes in May and ensured that at least one of them at a time harvested the full solar energy into an empty battery. The according amount of energy harvested in May is indicated as the dark gray *Harvested max* bar. We then interpolated the energy that could have been harvested from February to April and indicated it with the light gray *Harvested max (interpolated)* bars.

The harvested energy of the smaller solar panels is shown in Figure 9(b) and 9(c). Prediction model and measured values highly match even though both devices harvested a bit more than expected in May.

V. Conclusion

Supplying a sensor network with solar energy promises nearly perpetual operation, but several impact factors significantly influence the amount of potentially harvested energy and must be taken into account when design decisions are made. We presented a model that allows to predict both the harvested energy as well as the corresponding battery charge. We verified our model by long term experiments with different solar panels whose results are also shown. Even though the energy harvested in reality basically follows the predictions of the model, further work is needed.

The first results presented here for example hint that the values assumed for the temperature exceedance loss are not very realistic. However, the experiments will provide additional data that will help to improve the model.

In addition, we are planning to implement a duty cycle control system that is based upon the models presented here.

References

[1] S. Roundy, D. Steingart, L. Frechette, P. Wright, and J. Rabaey, "Power sources for wireless sensor networks," in *Wireless Sensor Networks, First European Workshop, EWSN*, ser. LNCS, no. 2920. Springer, 2004, pp. 1–17.

[2] P. Rothenpieler, D. Krüger, D. Pfisterer, S. Fischer, D. Dudek, C. Haas, A. Kuntz, and M. Zitterbart, "Flegsens - secure area monitoring using wireless sensor networks," *Proceedings of World Academy of Science, Engineering and Technology*, 2009.

[3] C. Buschmann and D. Pfisterer, "iSense: A modular hardware and software platform for wireless sensor networks," 6. Fachgesprch Drahtlose Sensornetze der GI/ITG-Fachgruppe Kommunikation und Verteilte Systeme, Tech. Rep., 2007. [Online]. Available: www.coalesenses.com

[4] T. Voigt, H. Ritter, and J. Schiller, "Utilizing solar power in wireless sensor networks," in *LCN '03: Proceedings of the 28th Annual IEEE International Conference on Local Computer Networks*. Washington, DC, USA: IEEE Computer Society, 2003, p. 416.

[5] A. Nahapetian, P. Lombardo, A. Acquaviva, L. Benini, and M. Sarrafzadeh, "Dynamic reconfiguration in sensor networks with regenerative energy sources," in *DATE '07: Proceedings of the conference on Design, automation and test in Europe*. San Jose, CA, USA: EDA Consortium, 2007, pp. 1054–1059.

[6] C. Moser, D. Brunelli, L. Thiele, and L. Benini, "Real-time scheduling for energy harvesting sensor nodes," *Real-Time Syst.*, vol. 37, no. 3, pp. 233–260, 2007.

[7] A. Kansal and M. B. Srivastava, "An environmental energy harvesting framework for sensor networks," in *ISLPED '03: Proceedings of the 2003 international symposium on Low power electronics and design*. New York, NY, USA: ACM, 2003, pp. 481–486.

[8] A. Kansal, J. Hsu, S. Zahedi, and M. B. Srivastava, "Power management in energy harvesting sensor networks," *ACM Trans. Embed. Comput. Syst.*, vol. 6, no. 4, p. 32, 2007.

[9] C. M. Vigorito, D. Ganesan, and A. G. Barto, "Adaptive control of duty cycling in energy-harvesting wireless sensor networks." in *SECON*. IEEE, 2007, pp. 21–30. [Online]. Available: http://dblp.uni-trier.de/db/conf/secon/secon2007.html#VigoritoGB07

[10] V. Raghunathan, A. Kansal, J. Hsu, J. Friedman, and M. Srivastava, "Design considerations for solar energy harvesting wireless embedded systems," in *IPSN '05: Proceedings of the 4th international symposium on Information processing in sensor networks*. Piscataway, NJ, USA: IEEE Press, 2005, p. 64.

[11] C. Alippi and C. Galperti, "An adaptive system for optimal solar energy harvesting in wireless sensor network nodes," in *IEEE-Transactions on Circuits and Systems: Part I, Fundamental theory and applications*, no. 55(6), Jul. 2008, p. 17421750.

[12] X. Jiang, J. Polastre, and D. Culler, "Perpetual environmentally powered sensor networks," in *IPSN '05: Proceedings of the 4th international symposium on Information processing in sensor networks*. Piscataway, NJ, USA: IEEE Press, 2005, p. 65.

[13] Fachverlag für Energie-Markting und -Anwendungen, "Photovoltaik - Strom aus der Sonne," 2004.

Lifetime Prediction for Supercapacitor-powered Wireless Sensor Nodes

Christian Renner, Jürgen Jessen, and Volker Turau
Institute of Telematics
Hamburg University of Technology
Hamburg, Germany
{christian.renner, juergen.jessen, turau}@tu-harburg.de

***Abstract*—Energy-aware task scheduling is a novel research direction for wireless sensor networks. It depends on accurate models for lifetime prediction. In other terms, nodes must be aware of present and future energy resources. This paper addresses the first step towards reaching this goal: It explores discharging-characteristics of supercapacitors, discusses analytical discharging-models for lifetime prediction, and evaluates these models by comparing them with real discharging curves.**

I. Motivation

In the recent past, energy-efficiency has become a major research topic in the field of wireless sensor networks. Though it can prolong a sensor node's lifetime, energy depletion will eventually emerge long before the desired date. Since battery capacities are not expected to rise in orders of magnitude within the near future, provided that larger-sized batteries are not an option, and since replacement of batteries is usually infeasible, a different card must be drawn.

Within the last couple of years, the potential of harvesting energy from the environment has become more and more attractive. Various harvesting solutions are possible, among the sources being light, radio frequency, wind, vibration, or temperature difference. Here, sunlight is highly promising, since it produces a sufficient amount of energy to supply wireless sensor nodes, which draw currents between several μA in the sleep state and some mA in full-operation mode.

Yet, sunlight—but also other sources—have the drawback of not harvesting energy continuously. Furthermore, the amount of energy produced may vary significantly depending on the environmental conditions. This leads to the necessity to buffer energy, so that nodes do neither suffer from temporal energy depletion nor is their operation restricted to periods of incoming energy.

Despite the rich bouquet of energy buffers available, most of them reveal a considerable shortcoming: the amount of energy stored cannot be estimated easily. However, this ability is advantageous or even mandatory, as it allows for adaptive duty-cycling or may shrink the chance of accidental energy depletion caused by running a highly energy-consuming task during periods of low energy reserves. Energy-awareness may also allow for performing these tasks during periods of energy excess. Thus, energy-aware task scheduling becomes possible.

In the recent years electric double-layer capacitors with high capacities have become available. They fill the gap between capacitors and rechargeable batteries and can store enough energy to keep up-to-date sensor nodes alive for a couple of days. Their main advantage over rechargeable batteries is the high number of possible charge-discharge cycles. While a lifetime of 2-3 years can be expected for lithium-ion polymers, supercapacitors can last for 10 years or even more. Supercapacitors do not need a complex charging circuit and render easy estimation of their energy reserves possible.

Examples of supercapacitors are Panasonic GoldCaps [1] and SAMWHA GreenCaps [2], which we have used in a solar energy-harvesting power-supply. In this paper, we will present our first experiences on this matter. We will develop and assess models for energy estimation, enabling node lifetime prediction. These models build up the cornerstone of a more complex system that will be developed in future research. This system is compassed to enrich our model with a prediction of future incoming energy, e.g., obtained from a solar cell.

II. Related work

Several approaches for self-sustaining power supplies for wireless sensor nodes exist. The Enviromote [3] is using a solar cell as power source and NiMH batteries for energy storage. Among the design goals are easy circuit design and cheap energy storage devices. The authors present charging and discharging characteristics of their power supply.

A solar harvesting supply with lead acid batteries for the IRIS [4] platform is developed in [5]. It targets at large capacity energy storage and keeps the solar cell at a static maximum power point.

Prometheus [6], also employing a solar cell, is based on a two-stage energy storage system. A supercapacitor serves as the primary energy source, which supplies the sensor node and limits access to the secondary source, a rechargeable Li^+ battery, to prolong the lifetime of the latter. Charging and discharging behavior of the circuit and supercapacitors are examined. A striking observation is that—for the presented power supply—a 22 F supercapacitor outperforms its 10 F and 50 F counterparts. The authors also take a first step into the direction of energy-aware scheduling by adapting the duty cycle to the current supercapacitor voltage. Prometheus has been successfully deployed in the Trio testbed [7].

Another approach is found in [8]. The Everlast platform stores energy obtained from a solar cell in a supercapacitor

solely. In order to increase the efficiency of the solar cell, i.e., to maximize the amount of energy available, maximum power point tracking (MPPT) is performed. The authors claim that their platform can operate for as long as 20 years while preserving high data rates.

III. Energy Harvesting Platform

As a first step on our road to energy-aware task scheduling on wireless sensor nodes, we have developed a prototype of a solar energy-harvesting platform as depicted in Fig. 1. It supplies an IRIS sensor node from Crossbow Technology. This prototype uses a solar cell as its energy source. The solar cell is currently feeding a supercapacitor via a simple circuit consisting of a Schottky-Diode to avoid discharge during cloudy periods or at night and a Zener-Diode to prevent overcharging of the supercapacitor. Here, the charging maximum is limited to approximately 2.3 V, which is the specific maximum voltage allowed for GoldCaps.

Fig. 1. Energy Harvesting Platform for the IRIS node

The discharging circuit consists of the DC-DC buck-boost converter TPS61221 from Texas Instruments [9], which ensures a stable and constant supply voltage of 3.3 V. The converter starts converting at an input voltage of 0.7 V and has an efficiency of up to 95%. This converter has been selected because of its low quiescent current of 5.5 μA and its high efficiency even at low currents. Figure 2 depicts the efficiency for different output currents and input voltages. The input voltage is equal to the supercapacitor voltage, which will range from 0.7 V to 2.3 V. For node-operation in normal mode with a current of a few mA the efficiency of the converter will be higher than 80%. Even in sleep mode with a current of a few μA, the efficiency will remain well above 60%. Most other DC-DC-converters, built for high efficiency at larger currents, can achieve an efficiency of 10% for low currents only.

IV. Lifetime Prediction

In this section, models for predicting supercapacitor voltage V_C and thus estimating node lifetime will be derived.

A. Simple Model

As a first step we analyze the temporal behavior of the supercapacitor voltage V_C in a simple model, i.e., we neglect self-discharge. Figure 3 illustrates the simplified circuit, which only consists of the supercapacitor with capacitance C, the DC-DC-converter, and the sensor node.

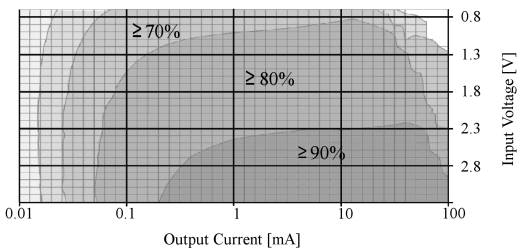


Fig. 2. Efficiency of the Texas Instruments DC-DC-converter TPS61221 [9]

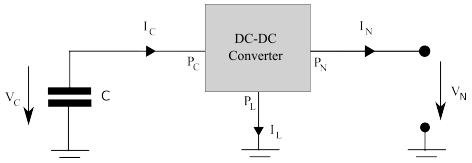


Fig. 3. Simplified Discharge Circuit

Due to conversion losses and the current I_L consumed by the DC-DC-converter, the input power P_C is larger than the output power P_N:

$$P_\mathrm{N} = P_\mathrm{C} - P_\mathrm{L} = \eta \cdot P_\mathrm{C} = \eta \cdot V_\mathrm{C} \cdot I_\mathrm{C}, \tag{1}$$

where η is the efficiency of the converter. Note that we assume a constant power state of the node, i.e., the current I_N consumed by the node is constant. In addition, the voltage V_N provided by the DC-DC-converter is stable and constant, so that $P_\mathrm{N} = V_\mathrm{N} \cdot I_\mathrm{N} = \text{const}$.

Supercapacitors behave like normal capacitors with

$$I_\mathrm{C} = -C \cdot \dot{V_\mathrm{C}}, \tag{2}$$

so that we can combine (1) and (2) to

$$P_\mathrm{N} = -\eta \cdot V_\mathrm{C} \cdot C \cdot \dot{V_\mathrm{C}}. \tag{3}$$

For simplicity, we assume η to be constant, so that the differential equation (3) can be solved as follows

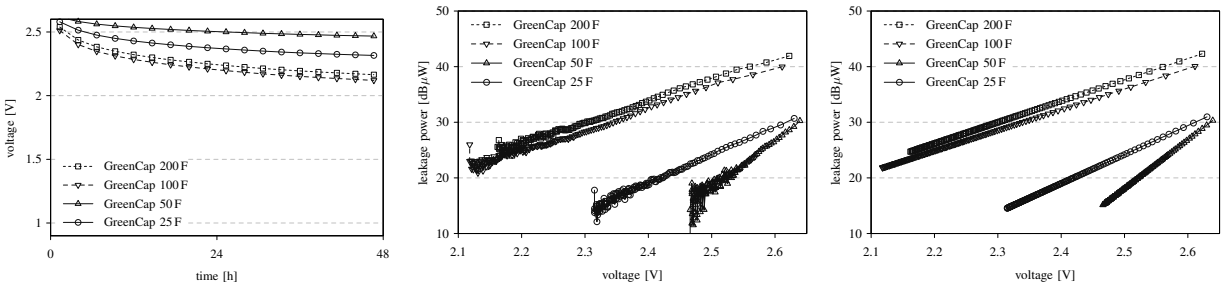
$$\int_{t_0}^{t_0+T_\mathrm{life}} \mathrm{d}t = -\frac{\eta C}{P_\mathrm{N}} \int_{V_\mathrm{C,0}}^{V_\mathrm{min}} V_\mathrm{C} \mathrm{d}V_\mathrm{C} \tag{4}$$

Here, the current time is denoted t_0, and $V_\mathrm{C,0}$ is the voltage of the supercapacitor at this time. The minimum voltage V_min is required by the DC-DC-converter for proper and reliable function. The elapsed time until V_C has dropped to V_min is T_life; the latter will be referred as the expected lifetime at time t_0. Finally, solving (4) yields

$$T_\mathrm{life} = \frac{\eta C}{2P_\mathrm{N}} \left(V_\mathrm{C,0}^2 - V_\mathrm{min}^2 \right). \tag{5}$$

B. Leakage Current

The previously developed model must be extended, if the current I_C drawn from the supercapacitor drops to a value close to the leakage current I_leak of the supercapacitor. We expect that this will be the case for low duty cycles of the



(a) GreenCaps, Self-Discharge (b) GreenCaps, Leakage Power (c) GreenCaps, Leakage Power Estimation

Fig. 4. Supercapacitor self-discharge behavior of GoldCaps and GreenCaps

attached sensor node—e.g., the IRIS platform draws a current $I_{\mathrm{N}} \approx 20\,\mu\mathrm{A}$ in the sleeping-mode [5].

We have recorded self-discharge time-voltage curves of different supercapacitors. Figure 4a shows the supercapacitor voltage of GreenCaps with capacities between 25 and 200 F. All capacitors have been charged close to the maximum allowed voltage of 2.65 V. It is remarkable that self-discharge is highly correlated with the voltage. This behavior matches the one for a different model shown in [1] and our additional recordings for GoldCaps (that are left out for brevity).

From these recordings, the leakage power of the supercapacitors can be approximated numerically from

$$E(V_{\mathrm{C}}) = \frac{CV_{\mathrm{C}}^2}{2} \;\Rightarrow\; P_{\mathrm{leak}}(V_{\mathrm{C}}) \approx \frac{\Delta E(V_{\mathrm{C}})}{\Delta t} = \frac{C\Delta V_{\mathrm{C}}^2}{2\Delta t}, \quad (6)$$

where ΔV_{C}^2 is the difference of V_{C}^2 at time t and V_{C}^2 at time $t + \Delta t$. The corresponding results are shown in Fig. 4b. The noise in the lower voltage regions is due to the noisy measurement of the slowly decreasing voltage. Note that power is shown in logarithmic scale, giving rise to an exponential behavior of leakage power:

$$P_{\mathrm{leak}} \approx P_0 \cdot \exp\left(\alpha V_{\mathrm{C}}\right). \quad (7)$$

We have determined estimations according to (7) for all of the tested supercapacitors using least squares. The results are displayed in Fig. 4c. The estimations follow the numerical approximation of P_{leak} closely.

C. Refined Model

Taking leakage power into consideration, (1) becomes

$$P_{\mathrm{C}} - P_{\mathrm{leak}} = P_{\mathrm{N}} + P_{\mathrm{L}}, \quad (8)$$

and therefore (3) has to be rewritten using (7) as

$$P_{\mathrm{N}} = -\eta \cdot V_{\mathrm{C}} \cdot C \cdot \dot{V}_{\mathrm{C}} - P_0 \cdot \exp\left(\alpha V_{\mathrm{C}}\right). \quad (9)$$

The solution to this equation can be found using mathematical software, such as Maple:

$$T_{\mathrm{life}} = -\frac{\eta C}{2P_{\mathrm{N}}}\left[V_{\mathrm{C}}^2 - \frac{2V_{\mathrm{C}}}{\alpha}\ln\left(1 + \frac{P_0\exp\left(\alpha V_{\mathrm{C}}\right)}{P_{\mathrm{N}}}\right) - \frac{2}{\alpha^2}\sum_{n=1}^{\infty}\left(-\frac{P_0\exp\left(\alpha V_{\mathrm{C}}\right)}{P_{\mathrm{N}}}\right)^n \frac{1}{n^2}\right]_{V_{\mathrm{C},0}}^{V_{\min}} \quad (10)$$

Unfortunately, this equation is highly complex due to the ln, exp, and the dilog (the infinite sum). It is thus not suitable to be evaluated on sensor nodes. However, it is a good first step in order to gain insight into realistic discharging behavior of nodes running on supercapacitors. Simplification of the equation will be future work.

V. Model Evaluation

In this section the models developed in Sect. IV are checked against discharging curves recorded for duty cycles ϑ of 1, 10, and 100% on the IRIS platform. Figures 5a to 5c show the remaining supercapacitor lifetime, i.e., the time elapsed until V_{C} falls below $V_{\min} = 0.9\,\mathrm{V}$. Smaller values of $V_{\min}$, as proposed in Sect. III, lead to a too low output voltage V_{N}.

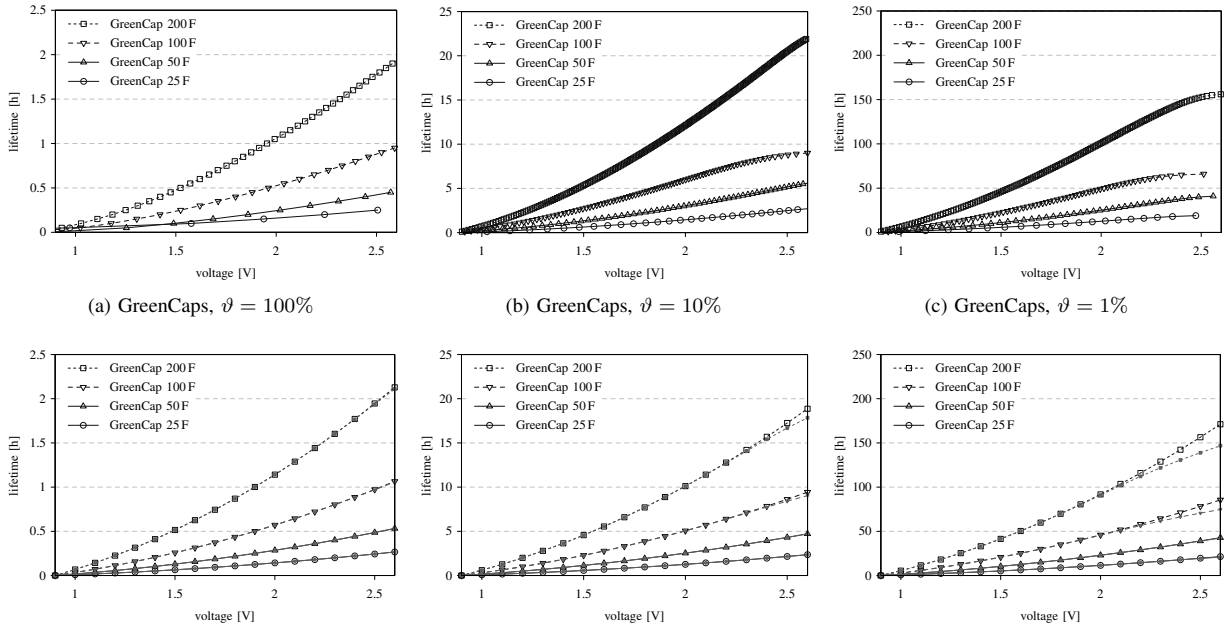
The results show that—as expected in Sect. IV-B—self-discharge has an impact on remaining lifetime for low duty cycles, while it can be neglected for high ones. This is indicated by the dip for large values of V_{C} in case of $\vartheta = 1\%$ in Fig. 5c as opposed to the behavior for $\vartheta = 100\%$ in Fig. 5a. As a result, lifetime would be dramatically overestimated for low duty cycles, if the simplified model were used.

Knowing about the real discharging behavior, we have computed lifetime predictions using (5) and (10). Having a constant node supply voltage of $V_{\mathrm{N}} = 3.3\,\mathrm{V}$, we used $I_{\mathrm{N,act}} = 20\,\mathrm{mA}$ for the active and $I_{\mathrm{N,sleep}} = 20\,\mu\mathrm{A}$ for the sleeping mode and averaged P_{N} according to the duty cycle ϑ :

$$P_{\mathrm{N}} = V_{\mathrm{N}}\left(\vartheta \cdot I_{\mathrm{N,act}} + (1-\vartheta)\cdot I_{\mathrm{N,sleep}}\right)$$

Based on Fig. 2 we assume an efficiency $\eta_{\mathrm{act}} = 85\%$ for the active and $\eta_{\mathrm{sleep}} = 75\%$ for the sleeping mode and averaged η as we did with P_{N}.

The prediction results are displayed in Fig. 5d through 5f. The curves with the open markers have been computed using the simplified model, whereas their filled counterparts come from the refined one. The results reveal that the models give accurate predictions for a large duty cycle and for low values of V_{C} in case of a low duty cycle. For $\vartheta = 1\%$ the two models show significant differences for a large V_{C}. During evaluation, we experienced that the dilog term in the refined model has only a marginal influence on prediction accuracy and can thus be omitted. The combined ln-exp term, however,



(a) GreenCaps, $\vartheta = 100\%$ (b) GreenCaps, $\vartheta = 10\%$ (c) GreenCaps, $\vartheta = 1\%$

(d) GreenCaps, $\vartheta = 100\%$ (e) GreenCaps, $\vartheta = 10\%$ (f) GreenCaps, $\vartheta = 1\%$

Fig. 5. Measured vs. predicted node lifetime for GreenCaps for duty cycles $\vartheta = 1\%, 10\%, 100\%$

should not be dropped, as it gives better prediction of T_{life} if the supercapacitors are almost fully charged.

Although we have solely calculated averages for P_{N} and η and taken rough estimates for the values in the two node states (sleeping and active), the curves follow the realistic ones considerably well and are thus promising. Yet, fine-tuning of the parameters may be required, as soon as the models are simplified, as this step already introduces prediction errors. In contrast, it appears that a more detailed modeling of the power-states and the DC-DC efficiency may not be required.

VI. Conclusion and Next Steps

In this paper, we have presented models for predicting the lifetime of wireless sensor nodes using a supercapacitor-based power supply. These models have been evaluated using real discharging behavior of this power supply and found to match the real discharging behavior closely.

The estimation and prediction models derived in this paper serve as a first groundwork. However, the influence of temperature or supercapacitor age has not been taken into consideration thus far. Hence, our models must be refined. In contrast to this, the parts of the models involving difficult to evaluate mathematical expressions—for low-power, low-resource hardware—must be simplified, while preserving as much preciseness as possible. This will be a major part of future work.

In addition, we will focus on self-configuration, i.e., nodes should become capable to determine and update the model parameters on their own. We will also equip our hardware with an effective charging circuit and derive models for estimating incoming power. Finally, these models will be combined with the discharging ones, thus yielding a sophisticated base for energy-aware task scheduling.

Acknowledgment

The authors would like to thank Jesús Álvarez Álvarez for his support in recording supercapacitor discharging curves.

References

[1] Panasonic, *Gold Capacitors Technical Guide*. [Online]. Available: http://www.panasonic.com/industrial/components/pdf/goldcap_techguide_052505.pdf

[2] SAMWHA, *Gold Capacitors Technical Guide*. [Online]. Available: http://www.samwha.com/electric/templatedirs/guest/list_pdf1/DP.pdf

[3] V. Kyriatzis, N. S. Samaras, P. Stavroulakis, H. Takruri-Rizk, and S. Tzortzios, "Enviromote: A New Solar-Harvesting Platform Prototype for Wireless Sensor Networks / Work-in-Progress Report," in *Proc. of the Annual IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC '07)*, Athens, Greece, 2007.

[4] Crossbow, *IRIS Wireless Measurement System - Datasheet*. [Online]. Available: http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/IRIS_Datasheet.pdf

[5] C. Lange, "Energiegewinnung für drahtlose Sensorknoten (Diploma Thesis)," Master's thesis, Hamburg University of Technology, Oct. 2008.

[6] X. Jiang, J. Polastre, and D. Culler, "Perpetual Environmentally Powered Sensor Networks," in *Proc. of the Intl. Symposium on Information Processing in Sensor Networks (IPSN '05)*, Los Angeles, CA, USA, 2005.

[7] P. Dutta, J. Hui, J. Jeong, S. Kim, C. Sharp, J. Taneja, G. Tolle, K. Whitehouse, and D. Culler, "Trio: Enabling Sustainable and Scalable Outdoor Wireless Sensor Network Deployments," in *Proc. of the Intl. Conference on Information Processing in Sensor Networks (IPSN '06)*, New York, NY, USA, 2006.

[8] F. Simjee and P. H. Chou, "Everlast: Long-Life, Supercapacitor-Operated Wireless Sensor Node," in *Proc. of the Intl. Symposium on Low Power Electronics and Design (ISLPED '06)*, Tegernsee, Germany, 2006.

[9] T. Instruments, *Datasheet TPS 61221*, Jan. 2009. [Online]. Available: http://focus.ti.com/lit/ds/symlink/tps61221.pdf

Energetic and Temporal Analysis of a Desynchronized TDMA Protocol for WSNs

Clemens Mühlberger
Chair of Computer Science V
University of Würzburg
Am Hubland, 97074 Würzburg
Email: muehlberger@informatik.uni-wuerzburg.de

***Abstract*—These days, wireless radio communication of sensor nodes is still very power consuming. Thus lots of MAC protocols yet exist to manage the collision free access to the common transmission medium with respect to energy consumption. In this paper, we focus on the biologically inspired and self-organized TDMA protocol DESYNC. We analyze its potential for energy-saving, and network latency respectively. We could identify some parameters, which may help network designers to adjust the DESYNC protocol according to their preferences, i.e. to save more energy or to get a lower latency.**

I. Motivation

One characteristic feature of Wireless Sensor Networks (WSNs) is the shared transmission medium of interacting sensor nodes. Concurrent assignment of common radio channels might cause loss of data due to packet collisions which require retransmission and consume additional energy. That's why the access of each node to the medium has to be coordinated carefully. Several Medium Access Control (MAC) protocols for WSNs already exist, amongst others a couple of Time Division Multiple Access (TDMA) protocols. They divide the radio channel into *time slots* offering collision free medium access for a specific node, like Z-MAC [1], TRAMA [2] or HashSlot [3]. Here we analyze the decentralized but self-organized DESYNC protocol [4], [5] with respect to energy savings and network latency.

In the next section we briefly introduce the decentralized TDMA protocol DESYNC and its underlying paradigm of desynchronization. In Section III we identify the potential for energy-savings, whereas Section IV discusses the emanating changes in network latency. Section V closes this paper with a short conclusion and an outlook to further research.

II. Introduction to the DESYNC Protocol

The biologically inspired paradigm of desynchronization [6] denotes the equidistant distribution in time of *oscillators*, for example periodically transmitting sensor nodes. Based upon this, Degesys et al. [4] developed the DESYNC protocol, a self-organized TDMA protocol for single-hop topologies. Because real-world deployments usually contain multi-hop topologies, an extended version of the DESYNC protocol is subject to current research (cf. [5]).

First of all, each element of the set N of nodes has a unique identifier i and oscillates at an identical frequency ω within the common period $T = \frac{1}{\omega}$. The period T must be long enough to provide at least one time slot for each of the n participating nodes, e.g. for single-hop topologies $n = |N|$, for multi-hop topologies n equals the cardinality of the maximum clique of two-hop neighbors. Next, the communication links are symmetrical and each node uses Carrier Sense (CS) just before transmission to avoid collisions in the first place.

To fulfill the paradigm of desynchronization, i.e. to spread out the time slots of all participating nodes equidistantly, each node i of the WSN tries to maximize the time lag relative to its neighbors. Therefore, the phase $\phi_i \in [0.0, 1.0]$ of a node i denotes the elapsed time since its last transmission normalized to T, e.g. $\phi_3 = 0.7$ means, that node 3 has already finished 70 % of its current period. When node i finishes its period, i.e. $\phi_i = 1.0$, it broadcasts a so called *firing packet* and immediately resets its phase to $\phi_i = 0.0$. The column vector $\overrightarrow{\phi} = \left[\phi_1 \cdots \phi_i \cdots \phi_{|N|}\right]^T$ describes the global system state, i.e. the phases of all nodes.

Two nodes are of special interest for node i: the *previous phase neighbor* $p(i)$ broadcasts its firing packet just before, whereas the *successive phase neighbor* $s(i)$ broadcasts its firing packet just after node i. Hence, node i can calculate the midpoint of its phase neighbors as

$$\mathrm{mid}(\phi_{s(i)}, \phi_{p(i)}) = \frac{\phi_{p(i)} + \phi_{s(i)}}{2}$$

and finally estimate its new phase ϕ_i' unassisted by itself as

$$\phi_i' = (1 - \alpha) \cdot \phi_i + \alpha \cdot \mathrm{mid}(\phi_{s(i)}, \phi_{p(i)}).$$

The jump size parameter $\alpha \in (0.0, 1.0)$[1] regulates how fast a node moves towards the assumed midpoint of its phase neighbors.

The stable state, when each node has the same temporal distance to its phase neighbors and thus the times of firing do not change anymore (unless the system changes), is called *desynchrony*. The convergence to desynchrony for single-hop topologies was proved in [4]. Figure 1 exemplifies the progress of desynchronization for a single-hop topology consisting of five sensor nodes.

[1]If $\alpha = 0.0$, there's no movement at all, and, according to [7], $\alpha = 1.0$ forces straight movement onto the midpoint under unstable emergence of new configurations. Thus, a reliable value would be $\alpha \approx 0.9$.

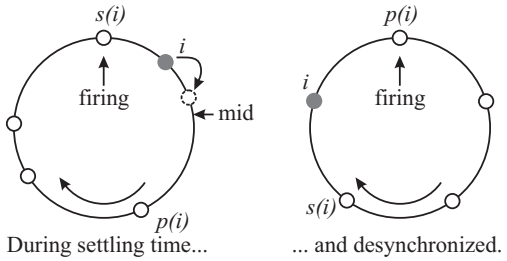


Fig. 1. Snapshots of the desynchronization progress

For a successful operation in real-world deployments, the DESYNC protocol requires an extension for multi-hop topologies as well as further improvements, e.g. back-off algorithms for concurrent start-ups of nearby nodes, or unreliable links. But space does not permit a discussion of that here, that's why we just analyze potential energy savings and network latency within this paper.

III. Energy

Still, the energy consumption of radio transceivers used at sensor nodes is much higher than that of current microcontrollers. Thus, to save much energy – especially at periodically transmitting sensor nodes – the radio controller has to be switched off as often and as long as possible. Since the comprehensive and constant period T depends on the maximum number n of supported nodes, we divide T into n *frames* $F(i)$ of equal size f, i.e. $f = |F(i)|$ for any $i \in \{1, \ldots, n\}$. Similar to other protocols like LMAC [8] or Crankshaft [9], each frame $F(i)$ again is subdivided into k *slots* $F(i, j)$, where $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, k\}$. The first slot $F(i, 1)$ is reserved for the firing packet of node i (cf. Section II), whereas the remaining slots for $k \geq 2$ can be used for further data transmissions, if desired. Please note, for $k = 1$ the length of a frame equals the length of its single firing slot, i.e. $f = |F(i, 1)|$.

To avoid collisions when nodes (re)join the network dynamically, and to compensate potential but individual clock drifts or other hardware or software delays, a safety gap $\sigma = \varepsilon \cdot |F(i, 1)|$ is prefixed to each firing slot $F(i, 1)$ (cf. Fig. 2). The factor ε should be selected carefully, because it shall cover possible drifts but not delay the firings of the nodes unnecessarily. Since there need not be any data slot, we made σ to be a function of the length of a firing slot. Finally, the period T has to hold

$$\begin{aligned} T &= n \cdot (\sigma + f) \\ &= n \cdot \big(\sigma + |F(i,1)| + \sum_{j=2}^{k} |F(i,j)|\big) \\ &= n \cdot \big((1+\varepsilon) \cdot \underbrace{|F(i,1)|}_{\text{firing slot}} + \underbrace{\sum_{j=2}^{k} |F(i,j)|}_{\text{data slot(s)}}\big) \end{aligned} \tag{1}$$

Because parameter n is specified by the network and the length of a firing packet dominates the safety gap σ but can only be influenced marginally as well, the only way to save energy is determined by the sum of the lengths of the data slots. Of course, the period T can be padded out to prolong the sleep periods of a radio transceiver, but for our further examinations we just consider the minimum value for period T specified by equation 1.

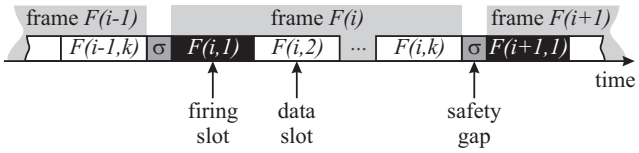


Fig. 2. Arrangement of frames, slots and safety gaps

To stay desynchronized, a node $i \in N$ must be just interested in the firing packets of its $n - 1$ neighbors, thus it has to turn on its radio transceiver for at least

$$\Delta t_{i,RF} = \Delta t_{i,RX} + \Delta t_{i,TX},$$

where $\Delta t_{i,TX} = \sigma + |F(i, 1)|$ denotes the duration[2] for broadcasting a firing packet, and $\Delta t_{i,RX} = (n-1) \cdot (1+\varepsilon) \cdot |F(i, 1)|$ terms the elapsed time for reception of the firing packets of all its neighbors. With it, the uptime of the radio unit of node i is

$$\Delta t_{i,RF} = n \cdot (1 + \varepsilon) \cdot |F(i, 1)| \leq T. \tag{2}$$

Assuming all data slots have the same length f_k, i.e. for all $i \in \{1, \ldots, n\}$ and $m \in \{2, \ldots, k\}$ holds $f_k = |F(i, m)|$, the gain of energy γ_i compared to an always activated radio controller of a node i in percent per period T is

$$\begin{aligned} \gamma_i &= \frac{T - \Delta t_{i,RF}}{T} \\ &= 1 - \frac{n \cdot (1+\varepsilon) \cdot |F(i,1)|}{n \cdot \big((1+\varepsilon) \cdot |F(i,1)| + (k-1) \cdot f_k\big)} \\ &= \frac{(k-1) \cdot f_k}{(1+\varepsilon) \cdot |F(i,1)| + (k-1) \cdot f_k} \end{aligned} \tag{3}$$

If we additionally suppose that firing slots and data slots are of the same length, i.e. for all $i \in \{1, \ldots, n\}$ holds $|F(i, 1)| = f_k$, equation 1 reduces to

$$T = n \cdot (k + \varepsilon) \cdot f_k,$$

and furthermore equation 3 simplifies to

$$\gamma_i = \frac{k-1}{k+\varepsilon}$$

That means, if there are only firing slots but not a single data slot (i.e. $k = 1$), and if the relevant number of neighbors is at maximum support for period T, it is not possible to save energy by reducing the uptime of the radio unit.

[2]Before transmission, each node has to use CS to detect joining nodes or drifting neighbors. Here we expect the same safety gap σ as length for the CS phase.

However, further savings could be achieved if node i only needs to receive some $\eta \in \{0, \ldots, n-1\}$ of firings of its neighbors per period T. This way, equation 3 adjusts to

$$\begin{aligned}\gamma_i^\eta &= \frac{T - \Delta t_{i,RF}^\eta}{T} \\ &= 1 - \frac{(\eta+1)\cdot(1+\varepsilon)\cdot|F(i,1)|}{n\cdot\big((1+\varepsilon)\cdot|F(i,1)| + (k-1)\cdot f_k\big)} \\ &= \frac{\left(\frac{n-\eta-1}{n}\right)\cdot(1+\varepsilon)\cdot|F(i,1)| + (k-1)\cdot f_k}{(1+\varepsilon)\cdot|F(i,1)| + (k-1)\cdot f_k}.\end{aligned} \tag{4}$$

With equal length for firing slots and data slots, equation 4 again reduces to

$$\gamma_i^\eta = \frac{k - 1 + (1+\varepsilon)\cdot\frac{n-\eta-1}{n}}{k+\varepsilon}.$$

If the radio transceiver is powered down for several periods, even more energy could be saved. But such a long down time implicates additional problems which can destabilize collision free communication and require extra administrative costs to keep track of the down times of nearby nodes. Hence, we won't go into detail here, but discuss in the next section the effect on network latency using some results from this section about energy-savings.

IV. Latency

So far, the period T mainly depends on n, the maximum number of supported nodes, and the slot lengths. But when examining the network latency, further parameters are of interest, like data rate or minimum packet length. Thus, if the firing packet contains additional information about the neighbors of the transmitting node, for instance to prevent the hidden node problem, the length of a firing slot $|F(i,1)|$ indeed depends on n. Introducing an adequate factor β with subject to network specific variables and leaving n fixed, the length of a firing slot can be specified as

$$|F(i,1)| = \beta \cdot n.$$

With it and according to equation 1, the minimal period T to support just firing packets (i.e. $k = 1$) plus safety gap for n nodes is

$$T = (1+\varepsilon)\cdot\beta\cdot n^2,$$

which is quite similar to equation 2.

Assuming that the length δ of the data section within a frame $F(i)$ is independent of n, the period T must hold

$$\begin{aligned}T &= \big((1+\varepsilon)\cdot\beta\cdot n + \delta\big)\cdot n \\ &= (1+\varepsilon)\cdot\beta\cdot n^2 + \delta\cdot n.\end{aligned}$$

But if the length of a firing slot is the disposing base unit as mentioned in Section III, where all slots have the same length $f_k = |F(i,1)|$, the length δ of the data section can be rephrased as a function of the number n of supported nodes

$$\delta = \delta_0 \cdot \beta \cdot n$$

by using another factor δ_0. Thus, the minimal period T now modifies to

$$\begin{aligned}T &= \big((1+\varepsilon)\cdot\beta\cdot n + \delta_0\cdot\beta\cdot n\big)\cdot n \\ &= (1+\varepsilon+\delta_0)\cdot\beta\cdot n^2.\end{aligned}$$

Overall, the number n of supported nodes has a much stronger influence on the length of period T, if n affects the length of all slots. That means, if the maximum number n of supported nodes increases, the period T grows with the square of n, the same is true for the network latency. Thus, a node has to wait in order of n^2 until its next firing, and so will a joining node, especially if they reside within an area of low density. For this reason, it seems not clever to make the slot lengths dependent on the number n.

That's why the trade-off between energy savings and network latency is quite complex – especially in networks of non-uniformly distributed nodes, containing areas of high density, causing a great value of n and – as a result – a great period T, and areas of low density, containing lots of unused slots.

V. Conclusion and Outlook

After a short motivation, we first introduced the biologically inspired and self-organized TDMA protocol DESYNC for Wireless Sensor Networks in Section II. Next, we analyzed its energetic characteristics in Section III, where we identified some adjustable parameters to save energy. In Section IV, we examined the latency performance of the DESYNC protocol with subject to the number n of supported nodes within a period. As a remarkable result, the length of the period is in order of square of n, if the slot lengths also depend on it.

For further research, we want to build a real-world testbed to specify some of our factors, like σ and δ. We also want to analyze the impact of an additional energy-harvesting unit at sensor nodes, which may influence the duty-cycle of the radio unit, too. As well, we try to promote a more universal version of the DESYNC protocol for multi-hop topologies using extra but locally available information. This additional information within a firing packet may be sufficient to support further add-ons, like time synchronization or routing.

References

[1] Injong Rhee, Ajit Warrier, Mahesh Aia, and Jeongki Min, "Z-MAC: a Hybrid MAC for Wireless Sensor Networks," in *SenSys*, Jason Redi, Hari Balakrishnan, and Feng Zhao, Eds. ACM, 2005, pp. 90–101.

[2] V. Rajendran, K. Obraczka, and J. Garcia-Luna Aceves, "Energy-Efficient, Collision-Free Medium Access Control for Wireless Sensor Networks," *Wireless Networks*, vol. 12, no. 1, pp. 63–78, Feb. 2006, tRAMA. [Online]. Available: http://dx.doi.org/10.1007/s11276-006-6151-z

[3] Marcel Baunach, "Speed, Reliability and Energy Efficiency of HashSlot Communication in WSN Based Localization Systems," in *EWSN*, ser. Lecture Notes in Computer Science, Roberto Verdone, Ed., vol. 4913. Springer, 2008, pp. 74–89.

[4] Julius Degesys, Ian Rose, Ankit Patel, and Radhika Nagpal, "DESYNC: Self-Organizing Desynchronization and TDMA on Wireless Sensor Networks," in *IPSN*, Tarek F. Abdelzaher, Leonidas J. Guibas, and Matt Welsh, Eds. Cambridge, MA, USA: ACM, 2007, pp. 11–20.

[5] Julius Degesys and Radhika Nagpal, "Towards Desynchronization of Multi-hop Topologies," in *SASO*. Venice, Italy: IEEE Computer Society, 2008, pp. 129–138.

[6] Renato E. Mirollo and Steven H. Strogatz, “Synchronization of Pulse-Coupled Biological Oscillators,” *SIAM Journal on Applied Mathematics*, vol. 50, no. 6, pp. 1645–1662, 1990.

[7] Alessandro Giusti, Amy L. Murphy, and Gian Pietro Picco, “Decentralized Scattering of Wake-up Times in Wireless Sensor Networks,” in *EWSN*, ser. Lecture Notes in Computer Science, Koen Langendoen and Thiemo Voigt, Eds., vol. 4373. Springer, 2007, pp. 245–260.

[8] L. F. W. van Hoesel and P. J. M. Havinga, “A Lightweight Medium Access Protocol (LMAC) for Wireless Sensor Networks: Reducing Preamble Transmissions and Transceiver State Switches,” in *1st International Workshop on Networked Sensing Systems (INSS)*. Tokio, Japan: Society of Instrument and Control Engineers (SICE), 2004, pp. 205–208.

[9] Gertjan P. Halkes and Koen Langendoen, “Crankshaft: An Energy-Efficient MAC-Protocol for Dense Wireless Sensor Networks,” in *EWSN*, ser. Lecture Notes in Computer Science, Koen Langendoen and Thiemo Voigt, Eds., vol. 4373. Springer, 2007, pp. 228–244.

[10] Koen Langendoen and Thiemo Voigt, Eds., *Wireless Sensor Networks, 4th European Conference, EWSN 2007, Delft, The Netherlands, January 29-31, 2007, Proceedings*, ser. Lecture Notes in Computer Science, vol. 4373. Springer, 2007.

Routing over Bursty Wireless Links

Muhammad Hamad Alizai, Olaf Landsiedel, Jó Ágila Bitsch Link, Stefan Goetz, Klaus Wehrle
Distributed Systems Group
RWTH Aachen University
Aachen, Germany
Email: {firstname.lastname}@rwth-aachen.de

***Abstract*—Accurate estimation of link quality is the key to enable efficient routing in wireless sensor networks. Current link estimators focus mainly on identifying long-term stable links for routing, leaving out a potentially large set of intermediate links offering significant routing progress. Fine-grained analysis of link qualities reveals that such intermediate links are *bursty*, i.e., stable in the short term.**

In this paper, we use short-term estimation of wireless links to accurately identify short-term stable periods of transmission on bursty links. Our approach allows a routing protocol to forward packets over bursty links if they offer better routing progress than long-term stable links. We integrate a Short Term Link Estimator and its associated routing strategy with a standard routing protocol for sensor networks. Our evaluation reveals an average of 22% reduction in the overall transmissions when routing over long-range bursty links. Our approach is not tied to any special routing protocol and integrates seamlessly with existing routing protocols and link estimators.

I. Introduction

Since the emergence of WSNs, research has mainly focused on link estimation and routing techniques [4], [5], [7] which identify and utilize consistently high quality links for packet forwarding. Links of intermediate quality, i.e. links with a PRR between 10% and 90%, are ignored to ensure routing stability and to attain high end-to-end reliability. Protocol studies [2], [8] have shown that these intermediate quality links are bursty, i.e., they frequently switch between stable and unstable periods of transmission for a limited number of consecutive packets. In this paper, we argue that: (1) Bursty links can be used for packet forwarding during their stable periods without affecting the reliability and stability of existing routing protocols; (2) These links often achieve significantly better routing progress and routing throughput than the long-term links chosen by existing routing protocols.

Today's link estimators [4], [5] measure the quality of a link in the ETX metric: the number of (re)transmissions required for a successful transmission. To achieve better connectivity and reliable packet communication, today's link estimators restrict communication to neighbors with constantly high-quality links. These high-quality links are identified based on the long-term success rate of a link collected over a time frame in the order of minutes. Widespread routing protocols in WSNs, such as BVR [5] and CTP [7], select links as suggested by their link estimator. In doing so, they limit packet forwarding only to long-term reliable links. They leave out a large class of potentially valuable communication links of intermediate quality that offer significant routing progress. Their use might therefore reduce the number of transmissions, lower energy usage in the network, and increase throughput.

Overall, this paper has three key contributions. First, it shows how short-term link estimation can be used for fine-grained estimation of bursty links to identify stable transmission periods. Thereby it enables routing protocols to forward packets over long-range bursty links and minimize the number of transmissions in the network. Second, we present an adaptive routing strategy which uses the STLE for packet forwarding over bursty links. Third, we present a Bursty Routing Protocol (BRP) - an integration of the STLE and the adaptive routing strategy with a standard routing protocol for sensor networks. As a result, we show how our approach can be integrated with existing routing protocols and link estimators. Our evaluation measures that routing with the STLE provides a reduction in transmission costs, i.e., number of transmissions, of 22% on average and 40% in the best case.

II. System Overview

Typically, routing protocols in WSNs aim to establish a routing tree: Some number of nodes in the network would advertise themselves as base stations, i.e., as tree roots. All other nodes join the tree with ETX as the routing metric. Figure 1 shows an example of such a routing tree rooted at the base station D. A path from source S to the destination D consists of a sub-sequence of immediate parents of each node, for example $S \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow D$. If we consider all links in this path to be 100% reliable, the minimum number of transmissions required by a packet to travel from the source to the destination is four. Now consider a situation in which an intermediate link $S \rightarrow 2$ or $1 \rightarrow D$ has become temporarily reliable. Routing over these links could result in a path sequence $S \rightarrow 2 \rightarrow 3 \rightarrow D$ or $S \rightarrow 1 \rightarrow D$, respectively. Hence, using these links for routing could reduce the total number of transmissions to three in the former and two in the later case. However, a traditional routing protocol cannot make use of such an opportunity because it uses a long-term link estimate. Hence, this design is intentionally unable to realize short-term changes in the link quality. Similarly, even if these short-term changes are captured, traditional routing schemes adapt slowly to ensure routing stability.

In contrast, our proposed technique takes advantage of the availability of intermediate links. It estimates links on a short-term basis by overhearing packets. In this particular case for example, node-2 overhears the packets addressed to

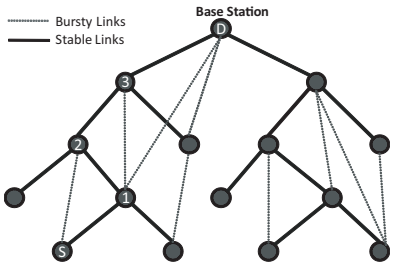


Fig. 1. Bursty links offer routing shortcuts that can reduce the number of overall transmissions in the network

node-1 by source S. After node-2 successfully overheard a certain number of consecutive packets from S, it informs S about the short-term availability of this link. Thereafter, S starts forwarding its packets to node-2 to reduce the number of overall transmissions for a packet to reach its ultimate destination. Our evaluation results in Section IV show that this technique significantly reduces the number of overall transmissions in the network. Thus, it allows to reduce energy consumption and increase network life time.

III. System Design

In this section we discuss the design of BRP, which consists of two basic components: (1) the STLE, which identifies periods of good transmissions in long-range bursty links by overhearing communication channels, (2) an adaptive routing strategy, which makes use of bursty links identified by STLE for forwarding packets, and its integration in existing routing protocols.

A. Short Term Link Estimation

The main task of STLE is to identify reliable periods of transmissions in intermediate links that offer better routing progress than long-term stable links. For this purpose, STLE overhears data packets send by neighboring nodes and records the recent history of success or failure over a link for the last h packets. Based on this recent transmission history, STLE decides whether a link is currently reliable or unreliable for transmission and informs the routing protocol accordingly. The prototype implementation of BRP consists of the STLE and the adaptive routing strategy with CTP, a standard collection routing protocol for sensor networks shipped with TinyOS. However, the STLE and its routing strategy are not bound to any specific routing protocol. It can easily be integrated with BVR and other routing strategies that support higher data rates for bandwidth limited systems.

1) Algorithm: Before elaborating the algorithmic details of STLE, we define three roles for any node in the network: a) *source-node*: the node which is actively sending or forwarding packets b) *parent*: the parent of any source-node in traditional routing and c) *overhearing-node*: node(s) which can overhear the communication between the source-node and its parent. A node in the network can assume any or all of these three roles at a time. The STLE algorithm works as follows:

Link Reliability: When overhearing a packet from a source-node, the overhearing-node infers the success rate - derived from the packet sequence number - of the link with that source-node. If the loss equals zero, i.e. if the overhearing-node was able to overhear a sufficient number of consecutive packets sent by the source-node to its parent, the overhearing-node declares the bursty link between itself and the source-node as active and triggers the next phase of the algorithm. If the overhearing-node was unable to overhear a sufficient number of consecutive packets (see Section III-A2) sent by the source-node to its parent - the overhearing-node drops the oldest packet sequence number for that source-node from its history and waits for the next packet.

Link Feasibility: In this phase, the overhearing-node queries the routing protocol for the path-ETX of the packet's destination, i.e., the parent of the source-node. If the path-ETX of the parent-node is greater than that of the overhearing-node, the overhearing-node declares the bursty link between itself and the source-node active. Consequently, the active bursty link can offer a better routing progress than the traditional path used by the source-node. However, if the path-ETX of the parent-node is not known or less than the path-ETX of overhearing-node, the overhearing-node temporarily ignores the source-node.

Link Announcement: If the path-ETX of the parent-node is greater than that of the overhearing-node, the overhearing-node informs the source-node about the active bursty link. It volunteers itself to become the temporary parent of the source-node as long as this bursty links remains active.

The path-ETX information used by the STLE at the overhearing-node can easily be obtained by using the neighborhood information maintained by any traditional routing protocol. We assume that there is a high probability that the original parent of the source-node is also a neighbor of the overhearing-node. This is because the overhearing-node can listen to the ongoing communication between the source-node and its parent. Additionally, the link announcement message, sent by the overhearing-node to the source-node, establishes a simple check to test for link-asymmetry.

Link Unavailability: At the source-node, the STLE declares a link unavailable for transmission after it fails to receive a number of acknowledgments (see Section III-A2) for the data packets sent over the bursty link.

2) History Size Thresholds: The STLE requires two thresholds for its operation: (1) a threshold to determine after how many successful transmissions, i.e. packets overheard, we define a link temporary available and (2) after selecting it for routing, threshold to define how many transmission failures we allow before considering a link temporary unavailable. Experiments by Becher et. al. [2] suggest a value of three, i.e. a history of size of three, for the first threshold and one for the second. However, these numbers were derived for a single testbed. As part of our evaluation, we repeat their experiments for widespread testbeds such as MoteLab [1] and TWIST [6] to calibrate STLE. Overall, our experimental results suggest the same thresholds as Becher et. al. Hence, we believe that

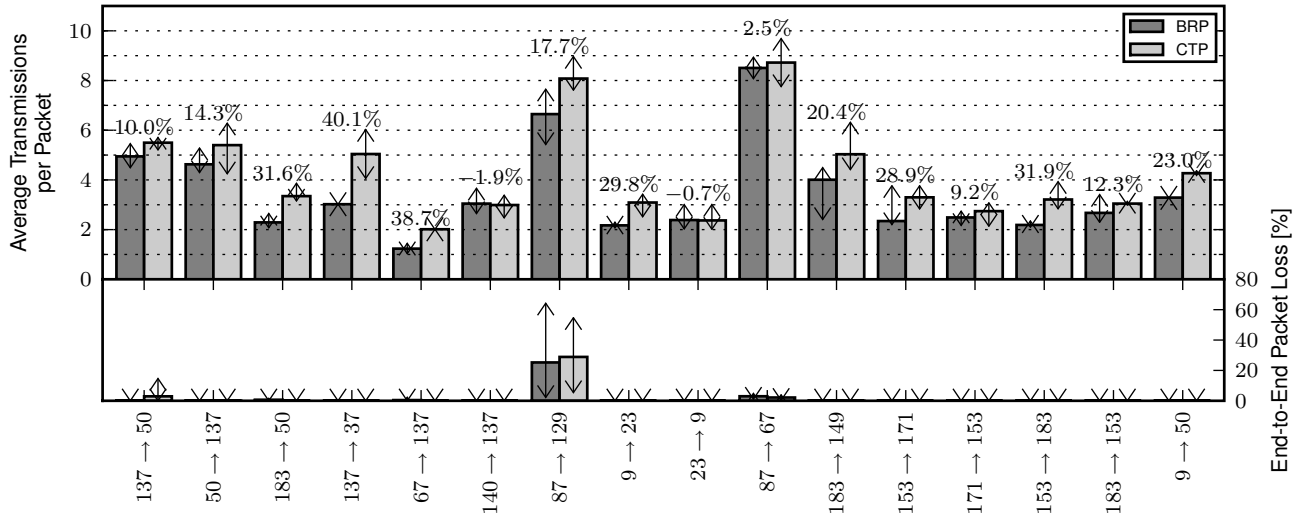


Fig. 2. Transmission cost reduction and reliability comparison of BRP and CTP. The graph above shows average number of transmissions per packet using BRP and traditional CTP for our experiments on MoteLab. The graph below shows end-to-end packet loss for the same experiments. The bar represents a node pair's average of five experiments. The inter-packet interval is 250 ms. The error bars represent the highest and the lowest average of the five experiments. The MoteLab topology and node addresses can be seen at www.motelab.eecs.harvard.edu.

these thresholds are valid in general and not only for a single deployment.

B. An Adaptive Routing Strategy

After discussing the operation of the STLE, we now detail a greedy and adaptive routing strategy based on it. Whenever the STLE at the source-node informs the routing strategy about an active bursty link, the routing strategy makes the overhearing-node its temporary parent and starts forwarding packets to it. However, this information is not propagated by the routing protocol to its descendant nodes, because these short term changes would trigger further parent changes down the tree. Eventually, it might destabilize the routing protocol and result in loops. This is one of the primary reasons why stability prevails over adaptability in today's routing protocols and link estimators. Hence, our routing strategy supplements their design considerations. When the STLE declares a bursty link inactive, the adaptive routing strategy proceeds as follows:

- It queries the STLE for another active bursty link. If such a link is available, the routing strategy starts forwarding packets over it.
- If there is no active bursty link, the adaptive routing strategy will regress to traditional routing until the STLE again finds an active bursty link.

Our goal is to enhance routing performance without affecting the stability and reliability of traditional routing protocols. Therefore, we neither replace the existing link estimators nor alter the stable routing topology maintained by traditional routing protocols. Rather, our approach is an additional component that assists routing protocols and link estimators in identifying the previously ignored class of bursty links which can enhance routing performance.

Test-bed	Transmission Reduction %		Throughput Increase %
	One Sender	Simultaneous Senders	
MoteLab	18.98	21.72	5.65
TWIST	16	19.33	10.43

TABLE I
SUMMARY OF THE PERFORMANCE RESULTS FOR MOTELAB AND TWIST

IV. EVALUATION

We perform our experiments on MoteLab and TWIST. Our major performance measure is a reduction in the number of transmissions in the network by enhancing routing progress. We compare the transmission cost of BRP with the original CTP. Figure 2 shows our results for 16 different node-pairs as senders and collection roots. We repeated our experiments for BRP and CTP five times for each of the 16 node-pairs to intensively validate our results. In most of the cases BRP performs better than CTP, averaging to approximately 22% overall reduction in the transmission costs i.e. the total number of transmissions from source to destination for single node-pairs.

Figure 2 also presents the end-to-end packet loss for our experiments. In most of the cases, the packet loss is negligible. From these results, it is fair to conclude that BRP does not affect the reliability of the underlying routing protocol and at the same time reduces the number of transmissions in the network. The only measurable end-to-end packet loss observed in our experiments is for the node-pair $87 \rightarrow 129$ and $87 \rightarrow 67$. However, Figure 2 shows that BRP performs better than CTP even in such lossy scenarios. Table I summarizes our evaluation results for MoteLab and TWIST.

Another property of bursty links that we investigate is timeliness: how often do they occur and for how long are

they active. Figure 3 presents empirical traces from our performance evaluation experiments. It clearly shows that bursty links are regularly available over time and are reliable for variable durations. Some of these links are active for only a few milliseconds (e.g $153 \rightarrow 183$), while others for seconds and even minutes (e.g $140 \rightarrow 37$). However, due to the slow adaptivity of traditional routing, i.e. CTP, even these relatively long-term reliable links with higher routing progress would not be utilized.

V. Related Work

The majority of existing link estimation techniques assume that individual packet loss events on a link are statistically independent of each other and that they follow a Bernoulli distribution. However, studies such as [2], [8] mark this assumption as inappropriate when wireless links are estimated over shorter time scales. For example, Becher et. al. [2] analyze the impact of recent transmission success and failure rate on the future quality of a link at fine-grain time scales. The conclusion of their study is that any link, no matter of what quality, becomes temporarily reliable after *h* consecutive packets are received over that link. Srinivasan et. al. [8] define a factor β, which measures the burstiness of a wireless link. β is calculated by using conditional probability distribution functions (CDFs), which determine the probability that the next packet will be received after *n* consecutive successes or failures. β is used to identify bursty links with long bursts of successes or failures and statistically independent links, with ideal bursty ($\beta = 1$) and independent ($\beta = 0$) links marking the two ends of spectrum.

Opportunistic Routing [3] in 802.11 based wireless networks reports a throughput increase of 35% by utilizing long range wireless links. However, it has a relatively high overhead with regard to computational cost, storage, and communication which we deem not feasible in resource constrained sensor networks. Overall, our short-term link estimator and its integration with routing protocols is designed according to lessons learned from afformentioned experimental studies on bursty wireless links. However, our work does not aim at modeling and developing the analytical or experimental understanding of wireless links. Instead, we take a step further and use these experimental models for packet forwarding over bursty links, and hence, enabling better utilization of wireless links.

VI. Discussion

In this paper, we presented a simple greedy approach to utilize bursty links of intermediate quality for packet forwarding. Our evaluation results show that, by transmitting over long range intermediate links, the number of transmissions in the network can be reduced. We believe that the improvement of 22% over traditional routing by transmitting over links with high loss rates is a credible and a realistic result.

In our prototype implementation, the development of a simple and a light-weight algorithm that illustrates the true effect of bursty links has been one of the primary objectives. After evaluating the effectiveness of transmissions over such links, we identify the following aspects as future work: 1) Employing a more perceptive approach for calibrating STLE in different network environments to successfully predict the short-term reliability of a link, 2) Classifying overhearing nodes based on their success history to avoid repeated selection of a node that did not offer significant improvement over the traditional path, 3) Limiting link selection to the ones that offer at least one hop reduction to avoid even the rare occurrence of bad results, 4) Integrating BRP with low-power listening techniques, 5) Extending this work towards 802.11 networks to show that our approach has a broader relevance in the wireless domain.

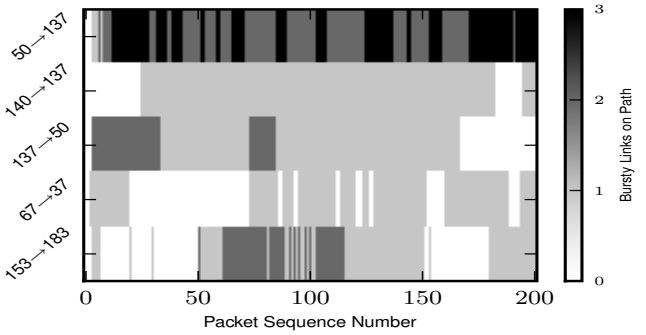


Fig. 3. Timeliness of bursty links for 50 second empirical traces for selected node-pairs: The graph shows the variability in the duration for which intermediate links are reliable. Most of the successful packets took one or more bursty links on the path from source to destination. Only the white segments in the graph represent complete packet transmissions on traditional path.

References

[1] G. W. Allen, P. Swieskowski, and M. Welsh. Motelab: a wireless sensor network testbed. In *IPSN '05: Proceedings of the 4th international symposium on Information processing in sensor networks*, 2005.

[2] A. Becher, O. Landsiedel, G. Kunz, and K. Wehrle. Towards short-term link quality estimation. In *Proc. of The Fith workshop on Embedded Networked Sensors (Emnets 2008)*, June 2008.

[3] S. Biswas and R. Morris. Exor: opportunistic multi-hop routing for wireless networks. *SIGCOMM Comput. Commun. Rev.*, 35(4):133–144, 2005.

[4] R. Fonseca, O. Gnawali, K. Jamieson, and P. Levis. Four-bit wireless link estimation. In *Proc. of the Sixth Workshop on Hot Topics in Networks (HotNets)*, November 2007.

[5] R. Fonseca, S. Ratnasamy, J. Zhao, C. T. Ee, D. E. Culler, S. Shenker, and I. Stoica. Beacon vector routing: Scalable point-to-point routing in wireless sensornets. In *2nd Symposium on Networked Systems Design and Implementation (NSDI)*, May 2005.

[6] V. Handziski, A. Köpke, A. Willig, and A. Wolisz. Twist: a scalable and reconfigurable testbed for wireless indoor experiments with sensor networks. In *REALMAN '06: Proceedings of the 2nd international workshop on Multi-hop ad hoc networks: from theory to reality*, pages 63–70, New York, NY, USA, 2006. ACM.

[7] F. Rodrigo, O. Gnawali, K. Jamieson, S. Kim, P. Levis, and A. Wo. The collection tree protocol. In *TinyOS Enhancement Proposal, TEP 123*, August 2006.

[8] K. Srinivasan, M. A. Kazandjieva, S. Agarwal, and P. Levis. The β-factor: measuring wireless link burstiness. In *SenSys '08: Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 29–42, New York, NY, USA, 2008. ACM.

A Roadmap for Hardware and Software Support for Developing Energy-Efficient Sensor Networks

Christoph Weyer, Christian Renner, and Volker Turau
Hamburg University of Technology
Institute of Telematics
Schwarzenbergstraße 95
D-21073 Hamburg, Germany
{c.weyer,christian.renner,turau}@tu-harburg.de

Hannes Frey
University of Paderborn
Computer Networks Group
Pohlweg 47-49
D-33098 Paderborn, Germany
hannes.frey@uni-paderborn.de

***Abstract*—Support for developing energy-efficient applications for wireless sensor networks is still scarce. In this paper a roadmap of a combined hardware and software approach is presented. The main idea is to collect state information and trace energy consumption of an application running in a testbed of real sensor nodes.**

I. Introduction

Wireless sensor networks consist of small, micro controller driven nodes with additional sensing capabilities. Once deployed in a certain environment the network must run unattended for a long period of time. In such scenarios energy consumption is the most important system parameter, even if energy harvesting is possible. A lot of research has been conducted in this context. However, the behavior of a new protocol is often evaluated by using simulations only. Producing an executable for real sensor networks still requires a lot of additional effort. Therefore, development support for energy-efficient sensor networks is necessary.

Especially for energy-constraint scenarios it is very important to develop and evaluate the application as soon as possible on real hardware with the additional sensor technology. Running such tests on real hardware with dozens of nodes is not feasible without any additional support. The following tasks must be automated: programming of nodes, monitoring the behavior of the network during the test run, and collecting debugging information. The runtime behavior of a node is described by the energy it consumes and the program parts that are executed over time. This information is important for debugging and evaluation purposes.

In this paper a roadmap of our combined hardware and software approach is presented. The software part will be used for an automated instrumentation of existing applications, such that it provides logging information about the runtime behavior. The hardware part is supposed to measure rate and cumulative current consumption of code execution per node and provides connectivity to manage the node directly from a central management station. We will outline first design decisions on both the hardware and software part in the following sections. After that we will report the first experiences, which we gained from preliminary prototypes.

II. Software Support

In order to get meaningful information about the state of the running application, additional logging data must be provided. In [1] we proposed TinyAID, an automated instrumentation tool that supports two kinds of automated code instrumentation: call-chain logging and message logging. Figure 1 depicts the tool chain of automated code instrumentation. Given any nesC source code, the TinyOS tool chain first creates a single, plain C file by combining this code with the TinyOS components used. The automated code instrumentation intercepts the TinyOS tool chain after this point, adding an additional preprocessing step. Given a certain configuration file the instrumenter inserts additional instrumentation code, provided by a code template, into the plain C file. This step results in an instrumented C file that will then be handed back to the remaining TinyOS tool chain. Depending on the target platform, an instrumented program image or a TOSSIM library is created.

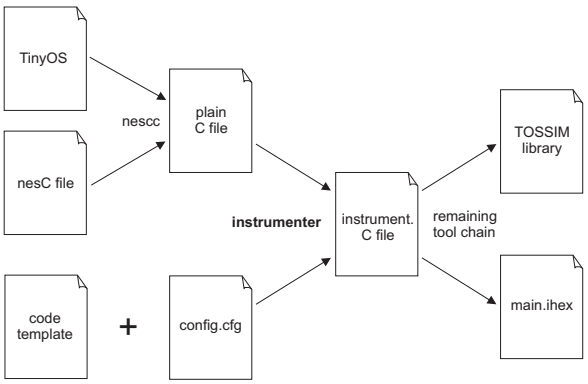


Fig. 1. Automated instrumentation by intercepting the TinyOS tool chain.

A. Call-Chain Logging

Call-chain logging is used for logging the enter and exit times of certain event handlers and functions. This is achieved via additional code that is added to these handlers and functions during the instrumentation pass. For every handler and

function, logging code is added immediately after the function entry point, at the end of the function, and immediately before each return. Since call-chain logging may result in very large data sets due to the names of event handlers and functions, the logged data only consists of unique integers. During the instrumentation pass, a separate file is created, which maps the names of event handlers and functions to a unique integer value.

The event or function to be logged is defined by the configuration file used during compile time. Every line in this file starts with either '+' or '−' to include or respectively exclude event handlers or functions that match the following expression. The inclusion or exclusion symbol is followed by `d`, `f`, or `h` to decide whether the following regular expression is applied on directory names, file names, or handler and function names. The instrumenter steps through the plain C file and checks for every encountered function or event handler entry point, if they match any of the expressions of the configuration file. For this, the list of expressions is scanned from top to bottom, until the first match is found. Depending on the inclusion and exclusion flag, this line decides, if code instrumentation is applied or not. If no entry is found, code instrumentation is not applied.

B. Message Logging

TinyAID also supports logging messages that have been created, sent, or received by the node. This is achieved by the automatic instrumentation of the appropriate TinyOS functions. Basically, additional logging code is added immediately after the entry points of the functions `AMSend.send`, `Receive.receive`, and `Packet.clear`. In order to follow the flow of a message in the network, the message header is extended by an unique message identifier. It consists of the address of the node having created the message (the message's origin) and a sequence number. Each message is tagged with this information at creation time. Here we utilize the TinyOS coding convention that for any newly created message the `Packet.clear` function has to be called.

Since message tagging is completely transparent to the application developer, any application can be monitored with this mechanism. On the event of receiving or sending a packet logging information about the current packet can be provided by the node. The level of detail of the logged data can be easily configured by applying different code templates to the instrumenter.

C. Manual Instrumentation

There are two main situations, in which manual code instrumentation may become unavoidable. These include identifying the visited states of certain state machine implementations, and secondly identifying the end points of communication protocols.

For the first aspect, a function `state(name)` is introduced. It can be added manually at any code line. The instrumenter will create a mapping from state names to automatically generated state identifiers. Again, as with call-chain logging, the additional mapping is used to keep the logged data compact. Code execution passing such function will produce additional logging data. For identifying correct delivery of messages the function `consume(msg)` is introduced, which has to be added at those code places where the message successfully reaches its semantically correct destination.

III. Hardware Support

The hardware adapter is supposed to host a single sensor node and to connect it to an Ethernet network. As sketched in Fig. 2, the adapter is built around a small micro controller with additional peripheral building blocks for adjusting available current, measuring actual and cumulative energy consumption, retrieving state information of current code execution on the sensor node, and communicating logging information towards and control information from a server controlling the experiment. The Atmel NGW100 is used for fast prototyping of the needed hardware components. The final version will be a self-developed, printed circuit board containing all necessary components. The hardware adapter and the actual sensor node are powered via power over Ethernet (PoE).

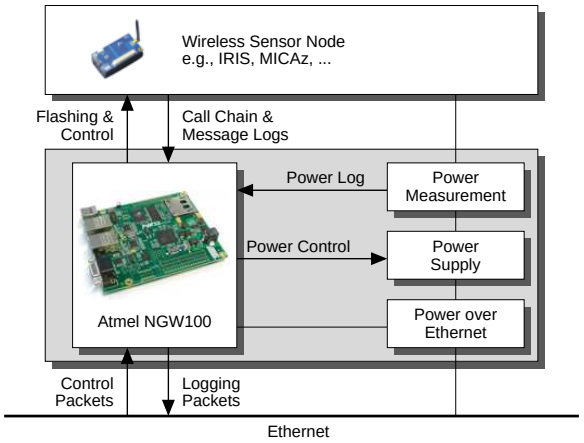


Fig. 2. An overview of the hardware adapter's building blocks.

A. Measuring and Controlling Current

The hardware adapter should enable fine-grained periodic sampling of the current drawn by a given sensor node running a certain application. The measurement should be precise enough to make energy consumption observable on the level of packet reception or function calls. An additional requirement is that our hardware adapter should be able to measure current consumption over several orders of magnitude. More precisely, a sensor node has a current consumption of a few μA, when the sensor node is running in a deep power saving mode, while the node will consume up to about 100 mA, when it is under high load with some additional active sensors. This poses the challenge that the hardware adapter should be able to measure current drawn from 10^{-6} to 10^{-1} A, i.e., five orders of magnitude.

In order to support the development of energy-aware protocols that react on available energy per node, the hardware platform should also be able to control the voltage available at the sensor node. Again, we assume 100 mA as an upper bound of current consumption.

B. Retrieving Log Data

For getting logging information out of a node without affecting its functional behavior in the network, it is important to log as less as necessary while using a most unobtrusive way to communicate such data to the outside world. We consider the usage of several I/O pins as one possibility for this communication between the sensor node and the hardware adapter.

Call-chain logging is performed by sending a single Byte via the I/O pins. Bit 8 encodes if the function is entered or left, while the remaining bits encode the function ID out of at most 127 possible ones, zero means that no data is available. Message logging requires additional information to be transmitted via the I/O pins. This includes one Byte each for the message ID, the receiver node, the message originator, and the sequence number.

For both call-chain and message logging, any additional information is determined on the hardware adapter. This includes the sensor node ID, the current time, the amount of energy consumed since the last call-chain event, and the rate of current energy consumption.

Determining the actual time requires additional effort, since we want to relate energy measurements and events on different nodes in our evaluations. We consider three possible solutions. First, the obvious way is the application of a time synchronization protocol (e.g., NTP) over the Ethernet connection. Second, depending on the deployment, all hardware adapters may be synchronized with a global clock signal over a shared control line. Third, no synchronization may be used at all. In this case, ordering of events on different nodes can be based on the fact that a receive event always happens after its corresponding send event. In other words, the latter technique may be useful if considering causal ordering of events is sufficient for the empirical study.

C. Communicating Data

Ethernet is used in order to exchange data between the hardware adapters and a central management station controlling the sensor network experiment. Information from the server to the hardware adapters includes new images to be flashed and configuration data for controlling the experiments. One example is configuration data for controlling the energy available to the sensor node. On the reverse way any generated logging information and energy consumption measurements are immediately packed into an Ethernet frame and transmitted to the management station. At this computer the information is collected and further processed.

For communicating data from the sensor node to the hardware adapter general I/O pins are utilized as described. Moreover, a single I/O pin is used for communicating between the hardware adapter and the sensor node. This flag can be used for conditionally generated log information, i.e., only if set to true, information will be logged at the sensor node and transmitted to the hardware adapter. In a future version additional ports are available at the hardware adapter for simulating sensor hardware. This is necessary when evaluating the behavior of an application that depends on specific sensor readings or events. Otherwise an automated test is not possible.

IV. Experience Report

A. Software Support

The concept of the automated instrumentation is evaluated in detail in [1]. In all cases TOSSIM is used for simulating the instrumented code. Thereby, the process of logging data is simplified by the fact that the information can be logged directly into files. The configuration of the instrumenter for TOSSIM is as follows. We have provided the modules responsible for creating, sending, and receiving packets with code templates producing trace information. An example code snippet for tracing packet reception is shown in Listing 1.

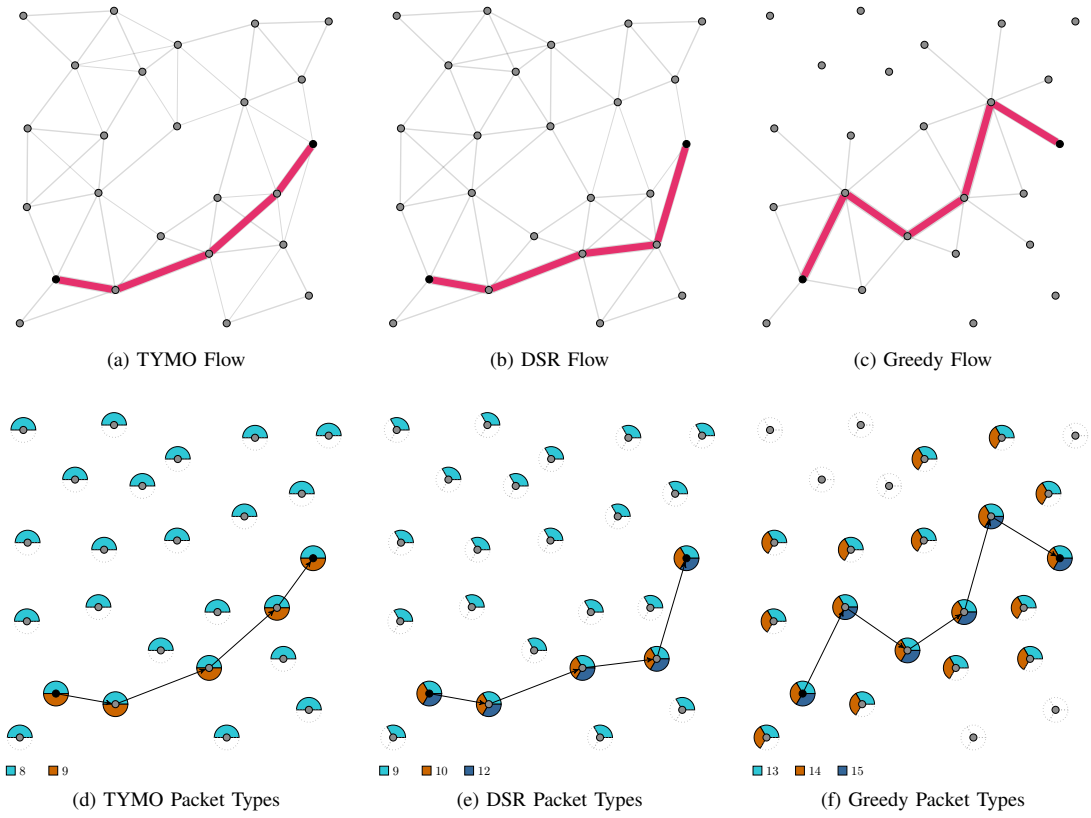
```
tossim_header_t  * header = getHeader(msg);
dbg_clear("TINYAID_PACKET_TRACING",
  "%d␣%lld␣R␣%d␣%d␣%d␣%d␣%d\n",
  sim_node(), (sim_time_t)(sim_time() * 1e-7 + 0.5),
  header->type, header->src, header->dest,
  header->origin, header->seqno);
```

Listing 1. Code template for packet reception

In order to demonstrate the useability of the automated instrumentation, the packet tracing is demonstrated by comparing three different routing protocols. The first one is TYMO, an implementation of the well-known DYMO protocol, which is included in the TinyOS 2.x code base. TYMO uses internal message types for route requests and route replies. These are sent in order to query and establish a new route, if a forwarding node does not know where to send a given message. The second routing protocol is Dynamic Source Routing (DSR), which follows a similar concept. The third protocol considered is Greedy Routing. Messages are forwarded using the geographic positions of forwarding nodes and the destination. The greedy aspect is realized by considering only neighbor nodes closer to the destination and sending the message to the neighbor closest to the destination. The output of the automated packet tracing is shown in Fig. 3.

B. Hardware Adapter

We have been experimenting with different hardware approaches for adjusting the sensor nodes available power supply and for measuring a sensor node's current consumption [2]. After evaluating all considered approaches it turned out that they are not feasible for the objectives, which we are aiming on with our planned hardware adapter. For both parts, in the following we briefly sketch the different approaches and our observations with that solution.



(a) TYMO Flow (b) DSR Flow (c) Greedy Flow

(d) TYMO Packet Types (e) DSR Packet Types (f) Greedy Packet Types

Fig. 3. Visualization of packet flows and packet type distribution

For adjusting output voltage levels an LM317 voltage regulator from National Semiconductor, which supports up to 1.5 A output current, is utilized. The voltage regulator can be adjusted by an analogous input appropriately. A digital adjustable voltage regulator is constructed by a resistor network, which is then controlled by the output lines of a binary register. As an alternative solution a DAC8831 digital analog converter from Texas Instruments is evaluated, which produces an analog voltage level according to a serial digital input. Both circuits are tested on how good they reproduce a sinus and a square wave with sampling rates between 50 Hz to 20 kHz. The results show that only the DAC8831 reproduce an acceptable shape. Unfortunately, the DAC8831 does not allow drawing enough current to support attached sensor nodes running under full load.

For measuring the current draw of the sensor node we considered different instrumentation amplifiers coupled with a shunt resistor, which is then used as an input into the logarithmic amplifier LOG104. The instrumentation amplifiers under consideration included the INA138 and the INA333. The first one was tested alone and in conjunction with an additional INA138 and a REF200 for amplifying the signal. The latter was as well tested alone and in conjunction with a cascaded amplifier INA138. In all investigated settings we were not able to measure current consumption precisely with 1 kHz over 5 orders of magnitude.

V. Conclusion

The development of energy-efficient applications requires supporting tools, especially when dealing with real hardware. We proposed a hardware and software based solution that will help during the development and first test deployments. The advantages of an automated instrumentation support over manual instrumentation are presented by a simple but effective example of message tracing. The results of a first prototype of the proposed hardware adapter have shown that it is difficult to measure the current consumption over 5 orders of magnitude at a frequency of 1 kHz. The next steps will be to solve this open issue in order to gain first experiences by using TinyAID in combination with such a hardware adapter.

References

[1] C. Weyer, C. Renner, V. Turau, and H. Frey, "TinyAID: Automated Instrumentation and Evaluation Support for TinyOS," in *Proceedings of the Second International Workshop on Sensor Network Engineering (IWSNE'09)*, Marina del Rey, CA, USA, 2009.

[2] H. Baumgart, "Entwurf eines Hardwareadapters zur Strommessung von drahtlosen Sensorknoten," Project Work, Hamburg University of Technology, Apr. 2009.

SANDbed: A WSAN Testbed for Network Management and Energy Monitoring

Anton Hergenröder, Jens Horneber, and Joachim Wilke
Institute of Telematics
Universität Karlsruhe (TH)
Germany
{hergen, horneber, wilke}@tm.uka.de

***Abstract*—In this paper we present our work-in-progress testbed SANDbed for wireless sensor actuator networks. In contrast to many existing testbed approaches for WSANs we concentrate our research on highly quantized and resolutive distributed energy monitoring in WSANs as well as on flexible management protocols for this kind of testbeds. Therefore the management software is being designed to support any sensor node platform and also any testbed topology.**

I. Introduction

The research cooperation *ZeuS* [1] is a project investigating the basic principles of reliable communication in wireless sensor actuator networks (WSAN). We believe that it is impossible to provide *full* reliability (in terms of authenticity, robustness, latency, etc.). Instead, a *trade-off* must be made, which balances energy consumption against different reliability measurements. If only little energy is available, reliability should be degraded in favour of network lifetime, which leads to the concept of *probabilistic* reliability. As an example, a query-result being authentic with 70% probability may me better than no authenticity at all [2].

In the ZeuS context, the Institute of Telematics is currently developing a testbed to evaluate energy-aware protocols in ways as yet impossible with existing solutions. Our testbed, called *SANDbed* (**S**ensor **A**ctuator **N**etwork **D**evelopment test**bed**), is not only an appliance, but an integral part of the project. It allows us to evaluate protocols and schemes on real hardware, getting real energy measurements to confirm the often untrustworthy simulative evaluations. Nevertheless, the conceptual design allows the testbed to be of use in many other WSAN scenarios beyond ZeuS.

Therefore, the contribution of this paper is twofold. First, we would like to present SANDbed to the community, to establish relations to research beyond the ZeuS cooperation. Second, we would like to point out the challenges implicated by the various requirements we impose. This is of use for other projects that may need evaluation in a WSAN testbed, too.

In the following Sections, we sketch SANDbed, its main features, challenges and current status. For this, we first outline the primary design goals, introducing core features. Then, we draw the architecture of the testbed itself, but also of software components used for management and monitoring. We conclude by specifying the project's current status and future work.

II. Goals

We identified three major goals for SANDbed development. The Testbed should provide side effect free monitoring of motes without any effects caused by the testbed environment itself. This is to obtain results as precise as possible from the examined sensor network. The second goal for SANDbed is to support multiple experimenters with deployment and management of their experiments and results. For experiment evaluation, we want to be able to get significant and detailed information about the used amount of energy on individual motes measured at distributed locations. In the following we describe each goal and resulting design decisions in more detail.

A. Side effect free monitoring

Monitoring of operating wireless sensor networks can be done on the sensor nodes or by dedicated additional hardware. The first approach is quite easy to realize on a single mote by adding software monitoring modules to the application to be analyzed. Especially in complex field research environments, additional hardware cannot be attached due to inaccessible motes, expenditures or integration problems like wiring dedicated monitoring hardware in the field. However a software approach also suffers from side effects with the running application on the monitored mote. Many sensor network specific operating systems' inability to run multiple threads simultaneously leads to imprecise monitoring results and additional energy consumption for monitoring overhead.

To avoid these drawbacks, our approach uses *dedicated monitoring boards* for energy measurement attached to each sensor mote in SANDbed.

B. Deployment and experiment management

As a useful tool for analysis and evaluation of WSAN protocols and algorithms a vital requirement for a testbed is the management of *user-defined experiment* sets. Each experiment set contains binary image files for sensor motes, configurations for scheduled measurements, mote locations of individual applications and tasks. Additionally, an experiment set can contain mote input for experiment runtime. Using experiment sets allows to repeat an experiment easily with slightly modified initial values and random seeds as well as multiuser support by the testbed. We use a database to store

experiment sets and results like application data, distributed energy measurement values and serial mote output for debugging purposes.

SANDbed allows to manage all motes by a web-based graphical user interface which provides functionality, e.g. running experiment applications and monitoring the status of individual motes. The intended purpose of the GUI is twofold. On one hand it provides a network management interface for the testbed and the embedded sensor network. Network management becomes necessary especially because we want to support and maintain heterogeneous mote types in a single sensor network and extensions to our management and monitoring hardware. Using a common network management protocol in SANDbed even enables us to integrate SANDbed in an Internet environment. On the other hand the GUI can also be used to setup, deploy and evaluate new protocols and applications. Mote programming can be done via interface from everywhere without physical access to the motes hardware making SANDbed a worthwhile tool for fast development and deployment of new ideas and sensor network prototypes.

C. Distributed energy measurement

The third goal for SANDbed development is to optimize *energy efficiency* of resource constrained wireless sensor actuator networks in order to increase network lifetime. In particular, the distributed nature of wireless sensor networks leads to difficulties in determining the impact of network-wide effects of communication algorithms and architectures. Motes can suffer increased energy demands even if they are not directly involved in a communication process between two nodes. MAC-layer characteristics like observing radio channels for incoming messages or free time slots to send own messages to active nodes are examples of such energy demanding behavior. To pinpoint those effects, we have to measure the used energy not just as sum over the complete run of an experiment, but in very short time intervals of milli- and microseconds.

We are planning to expand SANDbed with the capability of measuring energy consumption on mobile motes for advanced applications like event tracking. Protocol behavior may even depend on different kinds of energy supply or take remaining energy into account. SANDbed is therefore able to simulate various battery models and energy supplies specified by experimenters.

III. Architecture

A. Hardware architecture

The hardware architecture of SANDbed is organized in a tree-like structure shown in Figure 1, enabling high scalability of the testbed.

The root level of the tree comprises the user interface and the database. The database is used for storing all information of SANDbed that requires persistence. SANDbed stores its configuration, user data, experiment configuration and results in this database, where the data is provided to the user interface or the testbed itself. The second tree level is formed by the management nodes connected to the Internet. Each

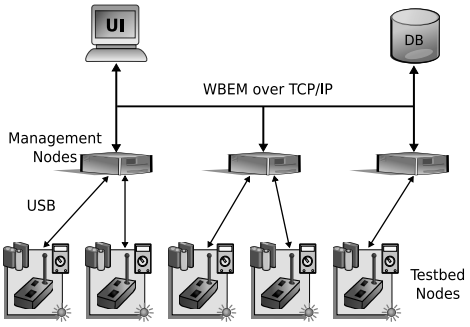


Fig. 1. SANDbed architecture

management node is responsible for monitoring a couple of motes connected over the USB interface. The leaves of the tree are the testbed nodes, consisting of a mote (e.g., MicaZ [3]) and the Sensor Node Management Device (SNMD). Figure 2 shows a possible configuration of a testbed node.

This architecture allows side effect free monitoring in a controlled testbed environment resulting in more precise measurements of energy usage without impact on mote hardware resources. Therefore we designed two orthogonal, non-interfering communication infrastructures in SANDbed. The *horizontal* wireless in-sensornetwork communication is controlled by the researched application while we *vertically* use a TCP/IP and wired USB infrastructure for management and monitoring communication in SANDbed.

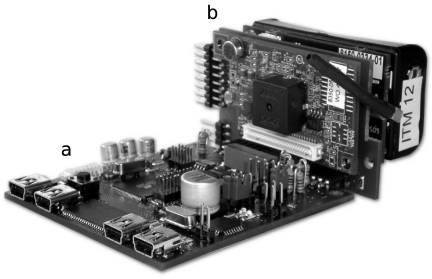


Fig. 2. (a) Sensor Node Management Device (SNMD) with (b) adapted MicaZ and Sensorboard forming one possible configuration for a testbed node

Sensor Node Management Device: SNMD is developed with the intention to analyze the energy consumption behavior of the motes. We achieve this by completely controlling and monitoring the energy supply of the managed sensor nodes. SNMD is able to provide power supply to the motes from an attached battery or by simulating a battery. The battery simulation is done by using USB as power source and controlling the voltage supplied to the sensor notes. SNMD is capable of high resolutive measuring of the voltage and current consumed by the nodes. The measurement results can be provided live (each sample individually) or buffered and sent in chunks over USB.

Furthermore, SNMD provides a comprehensive extension interface which can be used for attaching add-ons like displays,

SD-cards, etc. Storing the logs and measurement results on SD-cards enables long term buffered measurements and standalone operation. In addition, the extension interface can be used to simulate different environmental conditions for the attached mote. This can be achieved by connecting the SNMD output ports to the input of the mote, thus directly feeding it with simulated sensing data.

Due to the power supply being the only really required interface between SNMD and the mote, every sensor node platform with compatible electrical characteristics can be adopted for operation within SANDbed. Primary benefit of this fact is again side free monitoring, because there is no need of adapting existing WSAN-applications. In fact, the mote has no knowledge of being monitored in a testbed at all.

Table I summarizes the most characteristic technical specifications of the SNMD. Currently the SNMD is in preproduction state being tested as prototype.

TABLE I
TECHNICAL SPECIFICATIONS OF SNMD

Measurement sampling rate:	
unbuffered	$max\ 20kHz$
buffered	$max\ 400kHz$
Sampling resolution	$16\ bit$
Measurement accuracy	$< 2\%$
Sample buffer	$896kB \mathrel{\widehat{=}} 448.000\ samples$
Current measurement range	1). $0 - 100mA$ 2). $0 - 200mA$ 3). $0 - 500mA$
Voltage measurement range	$0 - 10V$
Charging Capabilities:	1). NiMH - $maxI = 200mA$ 2). Li-Ion - $maxI = 200mA$
Node power supply type	1). Real battery 2). Simulated battery using USB-Power
Power output:	
$maxU_{out}$	$4,2V$
$maxI_{out}$	$1A$
Programmer capabilities	Any Atmel AVR Chip
PC-Connection	Over USB-Hub
User interface	Serial-USB command line
Extension Interface	I^2C SPI 16 bit I/O Subsystem SD-Card

B. Software architecture

For managing and monitoring objectives we favor standards over proprietary solutions. Therefore the management software of SANDbed is designed as a client-server architecture based on the Web-Based Enterprise Management (WBEM) [4]. WBEM is a set of open standards defined by Distributed Management Task Force (DMTF). One of these is the Common Information Model (CIM) [5] which we use for designing the SANDbed information model. The management information is exchanged between clients and servers over the CIM-XML protocol [6] that uses XML over HTTP to exchange CIM information.

The management nodes implement the server side of WBEM. Therefore every client with WBEM abilities is able to manage our testbed. We prefer WBEM over other network management protocols like SNMP [7], because of its object orientation. This empowers us to easily implement device hierarchies and remote method invocations. The latter are indispensable for controlling the experiment runs and mote behavior.

IV. FUTURE WORK

SANDbed is still in development state. While the hardware is almost in production state, the management software is our next stage of development. The preliminary examinations of the SNMD are very impressive, so we are going to start deployment of SANDbed in the near future. The first SANDbed deployment site will be at the Universität Karlsruhe (TH). There, we will gain experience in managing and monitoring sensor networks and especially distributed energy measurement. In future we will extend SANDbed by further sites, so that more interested scientists will be able to participate in enhancement of SANDbed and performing experiments in wireless sensor actuator networks.

V. CONCLUSION

In this paper, we presented our WSAN testbed SANDbed. We pointed out the major goals, namely *side effect free monitoring*, easy *deployment and management* of WSAN experiments, and last but not least *distributed energy measurement*. We sketched the testbed architecture and showed its current status. Several issues, we identified, have not yet really been addressed to in other WSAN research. However, for a satisfying development and realistic evaluation of WSAN protocols, these issues *must* be solved. Here, the SANDbed will be a step into the right direction.

ACKNOWLEDGMENT

The authors wish to thank Detlev Meier for his essential work on the design and assembly of SNMD, Denise Dudek for her helpful advices and also Jochen Dinger, Moritz Killat and Patrick Armbruster for their contribution to the developement.

REFERENCES

[1] Research cooperation at University of Mannheim and Universität Karlsruhe (TH), "ZeuS – Zuverlässige Informationsbereitstellung in energiebewussten ubiquitären Systemen." [Online]. Available: http://zeus-bw-fit.de/

[2] J. Wilke, E.-O. Blaß, F. C. Freiling, and M. Zitterbart, "A Framework for Probabilistic, Authentic Aggregation in Wireless Sensor Networks," *Praxis der Informationsverarbeitung und Kommunikation (PIK)*, vol. 2, pp. 116–126, Apr. 2009.

[3] "MicaZ sensor node," http://www.xbow.com/.

[4] "WBEM Web-Based Enterprise Management by DMTF Distributed Management Task Force, Inc." http://www.dmtf.org/standards/wbem/.

[5] "CIM Common Information Model." http://www.dmtf.org/standards/cim/.

[6] "CIM-XML protocol." http://www.dmtf.org/standards/wbem/CIM-XML/.

[7] J. Case, M. Fedor, M. Schoffstall, and J. Davin, "Simple Network Management Protocol (SNMP)," RFC 1157 (Historic), Internet Engineering Task Force, May 1990. [Online]. Available: http://www.ietf.org/rfc/rfc1157.txt

Extended Abstract: Programming Wireless Sensor Networks using UML2 Activity Diagrams

Christoph Damm[1] and Gerhard Fuchs[2]

Department of Computer Science 7, University of Erlangen-Nuremberg
Martensstr. 3, 91058 Erlangen, Germany
christoph.damm@googlemail.com
gerhard.fuchs@informatik.uni-erlangen.de

***Summary*—Wireless Sensor Networks (WSNs) consist of a large amount sensor notes (spots). We address the problem of how theses spots can be controlled, so that they collaborate to fulfill a common task. UML2 Activity Diagrams (UADs) enable the user to model workflows, describing activities, in a graphical, structured and hierarchical manner. The novelty of our work is that we program WSNs using UADs. Our approach not only covers workflow description, but also action allocation. We develop a framework to design activities describing the behavior of WSNs and to execute those diagrams by spots, after an automated transformation of the model file. As a result of our first experiments, testing this framework using 12 spots, we can say that UADs can be used to program in a platform-centric as well as in an application-centric way. Additionally, we are able to adapt the behavior of a spot during runtime by reloading activities. Further research is necessary to see the full spectrum of drawbacks and benefits of our attempt.**

I. Introduction

WSNs [2] have become an important branch of research. Teething problems like routing, clustering and energy awareness in WSNs have been widely discussed and there is a trend towards discussing how to use this new technology for real applications. For us the research challange in the field of WSNs lies in the huge amount of spots (hundreds, thousands, ...), which must be coordinated. Many spots should interact and fulfill a common task. How can a programming model cope with these distributed operations?

In this extended abstract we introduce our framework that allows to program a WSN using a subset of UADs. We describe related work, basics of UADs, and our preliminary work in section II. We present important aspects of our framework in section III, section IV describes an example experiment using it. Finally section V concludes this exteded abstract, gives an brief outlook to further work, and puts some open questions.

II. Related Work, Basics and Previous Work

A. Programming WSNs

Sugihara and Gupta have written a detailed survey about programming models for WSNs [3]. They have introduced a taxonomy, distinguishing between an application-centric view and a platform-centric view that a programming model can take. Similar to us, they see collaboration as one important requirement for WSN applications and so for a programming model. Guerrero et al. have written a position paper [4], discussing some theoretical aspects in the field of workflow support for WSNs. To our knowledge a concrete implementation is not available. Unlike to our proposal they describe the workflows using state charts, similar to us they see the possibility to bring the programming closer to domain experts.

B. UML2 Activity Diagrams

The following subsection is based on Oestereichs book [5] that summarizes the official specification of UML2 [6], [7]. UADs describe a workflow. An activity (diagram) is defined by different kinds of nodes (action nodes, object nodes and control nodes) that are connected by object flows and control flows, symbolized by arrows. A control flow transports so called tokens, a object flow objects.

C. eXMIcutionUnit-Plugin for ROBRAIN

We gained our first experiences to programm systems using UADs during the Masters Thesis of Ipek [8]. He realized a prototype for Linux with C++ as a plugin (eXMIcutionUnit) of a multi robot programming framework called ROBRAIN [9]. As there are much more resource constraints for spots compared to the robots, there clearly was the need to create a similar (in the sense of describing workflows with UADs), but more featured and specialized framework for spots.

III. Our Framework

A. Avtivity-Centric View on WSNs

We aim our framework at programmers that want to code with the following view on WSNs: A spot can execute activities. If it executes an activity, it has the control about it. Its scope is limited to the workflow that is described by this activity. An action, which is included in an activity, can be allocated to a spot. The spot, executing the activity, has the control over this allocation process.

B. Components

Our framework consists of a tool for programming the UADs (`IDE`), an execution environment for UADs that runs on the spots (`CORE`), a transformation rule (`RULE`), and an access software to the WSN for the user (`ACCESS`). We use Papyrus UML 1.11.0 [10] as IDE, the rest is realized by us.

[1] implemented the fundament of our framework for his Master's Theses [1]
[2] corresponding author

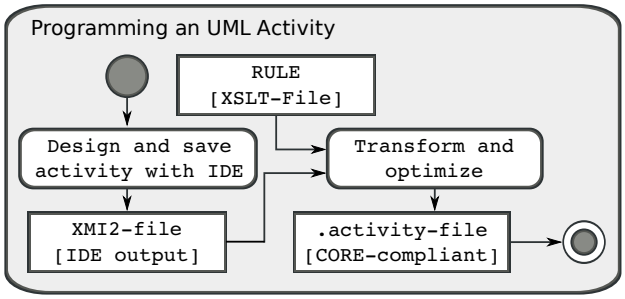


Fig. 1. Programming an UML Activity. RULE is offered by our framework.

C. Used Tools, Libraries and Technologies

CORE is realized for Sun SPOTs [11], which are programmed using Java ME [12]. We use kXml2.2.2 [13] as XML (Extensible Markup Language [14]) parser. For RULE we use XSLT (Extensible Stylesheet Language Transformation [15]). xsltproc [16] is used to convert the Papyrus UML output, which is XMI2.1 (XML Metadata Interchange [17]), to a RDF (Resource Description Framework [18]) compliant file. ACCESS is written in Java for a PC with a connected base station.

D. Features of the Current Implementation

- A programmer programs UADs by using IDE. The output of IDE is converted into a CORE-compatible syntax, using RULE.
- CORE can be pre-configured with activities, which will be loaded and parsed when CORE is started. At runtime additional activities can be added via ACCESS.
- The execution of an activity can be started by the user via ACCESS, or by the CORE of another spot.
- Status information of a spot (currently its supported activities and the battery load status) can be retrieved from CORE via ACCESS.
- CORE may execute several activities or file parsing operations simultaneously.
- A basic scope of UAD elements is supported: start nodes, final nodes, fork nodes, synchronisation nodes, decision nodes, merge nodes, guards, control flow and object flows. Furthermore implicit forks and synchronisation as well as a hierarchical structuring of activities is possible. The UADs were expanded with two stereotypes: $\ll allocated \gg$ (to mark and describe the allocation of actions) and $\ll root \gg$ (to mark the call of Root Activities).
- Programmers may program new RootActivities against our JAVA interface and integrate it into CORE.
- Programmers may program new AllocationProcesses against our JAVA interface and integrate it into CORE.

E. Programming UML Activities

The goal of the programming process is to gain a CORE-compliant `.activity` - file that can be interpreted by CORE (Fig. 1). First of all the programmer has to design and save the activity with IDE in an `XMI2-file`. The IDE output is transformed and optimized using RULE.

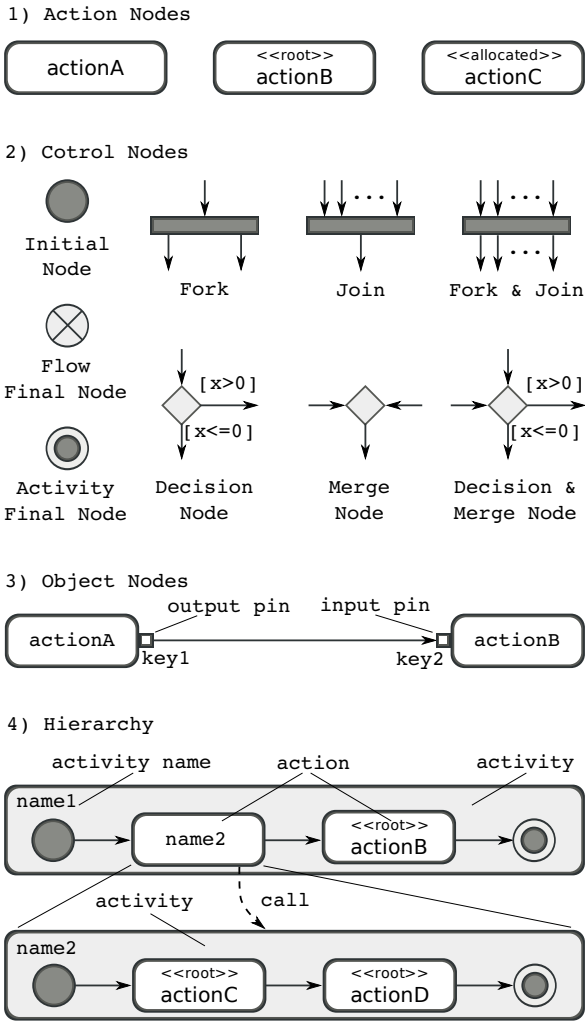


Fig. 2. Important aspects of UML2 Activity Diagrams.

F. Supported UAD elements

A programmer, who works with our framework, can currently use the elements shown in Fig. 2 to program an activity. The choosen syntax, semantik and description of the UAD elements is based on [5]–[7].

1) Action Nodes: An `action` symbolizes one step in an activity. In UML a stereotype can be used to add further information to an element. In our framework an action with $\ll root \gg$ - stereotype symbolizes that the corresponding activity is realized in Java and not programmed using UAD. The $\ll allocated \gg$ - stereotype shows, that the action is delegated to a spot. Our framework allows the combination of these two stereotypes.

2) Control Nodes: An `initial node` is at the start of the workflow of an activity. More than one initial nodes are possible in an activity. CORE looks for all initial notes and

starts for each one a thread for the execution. A `flow final node` is at the end of a single flow. CORE stops the execution of it, the other flows are **not** stopped. An `activity final node` indicates the end of an activity. CORE stops **all** flows in the activity.

A `fork node` allows parallelism in activities. One incoming flow is immediately split in several outgoing. CORE starts for each flow a thread for the execution. A `join node` reduces parallelism and allows synchronization in activities. CORE waits for **all** incoming flows before the outgoing is started. Is is a conjunction with **and**-semantic. A `fork & join node` is a combination of a fork and a join node. CORE waits for **all** incoming flows before it starts **all** outgoing flows.

A `decision node` must have a single flow entering it, and one or more flows leaving it. At the outgoing flows conditions are annotated that specify which flow must be chosen. They are called guards and must allow a unique decision. CORE has an *GuardProcessor* that parses the annotations and allow to compare Strings, Integers and Doubles using the operators $=$, $!=$, $<$, $>$, $<=$ and $>=$. The parameter `x` must be set by an action of the activity. A `merge node` has one ore more incoming flows. CORE waits for **one** incoming flow, before the outgoing is started. This is a conjunction with **or**-semantic. A `decision & merge node` is a combination of a decision and a merge node. CORE waits for **one** incoming flow, before it uses the *GuardProcessor* to take the decision.

3) Object Nodes: An `object node` indicates that an object or a set of objects exist. CORE uses them as an incoming or outgoing parameter. IDE allows to symbolized the object node as a pin (with a square at the border of an action node). CORE uses *HashTables* for the mapping between keys and values. Per convention in our framework, the names of the input and output pins (here *key1* and *key2*) must fit to the keys in the Hashtable.

4) Hierarchy: An activity consists of single actions. If CORE detects an action it calls the activity that has the same name (here *name2*). If an action is tagged with the $\ll root \gg$ - stereotype, CORE knows, that it must call the corresponding Java-Class (here *actionA, actionB, actionC*. As we want to concentrate on the programming using UAD, we differ from the official specification, which says that a `CallBehaviourAction` indicates the call of an activity.

G. Action Allocation

We have chosen the following syntax for an instruction that can be added with the $\ll allocated \gg$ - stereotype:

$$instruction := (method : parameter : set) \rightarrow set$$

`method` specifies an AllocationProcess (entry in a list). `param` allows the modeler to specify parameters which are necessary for the allocation process. `set` is a comma-separated list of spots, from which the allocation which the allocation process must select the target set of spots. It is possible to substitute `set` with an additional instruction, so more complex allocations can be processed recursively.

IV. Example Experiment

A. Experimental Setup

For this experiment we have build an example WSN, which consists of 12 spots. We have programmed a NetTemp and a SpotTemp activity (Fig. 3) and initially deployed SpotTemp to all spots. Afterwards we switched on the power, reset the spots, and waited about one minute[1]. We transferred NetTemp to spot a11 over the air, using the base station, started the execution and have observed the behaviour and the final state of the spots. We have made three experiments (series 1), waited about half an hour, and made additional two experiments (series 2).

B. SpotTemp and NetTemp

NetTemp and SpotTemp describe the following behaviour: The sensor network has to determine a mean value of a temperature, decide whether the result is grater or lower than 30°C, and to indicate it. SpotTemp is executed on a spot. GetTemp detects the current temperature using the sensor of the spot. The result is passed to an output-pin and if it is grater than 30°C all LEDs of the spot become red, otherwise green. NetTemp runs on spot a11 and allocates the execution of SpotTemp to four spots ($\ll allocated \gg$ - stereotype). The results are asynchronously passed to MeanValue, that is allocated to spot a09. Spot a09 start the execution not till then it has all 4 results (implicit join). If the result, calculated by MeanValue, is grater than 30°C all LEDs of spot a10 become blue, otherwise those of spot a11.

C. Observed Behavior

We have repeated the experiment 5 times. After different combinations of four spots, executing SpotTemp, turned on their (red/green) LEDs, spot a10 or a11 turned on its blue LEDs. Four times all four spots turned on their red LEDs, one time all four spots turned on their green LEDs. The green LEDs turned on at the first run of series2.

D. Interpretation and Results

As we haven't done an exact and independent measurement of the temperature, a quantitative conclusion is not possible. Qualitatively we can say, that our spots have behaved as expected. We have seen two different behaviors of the network. Both the network and the spots have made a decision and indicated it. We are able to program our WSN in a platform-centric and an application-centric way.

V. Conclusion and Further Work

The huge amount of spots that must be handled in large scale WSNs is a fascinating challenge. Spots should collaborate to fulfill a common task. Based on our experiences in the field of multi robot programming, we are currently investigating how UADs can be used for programming WSNs. UADs can be used to describe workflows in a graphical, structured and hierarchical manner. Actions nodes, control notes, object

[1]For our experiment we used the standard communication protocol stack of the spots. This means, that we had to wait, until the spots had initialized the network.

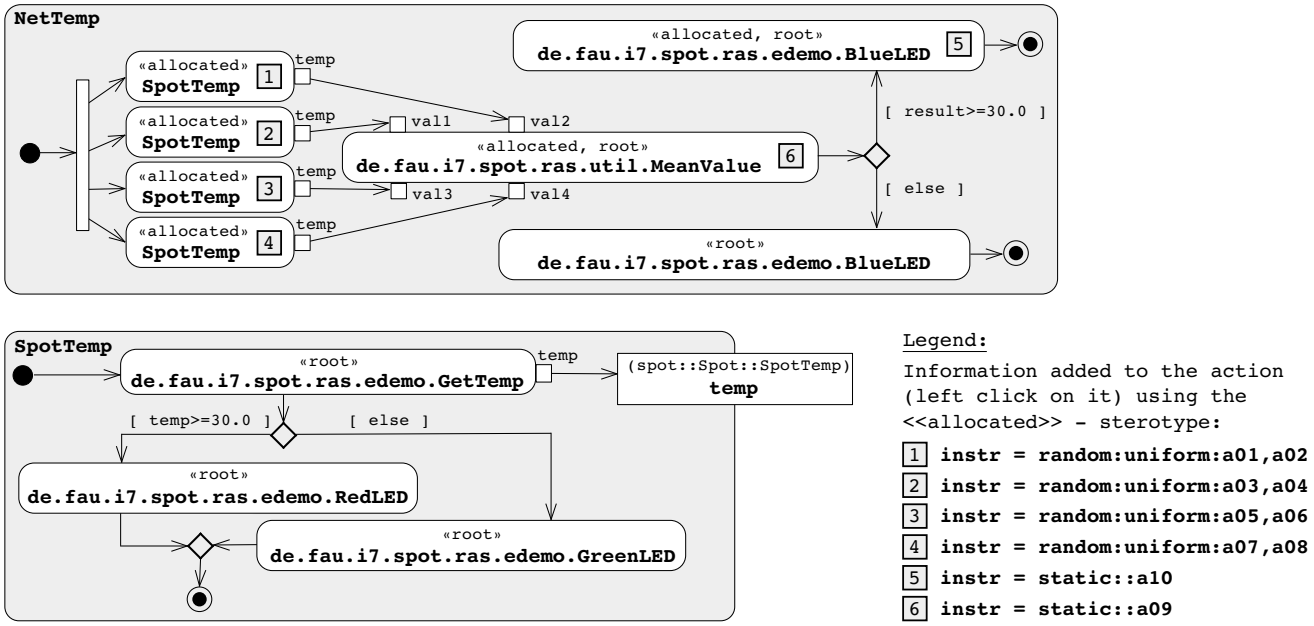


Fig. 3. Example for the programming of WSNs using UADs. NetTemp and SpotTemp are two different UADs, programmed with Papyrus UML. The images of them are the svg-export of Papyrus UML (with some typographical modifications). The diagrams are composed of the elements offered by Papyrus UML. With the exception of the boxes (1-6) the content of the diagrams correspond to the Papyrus UML view. NetTemp calls SpotTemp. NetTemp is an application-centric activity, SpotTemp a platform-centric activity.

nodes, hierarchy, parameters and stereotypes are a subset of important aspects of UADs. We use the expressiveness of them for our framework that give the user an activity centric point of view on a WSN.

Our framework uses Papyrus UML as an IDE for the design of activities, describing the behaviour of WSNs. We offer RULE for the transformation of these activities in a RDF-compliant file that can be executed by our CORE, running on the spot. Additionally we provide ACCESS to supervise, control and reprogram the spots.

First experiments, using 12 spots, show us, that our attempt can be used to program in a platform-centric as well as in an application-centric way. Additionally to workflow description, our framework currently supports static and random action allocation, and the extension of the repository of a spot during runtime, for network reprogramming.

We are currently investigating more sophisticated action allocation mechanisms and are about to increase the number of spots of our WSN. Additionally we are building robots that are controlled by the spots to gain a robot sensor network. To see all consequences of this work we have to ask amongst others: What features should / can be integrated in this framework? What is a good method and scenario for the evaluation of our framework?

REFERENCES

[1] C. Damm, "Implementierung und Bewertung eines RDF-basierten Frameworks zur Interpretierung und Ausführung von UML2-Aktivitätsdiagrammen auf Sensorknoten," Diplomarbeit, University of Erlangen-Nuremberg, Sep. 2008.

[2] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Elsevier Computer Networks*, vol. 38, pp. 393–422, 2002.

[3] R. Sugihara and R. K. Gupta, "Programming models for sensor networks: A survey," *ACM Trans. Sen. Netw.*, vol. 4, no. 2, pp. 1–29, 2008.

[4] P. Guerrero, D. Jacobi, and A. Buchmann, "Workflow Support for Wireless Sensor and Actor Networks," in *Proc. of the 4th International Workshop on Data Management for Sensor Networks*, ser. AICPS, vol. 273. ACM, 2007, (DMSN: Vienna, AT-09; Sep. 2007).

[5] B. Oestereich, *Die UML 2.0 Kurzreferenz für die Praxis*, 4th ed. Munich, DE-BY: Oldenbourg Wissenschaftsverlag GmbH, 2005.

[6] OMG Object Management Group, "OMG Unified Modeling Language (OMG UML), Infrastructure, V2.1.2," OMG Available Specification without Change Bars formal/2007-11-04, 2007.

[7] OMG Object Management Group, "OMG Unified Modeling Language (OMG UML), Superstructure, V2.1.2," OMG Available Specification without Change Bars formal/2007-11-02, 2007.

[8] M. Ipek, "Aufgabenbeschreibung für mobile Roboterschwärme," Diplomarbeit, University of Erlangen-Nuremberg, May 2006.

[9] Department of Computer Science 7; University of Erlangen-Nuremberg, "ROBRAIN," http://robrain.berlios.de/ [web: 2009/02/09].

[10] S. Gérard, "Papyrus UML," http://www.papyrusuml.org/ [web: 2009/05/18].

[11] Sun Microsystems, "Sun SPOT," http://www.sunspotworld.com/ [web: 2009/02/09].

[12] Sun Microsystems, "Java ME," http://java.sun.com/javame/ [web: 2009/05/18].

[13] S. Haustein, "kXML," http://kxml.sourceforge.net/ [web: 2009/02/09].

[14] W3C World Wide Web Consortium, "Extensible Markup Language (XML) 1.1 (Second Edition)," W3C Recommendation REC-xml11-20060816, Aug. 2006.

[15] W3C World Wide Web Consortium, "XSL Transformations (XSLT) Version 2.0," W3C Recommendation REC-xslt20-20070123, Jan. 2007.

[16] D. Veillard, "The xsltproc tool," http://www.xmlsoft.org/XSLT/xsltproc2.html [web: 2009/05/27].

[17] OMG Object Management Group, "MOF 2.0/XMI Mapping, Version 2.1.1," OMG Available Specification without Change Bars formal/2007-12-01, Dec. 2007.

[18] W3C World Wide Web Consortium, "RDF Primer," W3C Recommendation REC-rdf-primer-20040210, Feb. 2004.

Reliable Model Checking for WSNs

Christian Appold
Chair of Computer Science V
University of Würzburg
Am Hubland, 97074 Würzburg
Email: appold@informatik.uni-wuerzburg.de

***Abstract*—The capabilities of wireless sensor networks are promising a great future for them. Nevertheless it's necessary to have methods to verify their correct operation before deployment, for widening their range of application and enable their usage in e.g. safety critical environments. One method to examine if systems behave as desired is temporal logic model checking [1], which is a formal verification technique. When verifying wireless sensor networks, some special aspects like the correct modeling of the wireless communication and possibly other components of the sensor nodes are essential. In this paper we report about the verification of a traffic light synchronization protocol at 4-way intersections, where the traffic lights communicate wireless. We present some needful abstraction techniques and discuss particularities in the verification of wireless sensor networks.**

I. Introduction

In this paper we report about the verification of a traffic light synchronization protocol at 4-way intersections and the verification of wireless sensor networks (WSNs) in general. WSNs can consist of a large number of sensor nodes. Because verification of distributed systems is hard, and even a single sensor node and its software could be very complex, verification of sensor networks is a highly non-trivial task. But if they should become deployable in e.g. safety critical environments or areas where they can't be reprogrammed, it's unavoidable to verify that they fulfill their requirements. Otherwise implementation failures can be very costly and cause accidents, which in the worst case could lead to the death of humans. Common approaches to verify the functionality of WSNs are e.g. the use of simulators like TOSSIM [2] or live testing by using testbeds. As a drawback of these methods, they don't verify the desired properties for all possible computations of the sensor network. It is known that especially hard bugs in distributed systems often appear only in a few corner cases. Therefore such complex bugs cannot be detected reliably by these methods. An approach for early stage sensor network verification is the use of model checking. Because model checking isn't easy to apply correctly without some verification experience, it currently isn't widely used in the area of WSNs. Hence we outline in our paper guidelines and abstractions for improving the verifiability of WSNs. This should help to achieve fast and correct verification and make formal verification amenable for the WSN domain.

The paper is organized as follows. In Section II we discuss related work on the field of verification. Section III gives a short introduction in model checking and the symbolic model checker NuSMV [3], which we used for our verification experiments. The verified traffic light synchronization protocol is described in Section IV and Section V shows pitfalls when modeling wireless communication. In Section VI we present useful techniques to model the protocol in the input language of NuSMV. The paper closes with concluding remarks and an outlook to further investigations.

II. Related Work

WSNs often contain stochastic elements (e.g. backoff procedures of communication protocols or elements of the environment). To allow reliable verification of them, these have to be modeled as accurate as possible. PRISM [4] is a probabilistic model checker for analyzing quantitative properties of systems which exhibit stochastic behavior. But though the possibility to model probabilistic elements accurately, its input language is very restricted and probabilistic models are typically more complex, which decreases the limit what can be analyzed. Therefore we have chosen for our work the symbolic model checker NuSMV, which doesn't directly support the specification of probabilistic elements.

In [5] the authors verified the IEEE 802.3 Ethernet CSMA/CD protocol using the symbolic model checker SMV [6]. This protocol is a wired protocol, so they hadn't to deal with the special characteristics of wireless communication. Fehnker et. al [7] verified the LMAC protocol, a medium access control protocol for WSNs using Uppaal [8], a model checker for timed automata. Their property of main interest was detecting and resolving collisions, which they verified for different topologies. They showed that the truth of properties may depend on the network topology. The focus of both papers mentioned above was to verify a communication protocol, whereas our work aims towards verifying networks of sensor nodes considering not only communication, but also implemented functionality. We show that it's often necessary to model communication or possibly other important system components to verify functionality.

III. Model Checking and NuSMV

Model checking is an automatic formal verification technique for verifying properties of finite state systems. A model checker is a tool which, given as input a model of a system and a property of interest formulated in a temporal logic, automatically decides whether the property is valid for all possible computations of the model. To decide if a property is valid, the model checker has to explore all possible system

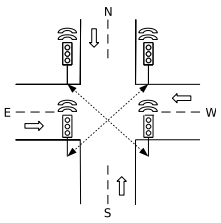


Fig. 1. 4-way road intersection with traffic lights

states exhaustively. As a consequence, the main problem of model checking is the state explosion problem. This problem especially appears in the verification of distributed systems, where the number of possible system states grows exponentially in the number of components. One method to reduce the state explosion problem is to use symbolic model checking [9], which uses BDDs for representing sets of states and the transition relation symbolically, instead of representing them explicitly.

Because symbolic model checking allows the handling of systems with very large state spaces, we used the symbolic model checker NuSMV [3] for our work. NuSMV is a reimplementation and extension of the symbolic model checker SMV. NuSMV permits the description of synchronous and asynchronous systems and has its own input language. For property formulation, NuSMV supports the temporal logics LTL and CTL [1], which extend propositional logic with temporal operators.

IV. The Traffic Light Synchronization Protocol

To show particularities of WSN verification and the usefulness of our abstractions, we developed a simple traffic light synchronization protocol for 4-way intersections. Figure 1 shows a 4-way road intersection with one traffic light for each incoming road. The purpose of the protocol is to synchronize traffic lights which communicate wireless. Thereby one of the main targets is to ensure that only diagonally arranged traffic lights are allowed to show green at the same time, to prevent accidents. A simplified state diagram of the control flow of the protocol for a single traffic light can be seen in Figure 2. The conditions for the feasibility of transitions and also transitions without state changes have been omitted for clarity. These conditions consist of combinations of values of local state variables and types of incoming messages from other traffic lights.

In the control states *red* (initial state), *yellow* and *green*, the protocol triggers its lights to get the corresponding color. If a traffic light is in the state *red* or *green* and gets a message to change its light color (e.g. from a traffic measurement sensor at the road), the protocol sends a command to transmit a light change request to the underlying MAC protocol. When the MAC protocol confirms the sending of the message, the transmitting traffic light changes its control state to an acknowledgement receiving state. If no communication errors occured, the diagonally arranged traffic light at the

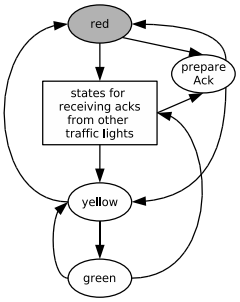


Fig. 2. Control flow state diagram of the traffic light synchronization protocol for a single traffic light

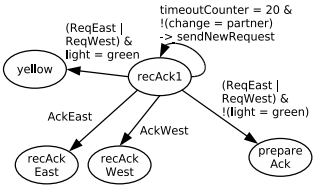


Fig. 3. Control state *recAck1* with transitions

intersection subsequently changes its control state also to an acknowledgement receive state. There are four such states in the protocol, but for clarity they are summarized in Figure 2 into one state. The other two traffic lights change their state to *prepareAck*, if their lights are red at the moment. When their lights show green, they first change their light color to red and then go to this state. In the state *prepareAck* they prepare and send an acknowledgement transmit request to the MAC layer. They leave this state, when they get the confirmation that the request has been sent.

Figure 3 shows exemplary transitions and next states of the receiving state *recAck1*. A traffic light changes to this control state from the states *red* and *green*, when it receives the confirmation that a light change request has been sent from the MAC layer. The Figure is drawn with identifiers for the traffic lights at the incoming roads from north and south, whose behavior is symmetric in this state. If the traffic light receives an acknowledgement from the traffic light at east or west and there hadn't been a communication error, it changes its state to *recAckEast*, and *recAckWest* respectively. Because it could be possible, that two light change requests from traffic lights collide, or a traffic light hasn't received a send request from another traffic light, this had to be considered in the protocol. Therefore the protocol uses a timeout counter, which is incremented every step when the traffic light is in state *recAck1* until the timeout limit is reached. Also the protocol uses a variable *change*, which has the value *partner*, when the traffic light received a change request from the diagonally arranged traffic light at the intersection.

By executing the transition with the variables *timeoutCounter* and *change* in Figure 3, a command to transmit a traffic light change request is send to the MAC layer. The state changes from *recAck1* to *yellow* or *prepareAck* are needed,

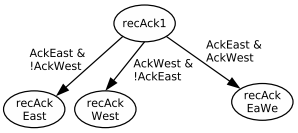


Fig. 4. Receiving of acknowledgements in *recAck1* without collisions

because a traffic light could be in the state *recAck1* while a change request from another traffic light could have been sent. This behavior could appear because of collisions or transmission errors, when sending a light change request. If the traffic light shows green, it first changes its state to *yellow* and then *red*, before going to state *prepareAck*. Otherwise it changes its state to *prepareAck* to initiate the transmission of an acknowledgement. Two different types of acknowledgements can be sent by a traffic light. One type to acknowledge a request of the diagonally arranged traffic light and another type to acknowledge traffic light change requests from the other two traffic lights. After reception of all necessary acknowledgements without timeout, the initiating traffic light sends a sendComplete message to inform the other traffic lights about successful light change and changes its light color accordingly.

V. Communication Modeling Pitfalls

In this section we show how neglecting wireless communication characteristics can circumvent the detection of design errors through model checking.

A. Nonobservance of collisions and impossibility to listen during sending

Here we show how disregarding collisions together with nonobservance of impossibility to listen during sending can prohibit the detection of design errors. Figure 4 shows an example model of the outgoing transitions of state *recAck1* where acknowledgements are received for the traffic light at north, without considering collisions. In a real world deployment acknowledgements from the traffic lights at east and west cannot arrive at the same time, because there would be a collision in wireless communication. When verifying the property that no deadlock exists for this model of the protocol, it could be verified by the model checker as correct even without using a timeout and request resend mechanism in the acknowledgement receiving states. If the acknowledgements from the traffic lights at east and west collide, without a timeout and request resend mechanism all traffic lights are stuck in their states. When ignoring that listening in wireless communication usually isn't possible during sending, the other parts of the protocol could be implemented for the model checker in a way, that these deadlock doesn't appear on any computation path of the model. As a consequence the model checker can't find the deadlock.

B. Nonobservance of variations in radio wave propagation

Variations in radio wave propagation, e.g. through changing environmental conditions or obstacles, can cause situations where a message sent by a traffic light could be received only by a subset of the desired receivers. In an early version of the protocol we used only one type of acknowledgements. During verification runs considering collisions but without variations of radio wave propagation, we couldn't find a counterexample for the property that only diagonally arranged traffic lights are allowed to be green at the same time. When we inserted variations of radio wave propagation in our verification model, we could find computation paths where three traffic lights could show green at the same time.

This behavior could appear, if all traffic lights showed red and the south traffic light changed its control state from *recAck1* to *prepareAck*, because of a light change request from the west traffic light. The light at north didn't receive this request. Subsequently the light at north did send a light change request and the south traffic light approved this by sending an acknowledgement, whereas its state change to *prepareAck* has been caused by a change request from the west traffic light. As a consequence the south traffic light switched to control state red and the north traffic light to state green. After that the north traffic light changed its state to *red* and then to *prepareAck*, because the traffic light at east transmitted a light change request to green. Then the traffic light at south, which didn't receive this request, transmitted a light change request which was received by north. In the old protocol the traffic light from north was able to change its variable *change* (see Section IV and Figure 3) to the value *partner* in state *prepareAck*. Therewith it could execute the transition from state *prepareAck* to state *yellow*. Subsequently the west traffic light send a request to change its lights color to green and the north traffic light acknowledged this and also changed its color to green.

VI. Modeling suggestions for reliable verification

In this section we describe suggestions to model the characteristics of wireless communication for the model checker NuSMV. We present suitable abstractions for modeling variations in the radio range, transmission errors, the possibility of packet collisions and the circumstance that collisions normally cannot be detected by the sending nodes. Their use allows the reliable verification of WSNs.

Verification models for NuSMV consist of several processes, which can be executed completely synchronous or completely asynchronous. For our verification runs we chose synchronous execution, because of the lower complexity of the verification model and the lower verification effort for the model checker. To model the wireless communication channel, we used DEFINE statements from the input language of NuSMV. These work like macros. We therewith specified for each traffic light defines for channel free, collision and one for each message of any other traffic light. The value of these defines is determined by the current control states of the traffic lights and the values of input variables. For this purpose we inserted the control states *sendReq*, *sendAck*, *sendAckPartner* and *sendComplete* in our verification model.

```
DEFINE
  free := !(sendReqEast | sendAckEast | sendAckPEast | sendCompleteEast | ...);
  collision := ((sendReqEast | sendAckEast | sendAckPEast | sendCompleteEast) &
                ((sendReqWest | sendAckWest | sendAckPWest | sendCompleteWest) | ...));
  sendReqEast := (east.cState = sendReq) & inputEast;
  sendCompleteEast := (east.cState = sendComplete) & inputEast;
```

Fig. 5. Example DEFINE commands for channel modeling

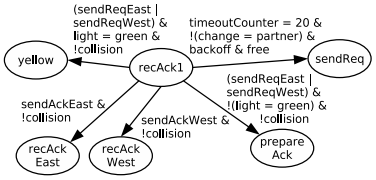


Fig. 6. Control state *recAck1* with transitions and abstractions

A traffic light in our model transmits, if its control state currently equals one of these new states. Figure 5 shows examples of DEFINE commands for channel free, collision occured and some incoming messages for the traffic light from north. A big advantage of this modeling, beneath its correctness and easy implementation is, that no new state variables are needed for it, because defines work like macros. In contrast the channel model of [7] introduces new state variables, which in large networks can affect the verifiability. The domain of the defines we formulated is Boolean. For *free*, the define holds the logical value true if no other traffic light at the intersection currently sends a message which north receives, otherwise false. To detect collisions, the define *collision* takes the value true if two or three other traffic lights send simultaneously messages which north receives. The last two defines in Figure 5 indicate if the traffic light for the incoming road from east sends a sendRequest or sendComplete message, which north receives correctly. Their logical value depends on the current control state of the traffic light at east and the value of the Boolean input variable *inputEast*. Input variables in NuSMV get their value from the verification environment through the model checker, which assigns all possible values to them in every state. They are used in the defines *sendReqEast* and *sendCompleteEast* for modeling variations in radio range and transmission errors.

Figure 6 shows the outgoing transitions and successor states of state *recAck1*, as in Figure 3, with our modelings and abstractions for wireless communication. To include collisions in our model, we added the condition *!collision* to transitions where messages have to be received for their feasibility. With our experiments we intended to verify the protocol for wireless communication and a MAC protocol with carrier sense and a randomized backoff procedure. Thus, for reliable verification we needed a suitable model for it, which preserves all possible computations and keeps the state space small. We developed an abstraction using Boolean input variables. Through adding conditions about a certain value of an input variable, we restricted the feasibility of transitions which lead to a state which models the sending of a message. In the transition from state *recAck1* to *sendReq* in Figure 6 this is the input variable *backoff*. Additionally we added the condition that the define *free* of the communication channel model also has to be true for feasibility of the transition. In the transition *free* is used to model the carrier sense mechanism and the input variable is responsible for modeling all possible behaviors of the backoff procedure.

VII. Conclusion and Outlook

In this paper we reported about verification of WSNs. We developed a traffic light synchronization protocol for 4-way intersections and showed some particularities in verifying WSNs. One conclusion is, that often system components, like e.g. synchronization protocols, cannot be verified isolated in WSNs. Frequently, a model of the communication protocol and models of other sensor node components, like e.g. timers or even parts of operating systems, are also necessary. A big challenge is to find suitable models which don't affect the verifiability (by leading to the state explosion problem) but describe the intended behavior correctly. Therefore especially for non-verification experts, suitable and faultless abstraction techniques should be available. In this paper we presented a way to model a communication channel with collisions, transmission errors and variations of radio wave propagation for the model checker NuSMV. Additionally we presented a model of a backoff procedure.

For future work we want to develop abstractions for several other sensor network components. Additionally we will examine the impact of different and dynamic topologies together with varying radio ranges on verification results.

References

[1] C. Baier and J.-P. Katoen, *Principles of Model Checking*, Cambridge, Mass: The MIT Press, 2008.

[2] P. Levis, N. Lee, M. Welsh and D. Culler, *TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications*, In Proceedings of the 1st International Conference on Embedded Networked Sensor Systems (SenSys 2003).

[3] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani and A. Tacchella, *NuSMV 2: An OpenSource Tool for Symbolic Model Checking*, In Proceedings of the 14th International Conference on Computer Aided Verification (CAV 2002). Springer, 2002.

[4] A. Hinton, M. Kwiatkowska, G. Norman and D. Parker, *PRISM: A Tool for Automatic Verification of Probabilistic Systems*, In Proceedings of the 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2006). Springer, 2006.

[5] V. G. Naik and A. P. Sistla, *Modeling and Verification of a Real Life Protocol Using Symbolic Model Checking*, In Proceedings of the 6th International Conference on Computer Aided Verification (CAV 1994). Springer, 1994.

[6] K. L. McMillan, *Getting Started with SMV*, User's Manual, Cadence Berkeley Laboratories, USA, 1998.

[7] A. Fehnker, L. van Hoesel and A. Mader, *Modelling and Verification of the LMAC Protocol for Wireless Sensor Networks*, In Proceedings of the 6th International Conference on Integrated Formal Methods (IFM 2007). Springer, 2007.

[8] G. Behrmann, A. David, K. G. Larsen, J. Hakansson, P. Pettersson, W. Yi and M. Hendriks, *Uppaal 4.0*, In Quantitative Evaluation of Systems (QEST 2006). IEEE Computer Society, 2006.

[9] K. L. McMillan, *Symbolic Model Checking: An approach to the state explosion problem*, Ph.D. thesis, Carnegie Mellon University, 1992.

Modeling Security Aspects of Network Aggregation Protocols

Frank Werner
Institut für Theoretische Informatik
Universität Karlsruhe (TH)
frank.werner@kit.edu

Raoul Steffen
Institut für Theoretische Informatik
Universität Karlsruhe (TH)
steffen@ira.uka.de

***Abstract*—We verify the correctness of a protocol for secure data aggregation using formal methods. For this purpose, a scenario is modeled and analyzed in which malicious nodes are randomly placed by an adversary among protocol compliant devices. We specify properties using linear temporal logic (LTL) and in combination with an implemented design, an exhaustive state space analysis is conducted using model checking techniques. We conclude this work with statements and assertions about the correctness of the investigated protocol and determine uncritical cases under which nodes forge packets but remain undetected.**

I. Introduction

The design of secure and safety critical protocols is even in the miniaturized world of embedded devices that run with less than 512kB of memory a challenge to engineers. It is complicated not only due to the complexity of the underlying algorithms but mainly because of the concurrent nature of such a distributed system. Simultaneous access to a common resource (like the communication medium) or the concurrent execution of tasks hamper not only implementation and design, but also the proof for correctness.

Formal methods have evolved over time to strong algorithms that can handle the complexity of those systems, and in contrary to testing and simulation, design flaws are found if they exist by an exhaustive state space analysis. One prominent tool that fulfills this claim is Spin, a model checker, that offers mechanisms for non-determinism and concurrent system modeling. The tool has demonstrated its suitability in various industrial case studies (e.g., [1], [10]) to be powerful enough in providing a sound basis for formal verification approach.

In this work we verify a protocol for authentic data aggregation by means of the Spin model checker [8] which has proven to be efficient enough to handle model instances with more than a million states and transitions. The formal modeling approach aims to complement the arguments found in the work of [5], provide new insights and possibly find potential yet undiscovered design issues in the protocol.

This paper is structured as follows. In Section II we briefly sketch the ESAWN [4] protocol with its main aspects and functionality. Section III describes the general assumptions like the adversary model. The Promela model including the modeling modalities and the properties of interest are defined and discussed in Section IV. There after in Section V the results are shown and finally Section VI wraps up this work and provides an outlook about future work.

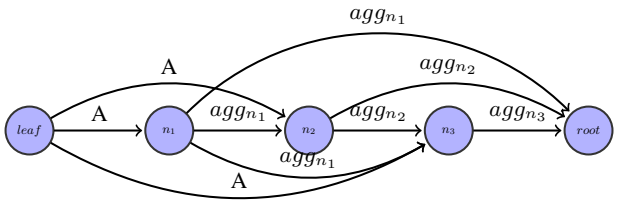


Fig. 1. ESAWN scenario of an aggregation sequence with 2 witnessing nodes ($W = 2$) i.e., that each aggregated is forwarded not only to the direct neighbor, but also to additionally two nodes which attest the correctness of the computed aggregates. For example node n_1 sends agg_{n_1} to node n_2, and in addition to n_3 and $root$ as witnesses.

II. The ESAWN Protocol

The investigated ESAWN protocol [4] (Extended secure aggregation for Wireless sensor Networks) handles the transport and aggregation of messages with guaranteed end-to-end authenticity in the presence of multiple compromised nodes.

This is mainly achieved by the use of witnesses like shown in the example of Figure 1: node $leaf$ sends value A to node n_1. Since node n_1 could be compromised witnessing nodes are involved which also receive the value A. In the above mentioned figure, two witnesses ($W = 2$) are used which attest the proper behavior of node n_1, namely node n_2 and n_3. Each witness compares the received aggregates with previously received aggregates, and in case that they equal, no faked aggregate was sent in between. On the other hand if the aggregates differ, there is at least one compromised node cheating which is made public by broadcasting an alarm message to all nodes. In the next step, a new aggregate is computed consisting of the node's own aggregate and the received ones. In the example of node n_2 the aggregation function $f_{n_2}(agg_{n_1}, A)$ computes the new aggregate agg_{n_2} which then encrypted and sent to parent nodes on the aggregation tree. By using a symmetric encryption (e.g., SKEY [2]) the authenticity of messages between nodes is achieved, since the protocol requires the authenticity of aggregates to be verifiable all the way down to the sink.

Each single data aggregate is sent and received multiple times (exactly $W+1$). Therefore it is obvious, that the protocol is causing a communication overhead and is very expensive in terms of energy. Due to this reason, the user has means to

relax the authenticity of the data that arrives at the root (base station), thus saving energy, but weakening the authenticity of the data. This is done using parameter p which gives the probability that a node verifies the correctness of an aggregate. In consequence with probability $1-p$ the packets' authenticity is not checked. The resulting trade-off behind ESAWN [4] is, the more authentic the user wants data to be sent over the network links, the more energy is needed to accomplish this.

III. Network and Intruder Assumptions

For the design of the ESAWN Model the following aspects are taken into account. We are interested in the security of the protocol and do not consider a relaxation of the authenticity. For this reason we set probability $p = 1$ meaning that all aggregates are checked. Note that in particular when setting $p < 1$ a fake aggregate will stay undetected with probability $1 - p$ and authenticity can no longer be guaranteed.

In addition, the scenario is chosen where nodes are lined up (see Fig. 1). In this case the ESAWN Protocol is not working most efficiently due to the missing aggregation of packets which result in fewer transmissions. On the other side the same properties hold here as in a tree-like scenario. Hence a tree-like spanning topology is not required to proof the protocol's correctness.

The use of aggregation is an essential part in ESAWN but does not need explicit modeling. Especially since we refrain from sending real measured data values it is sufficient to abstract from a concrete value by a data packet with what-so-ever content which dramatically reduces the model's complexity.

A. Adversary Model

The attacker can compromise some nodes (up to k) in the network by reading the nodes' memory, obtaining their secret keys, reprogramming and placing them undetected back to the network. It can randomly select the nodes although the highest potential security threat is in the case where compromised nodes are located closely together. Hereby they build a region of compromised nodes, cooperate among each other and amplify their impact on other legitimate nodes.

The *root* node is assumed to be out of the adversary's reach, operating honestly for the following reasons. If the sink would act malicious, there would be no meaningful verification of the authenticity possible since the user could not trust the base station. In addition if the attacker would have control over the root node, it has all means to take over control of the whole network sending its own requests which will not be denoted by the user.

The same holds for the *leaf* nodes. Here it can never be checked whether the extrinsically measured data is correct. As such, an adversary may forge a sensor's temperature sensor by simply using a lighter at the hardware, and the base station would not notice this counterfeit values.

IV. Promela Model

Promela (Process Meta-Language) [8] is the process description language for Spin with special emphasis on modeling process, synchronization and coordination. We model the real world ESAWN Protocol and define a variable number of N nodes to be present in our scenario. Out of this, there are up to k malicious nodes that forge packets from time to time but mostly operate normal and inconspicuous. We define variable W as the number of witnesses present, i.e., a node has to sent each packet to at least W witnesses which verify the correctness of its aggregates.

Four types of nodes form a network scenario namely the *leaf* in charge of initially sending the collected sensor data to the network which will always behave protocol conform. Among the inner nodes we distinct between *InnerNotCorrupt* nodes behaving honest and non-compromised, and *InnerCorrupt* nodes, trying to fake aggregates from time to time. The *root* node eventually received the aggregates and cannot be compromised by the adversary. The detailed behavior of the Spin process for the different type of nodes is depicted in Figure 2.

A. Modeling Channels

Before the actual Promela model of the ESAWN protocol is run, each node initializes the required message channels on the aggregation path. This means that each node obtains input channels from children nodes and is allocated outgoing channels to its succeeding parent nodes. The use of a separate channel for each node is legitimate and can be motivated by the fact that in the ESAWN protocol implementation nodes share pairwise symmetric keys (SKEY - Secure KEYing [11], [3]). In the Promela model we check these requirements by the use of channel assertions `xs` and `xr` on which nodes or processes – in terms of the Spin model – have exclusive access. The globally defined channels of type `chans`, defined as an 1-byte variable that can contain one data packet at a time.

B. State Variables

The global variables represent the overall state of the model and specify the LTL proof obligations. `CheatDetect` represents a detected faked aggregate. Note that only `InnerNotCorrupt` and `root` nodes detect a fake message since the intruder has no incentive to expose himself. This is modeling is compliant to the protocol which specifies an alarm message sent to all known surrounding nodes in case a fake aggregate is discovered.

The `root` process announces a received aggregate (`AnnounceRcv`). It then checks whether the received aggregate is correct in which case `RcvDataCorrect` is set. The variable `FakeNodes` is incremented whenever a compromised node turns active and each time a forget aggregate is sent counter `FakePacketsSnd` is incremented.

C. Process Models

Initially all four node types (`Leaf`, `InnerCorrupt`, `InnerNotCorrupt`, `Root`) initialize their channels. In

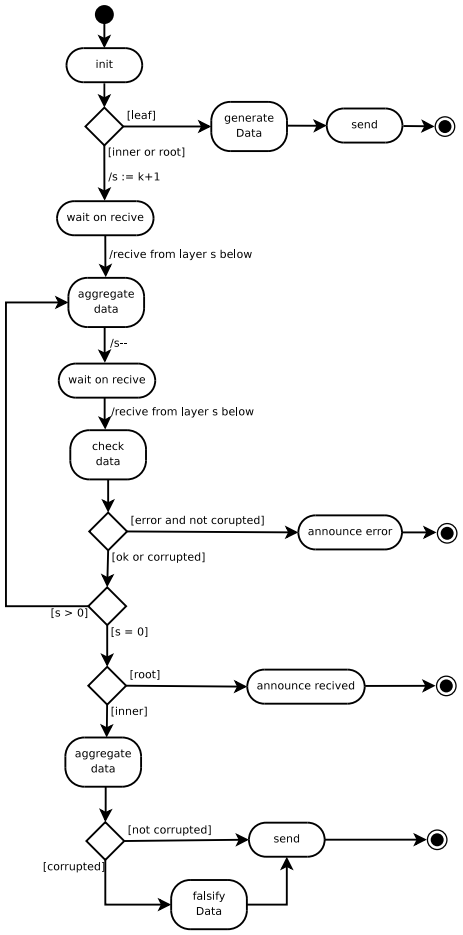


Fig. 2. Action diagram for the ESAWN protocol that is run in each Spin process independently.

more detail they set up channels with their designated predecessor and successor nodes.

1) Leaf Process: As in the scenario in Figure 1 the *leaf* generates a sample and initiates a protocol run by sending this sample its parent node and witnesses. The data packet is represented by value "0", representing the original value sent by the leaf node. After a successful sent the leaf process stops.

2) InnerNotCorrupt Node: A protocol compliant node waits until W children sent their packets and then compares the received data for equality. In case cheating is detected (`aggStore[`i`]!= aggStore[`$i-1$`]`) for one aggregate, variable `CheatDetect` is set to `true` and the node processing stops. In case that all aggregates equal, they are sent to all W parent nodes for witnessing.

3) InnerCorrupt Node: A malicious node behaves different from loyal ones. After the channel is initialized, they wait until all aggregates from child nodes are received. Where a non-malicious node verifies the aggregates, a corrupt node won't do so in order not to reveal an attack started by other corrupt nodes. In addition, a compromised node can suddenly turn active and send a forged aggregate to only a parent node, or at most W nodes.

4) Root Process: The *root node* waits for data to arrive. If aggregates arrive they are pairwise checked for equality and in case that this check fails, this is reported by setting variable `CheatDetect`. In case the results are all validated and all packets are checked for correctness, variable `RcvDataCorrect` is set to `true`. In addition variable `AnnounceRcv` is set and the process terminates.

D. Properties of Interest

In this section the model from above is feed into the model checker Spin. In the tool setting we set memory use for building the state space to 512MB. As parameters an estimated state space size of $500 \cdot 10^3$ and a maximum search depth of 10 000 steps is used. The LTL properties are defined as:

P1: $\Diamond(AnnounceRcv \vee CheatDetect)$
It is eventually the case that either the root *announces received* data ($AnnounceRcv$) or one or more nodes are cheating which is detected by at least one honest node (*cheating detected* $CheatDetect$). The use of "one node" is sufficient since it will trigger the alarm.

P2: $\Box AnnounceRcv \rightarrow (RcvDataCorrect \vee CheatDetect)$
Whenever the root node receives a data packet ($AnnounceRcv$), it is either identical with the one sent by the leaf node and no faked aggregates are sent (*received data correct* $RcvDataCorrect$) or at least one node detected a corruption ($CheatDetect$)

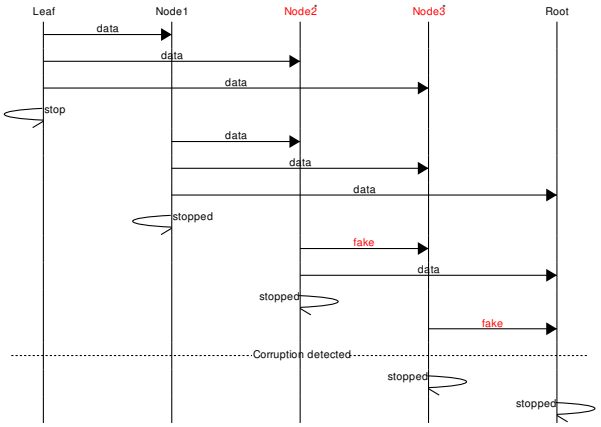
P3: $\Box(FakePacketsSnd > 0 \rightarrow \Diamond CheatDetect)$
Whenever there is a *forged packet send* ($FakePacketsSnd > 0$) this will eventually be detected and reported ($CheatDetect$)

P4: $\Box(RcvDataCorrect \rightarrow !(FakeNodes > 0))$
Whenever the data received by the root node is correct ($RcvDataCorrect$), there has been no faked message although *forging nodes* ($FakeNodes > 0$) might be present

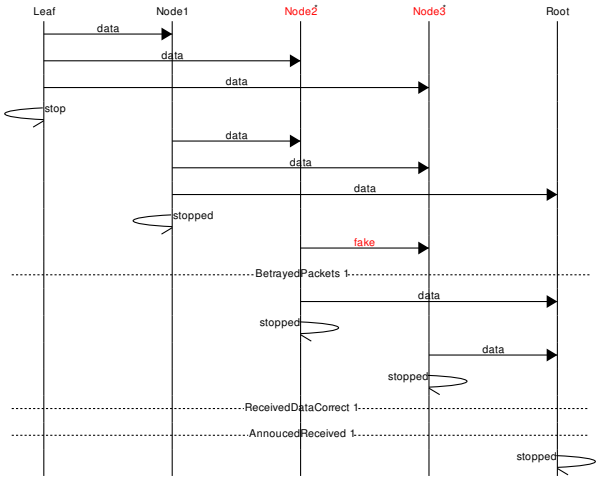
V. Results

The results are displayed for parameters $k = 2, N = 5, W = 3$ in the message sequence charts in Figure 3, which reflect only one possible trace of execution. The MSCs represent the behavior of the processes over time until termination. All global happenings like *corrupted node detected* or *betrayed packet* are displayed by dashed horizontal lines.

In the MSC of Figure 3(a) two compromised nodes (n_2, n_3) are present as denoted by the "*". The verification results are shown in Table I. Properties 1 and 2 are valid which state that no packets are lost. Consequentially, either cheating is detected, or a sound aggregate arrives at the root node. In contrary, properties 3 and 4 do not hold, proving that



(a) Correct run where cheating is detected



(b) Run where property 3 and 4 fail

Fig. 3. Message Sequence Charts (MSC) for two scenarios, where compromised nodes are indicated by the small asterisk.

property	result
$\Diamond(AnnounceRcv \vee CheatDetect)$	valid
$\Box AnnounceRcv \rightarrow (RcvDataCorrect \vee CheatDetect)$	valid
$\Box(FakePacketsSnd > 0 \rightarrow \Diamond CheatDetect)$	not valid
$\Box(RcvDataCorrect \rightarrow !(FakeNodes > 0))$	not valid

TABLE I
RESULTS WITH PARAMETERS $n = 5, k = 2, w = 3$

compromised nodes cooperate. In the MSC each of the compromised nodes is sending a faked packet to its direct child node as in Figure 3(b). Since we assume that the adversary wants to remain undetected, n_3 will not trigger an alarm since corrupt nodes cooperate. And thus the root receives two valid aggregates although a forged aggregate was sent. This does not necessary mean, that we discovered a flaw in the ESAWN protocol but rather that a scenario is possible where the data was received correctly, in the presence of a forged aggregate.

VI. CONCLUSION

The here presented analysis using the Spin tool verifies the authenticity and safety of the ESAWN protocol. In turn the failed property doesn't open space for intrusion attacks, since the adversary gained nothing in this case. Since we have not looked into the source code, we cannot guarantee the protocol's correctness after deploying it to a real world sensor network. Hence some uncertainties about the reliability of the hardware, the operation system of the sensor node, or the compiler still remain. Hence, using a more realistic model without the chosen level of abstraction and human interference would be desirable to have that could be extracted from the source code by tools like SLEDE [7] or Modex/Feaver[9]. SLEDE translates TinyOS protocols into Promela code automatically. Unfortunately only an old format of the TinyOS framework is supported by SLEDE up today.

Another way to continue this work is the generation of a behavior model using the TinyOS build-in *NULL* platform and afterwards use software verification tools like CBMC [6]. Although this approach would be restricted to sequential properties that describe the behavior of a single node, algorithmic aspects and errors introduced during the generation could be discovered before deployment.

REFERENCES

[1] Kusy B. and Abdelwahed S. FTSP Protocol Verification using SPIN. Technical Report ISIS-06-704, ISIS technical report, May 2006.

[2] Erik-Oliver Blaß, Michael Conrad, and Martina Zitterbart. A Tree-Based Approach for Secure Key Distribution in Wireless Sensor Networks. In *REALWSN – Workshop on Real-World Wireless Sensor Networks*, Stockholm, Sweden, June 2005.

[3] Erik-Oliver Blaß, Michael Conrad, and Martina Zitterbart. A Tree-Based Approach for Secure Key Distribution in Wireless Sensor Networks. In *REALWSN – Workshop on Real-World Wireless Sensor Networks*, Stockholm, Sweden, June 2005.

[4] Erik-Oliver Blaß, Joachim Wilke, and Martina Zitterbart. A Security–Energy Trade-Off for Authentic Aggregation in Sensor Networks. pages 135–137, Washington D.C., USA, September 2006. IEEE Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON), Extended Abstract. ISBN: 1-4244-0732-X.

[5] Erik-Oliver Blaß, Joachim Wilke, and Martina Zitterbart. Relaxed Authenticity for Data Aggregation in Wireless Sensor Networks. Istanbul, Turkey, September 2008. 4th International Conference on Security and Privacy in Communication Networks (SecureComm 2008). to appear.

[6] Edmund M. Clarke, Daniel Kroening, and Flavio Lerda. A Tool for Checking ANSI-C Programs. In Kurt Jensen and Andreas Podelski, editors, *10th Intl. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 2988 of *Lecture Notes in Computer Science*, pages 168–176. Springer, 2004.

[7] Youssef Hanna. Slede: lightweight verification of sensor network security protocol implementations. In *ESEC-FSE companion '07: The 6th Joint Meeting on European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering*, pages 591–594, New York, NY, USA, 2007. ACM.

[8] Gerard J. Holzmann. *The Spin Model Checker – Primer and Reference Manual*. Addison-Wesley, September 2003.

[9] Gerard J. Holzmann and Margaret H. Smith. Automating software feature verification. Technical report, Bell Laboratories, 2000.

[10] Tanaka Shin Ya, Sato Fumiaki, and Mizuno Tadanori. Multimedia network system. security protocol verification system based on spin. *Transactions of Information Processing Society of Japan*, 42(2):147–154, 2001.

[11] Martina Zitterbart and Erik-Oliver Blaß. An Efficient Key Establishment Scheme for Secure Aggregating Sensor Networks. In *ACM Symposium on Information, Computer and Communications Security*, pages 303–310, Taipei, Taiwan, March 2006. ISBN 1-59593-272-0.

UML 2.0 for modeling TinyOS components

Sebastian A. Bachmaier
Material Testing Institute
Universität Stuttgart
Stuttgart, Germany
sebastian.bachmaier@mpa.uni-stuttgart.de

***Abstract*— TinyOS' software architecture is based on the concept of components. These components are visualized graphically in many papers and documentation manuals. However, software architects and authors use their own graphical representation, usually. This makes it hard to understand new software quickly. The Unified Modeling Language (UML) also knows the concept of components. Although usually used in an object-oriented context, the notion is more general and UML can be used for the design of software architectures that do not base on the object-orientation paradigm. UML is nowadays known to most software engineers. The use of component diagrams as specified in UML 2.0 is therefore proposed for modeling TinyOS components, to facilitate the exchange of TinyOS software designs.**

***Index Terms*—UML 2.0, component diagram, TinyOS best practice, modeling, software architecture**

I. Introduction

The Unified Modeling Language (UML, [1]) is a standardized notation for modeling software architectures, usually used in the object oriented domain. UML includes therefore all graphical elements to model object oriented software. However, as UML is not a software development process but a notation only, it can be used in non-object oriented software development and in general in other engineering disciplines as well. UML consists of graphical notation elements and of diagram types that can be constructed from the notation elements. The graphical elements carry semantics in themselves but further meaning can be added by special notation elements, like stereotypes. UML comprises many traditional diagram types like state machines, timing diagrams and sequence diagrams, that have been in use in engineering sciences long before. However, with UML the notation is now standardized and therefore more accessible for someone who worked with UML before.

II. TinyOS and NesC Components

TinyOS programs (*applications* in TinyOS nomenclature) are written in a C programming language dialect called NesC [2]. While programming languages that are usually used for embedded system programming allow dynamic allocation of memory and dynamic binding of functions, NesC does not [3]. E.g., dynamic call resolution is too complex for general use in very constrained embedded systems. Therefore, NesC supports static binding only, however with a component model to allow for quick exchange of different implementations for certain functionality. The TinyOS component model is based on interfaces, which express the intersection of functions between components. A component can either offer or use an interface. By repeated application of this mechanism, hierarchal structures are obtained. The modules implementing or using an interface are *wired* by a special component, the *configuration*, which selects which user is connected to which provider.

III. Component Diagrams for TinyOS

A. Overview of currently used representations

Publications describing TinyOS designs usually refer to the component architecture of NesC. For better comprehensibility, the designs given are mostly backed by graphically representations. Examples are shown in Figure 1 to Figure 3.

In each of these figures, it can be seen, that the hierarchical provider and user structure is shown, as well as the interface between user and provider. However, the graphical representation is different for each figure.

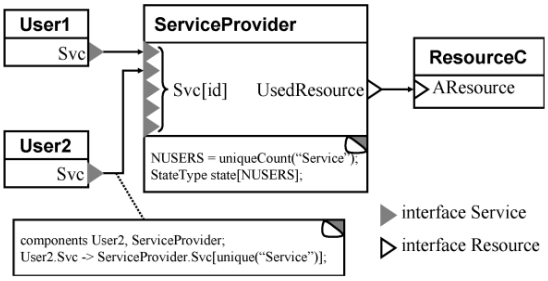


Figure 1. Component diagram as used by Culler [4]

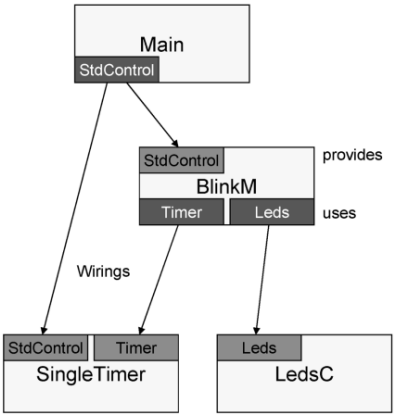


Figure 2. Component diagram as used by [5]

This work was funded by the 7th Framework Programme of the European Commission.

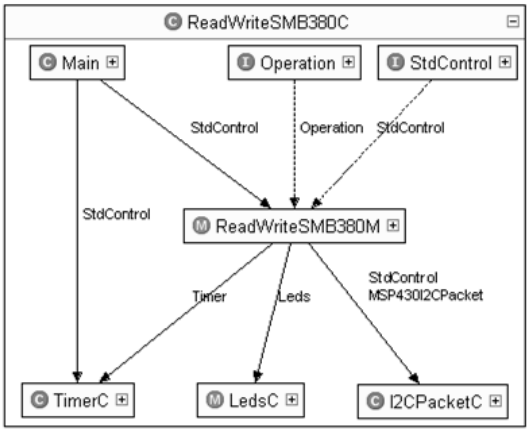


Figure 3. Component diagram as generated by the Eclipse plugins YETI [6] (and similarly YETI 2 [7])

B. *Use of UML 2*

The structure of a system, i.e. of an application, can be described by components, which represent reusable software units. According to [8], components enclose – among others – the following properties:

- a component comprises the specification of all realized and required interfaces
- a component can be exchanged by another which implements the same specification
- internals of a component are hidden, i.e. functionality of components is to be accessed via the offered interfaces only.

These three main properties are fulfilled for TinyOS components.

Components can have black-box representations, i.e. their internal implementation is not of interest, or white-box representations, where the realization is given, e.g. by means of nested components or class diagrams [9]. For the proposed TinyOS modeling, the less abstract white-box notation is proposed. The components offer interfaces and ports. Components can be exchanged by other similar components, resulting in a functionally equivalent application.

Component diagrams had been part of UML since the 1.0 version. The graphical representation changed slightly with UML 2.0 with regard to the component block layout (now a simple rectangle with a small rectangle with two bars as stereotype icon in the upper right corner) and the port element. Basic building blocks of a component diagram are the rectangle, representing a component (cf. Figure 4 a), the complete circle (*ball* or *lollipop*, naming from [10]) for a provided interface (cf. Figure 4 b) and the half-circle (*socket*) for a required ("used" in NesC-nomenclature) interface (cf. Figure 4 c).

Note that not all interfaces a component provides have to be used; however, for all required interfaces a provider is necessary.

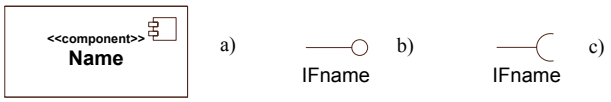


Figure 4. Building blocks of a component diagram:
a) component with name "Name" b) provided interface with interface name "IFname" c) required interface with interface name "IFname"

As an example, the Service Instance design pattern from [4] is given in Figure 5 in UML representation. For comparison, see original representation in Figure 1.

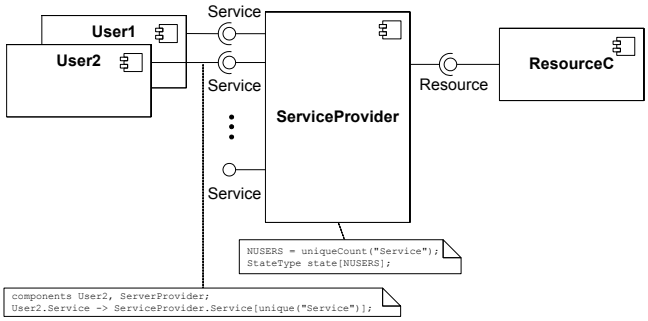


Figure 5: Service Instance pattern [4] in UML notation

The stereotypes "specification" and "realization" can be used to represent TinyOS configuration components and module components respectively. In Figure 6 these stereotypes are used to model the BlinkC component, as given in proprietary notation in Figure 2.

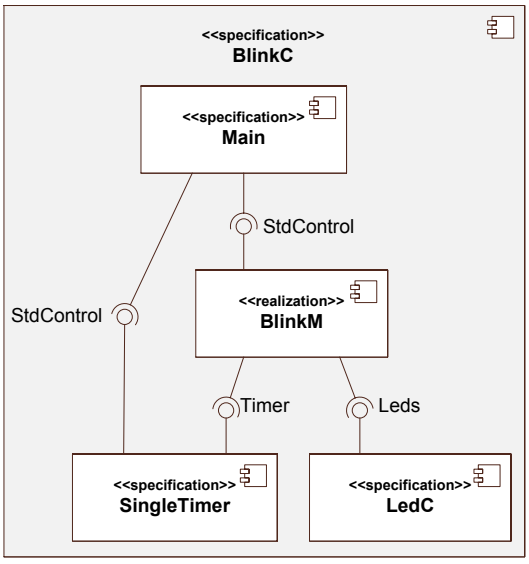


Figure 6. Component diagram in UML notation, showing use of "specification" and "realization" stereotypes

In Figure 7 the component of Figure 3 is depicted in the proposed notation, likewise giving an example of the UML *port* element. Ports assort interfaces that offer functionality together. In the suggested notation, ports offer public interfaces, which can be used by other components. Therefore, ports –

together with the *delegate* stereotype – lead interfaces through from the realizing component to the specifying component.

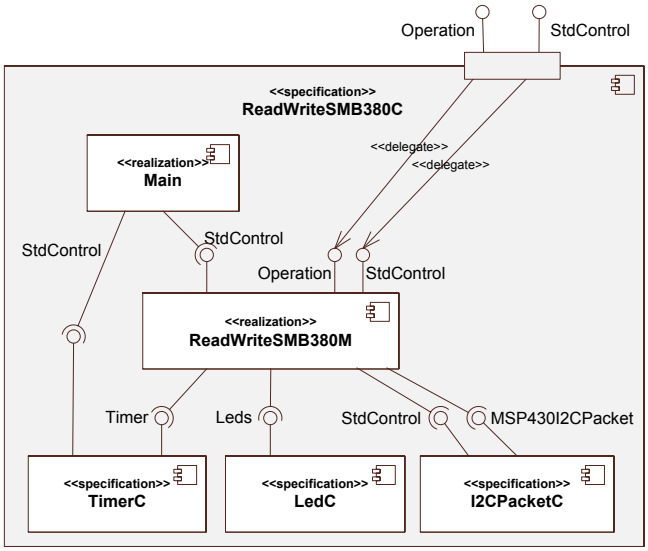


Figure 7. Component diagram in UML notation, depicting use of ports

C. *Applicability in larger systems*

Specifications of larger systems by use of this UML notation are possible, though they can become unclear if the diagrams are monolithic. One remedy is to model only one specification component with its realizing components in one diagram, as done in Figure 5 – 7. Between the single component view and the monolithic system view, any intermediate stage of diagram depth can be chosen, as appropriate to show the essentials of the system. Additionally, the "subsystem" stereotype can be used to model only specific parts of the whole system [11]. Using these techniques, the author was successful in specifying and documenting larger TinyOS applications.

In an automatic graphical tool, like in the YETI/YETI 2 tools, each specification component could be expanded and collapsed, thus revealing relevant and disclosing irrelevant parts of the diagram.

IV. Conclusion

The popularity of UML in computer science makes it the standard notation for documenting software architectures. The use of component diagrams is feasible and advisable to provide for a comprehensive insight to software designs, enabling efficient communication among developers and management.

The TinyOS community is invited to apply more UML and in general more design principles of the traditional PC-computer science to embedded systems. Standardization will simplify the software development for TinyOS, making design ideas more evident and finally resulting in a more economic and reusable software.

Acknowledgment

The author likes to thank Dr. Markus Krüger, Prof. Christian Große and the team of department "Non-destructive Testing and Monitoring Techniques" for their support.

References

[1] OMG Unified Modeling Language, Superstructure, V 2.1.2. Object Management Group Inc., 2007. Available: http://www.omg.org/spec/UML/2.1.2/Superstructure/PDF

[2] (no author given). http://nescc.sourceforge.net/, date of access: 2007-10-15

[3] P. Levis. (2006). *TinyOS Programming*. http://csl.stanford.edu/~pal, date of access: 2007-12-17

[4] D. Gay, P. Levis, D. Culler. (2005, July). Software design patterns for TinyOS. In *Proceedings of the 2005 ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems*. SESSION: System design issues table of contents. pp. 40-49

[5] A. Lachenmann. (2007). *Einführung in TinyOS und nesC* (in German). http://www.ipvs.uni-stuttgart.de/abteilungen/vs/lehre/lehrveranstaltungen/uebungen/SS07/UESN_termine/start, date of access: 2007-10-15

[6] R. Schuler, N. Burri. (2006). *TinyOS Plugin for Eclipse*, http://dcg.ethz.ch/~rschuler/OLD/index.htm, date of access: 2007-10-09

[7] Benjamin Sigg. (2008). *TinyOS 2 Plugin for Eclipse*, http://tos-ide.ethz.ch/wiki/index.php, date of access: 2008-10-09

[8] M. Born, E. Holz, O. Kath. (2004). *Sotwareentwicklung mit UML 2* (in German). Addison-Wesley

[9] P. Forbrig. (2007). *Objektorientierte Softwareentwicklung mit UML* (in German). 3rd ed., München: Carl Hanser

[10] S. W. Ambler. (2004). The Object Primer: Agile Model-Driven Development with UML 2.0, 3rd ed., Cambridge University Press

[11] D. Pilone, N. Pitman. (2005) *UML 2.0 in a Nutshell*. O'Reilly

[12] B. Oestereich (2005). *Analyse und Design mit UML 2* (in German). 7th ed., München: Oldenbourg

[13] http://www.tinyos.net/tinyos-2.x/doc/html/tutorial/lesson1.html, date of access: 2007-10-15

Drahtlose Sensoren für Batterie-Zellen
- ein Diskussionsbeitrag aus Sicht einer Anwendung -

Tobias Krannich, Stephan Plaschke, Karl-Ragmar Riemschneider, Jürgen Vollmer
Hochschule für Angewandte Wissenschaften Hamburg
Fakultät Technik und Informatik, Berliner Tor 5, 20099 Hamburg
vollmer@etech.haw-hamburg.de karl-ragmar.riemschneider@haw-hamburg.de

***Zusammenfassung*—Bisher werden die mehrzelligen Traktions- und Automobilbatterien nur als Gesamtsystem überwacht. Es wird ein Lösungsansatz vorgestellt, bei dem jede Zelle dieser Batterien mit drahtlosen Sensoren ausgestattet wird. Das erlaubt den robusten und vor allem kostengünstigen Zugang zu den Zellen. Die Sensoren messen die Zellenspannungen und die Temperaturen. Mit dieser Vielzahl an zusätzlichen Messstellen kann die Batterieüberwachung, die Zustandsprognose und der wirtschaftliche Batteriebetrieb erheblich verbessert werden. An der HAW Hamburg wurden verschiedende Varianten von drahtlosen Sensoren aufgebaut. Auf einer Gabelstapler-Traktionsbatterie wurde eine Batterieüberwachung erprobt. Die Anwendung für Automobilbatterien soll als Hauptziel folgen.**

I. Einführung

Große mehrzellige Akkumulatorbatterien finden sich in vielen Bereichen unserer technischen und industriellen Umgebung wieder. Von ihnen wird hohe Zuverlässigkeit und Wirtschaftlichkeit verlangt.

Traktionsbatterien liefern die Energie für einen großen Teil der Gabelstapler, für Förderwagen in automatisierten Fabriken und die umweltfreundlichen Hybridautos der Zukunft.

Weltweit starten viele hundert Millionen Akkumulatoren die Verbrennungsmotoren unserer Automobile[1]. Darüber hinaus puffern sie das Automobil-Bordnetz mit immer mehr elektrischen Verbrauchern. In dieser Rolle kommt der altbekannten Autobatterie in bereits wenigen Jahren eine lebenswichtige Funktion zu, wenn mit elektrischer Energie gelenkt und gebremst werden wird.

Problem: Unterschiedliche Zellenzustände

Die Alterung der Batterie führt im Laufe der Betriebsdauer dazu, dass der Zustand der in Reihe geschalteten Zellen sehr unterschiedlich wird. Bisher ist eine zuverlässige Beurteilung der verbleibenden Batteriekapazität nur mit erheblicher Unsicherheit möglich, da keine Messwerte von allen Zellen verfügbar sind, sondern die Gesamtbatterie nur als 'Blackbox' betrachtet wird.

Lösungsansatz: Einzel-Zellen-Überwachung

Im verfolgten neuartigen Lösungsansatz werden Messwerte von jeder einzelnen Batteriezelle aufgenommen und drahtlos übertragen. Sehr kompakte Sensor-Mikrocontroller-Systeme sollen dazu direkt in jeder Zelle montiert werden. Diese Sensoren sind so robust auszulegen, dass sie in den Innenraum der Batteriezelle eingebaut werden können. Dort werden sie an beiden Elektrodenplatten angeschlossen, andere Leitungsverbindungen sind nicht notwendig. Die ermittelten und vorverarbeiteten Sensordaten der Zellen werden von einer zentralen Einheit gesammelt und ausgewertet. Gemeinsam bilden sie ein Netz, bei dem Kommunikation und Messaufgabe aufeinander sehr eng abgestimmt sind.

Die Überwachung erfolgt also in einem Zusammenwirken aus verteilter Datenerfassung und Vorverarbeitung, einer auf sehr verschiedenartige Betriebsfälle angepassten Netzwerkkommunikation und einem 'Entscheider' auf der Basis eines zu entwicklenden, verteilten Batteriemodells.

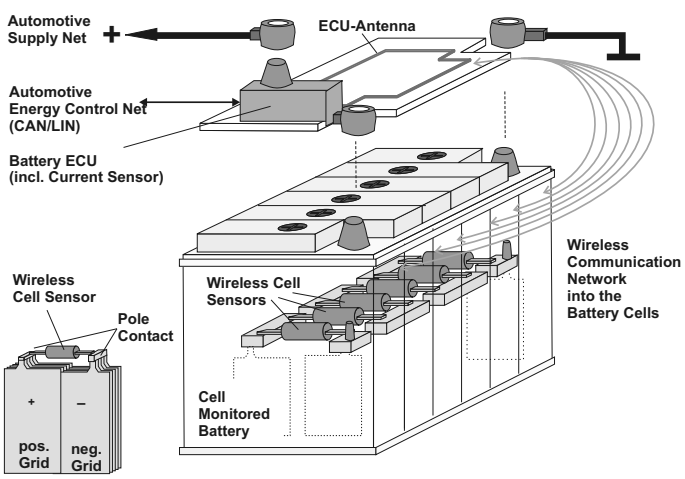


Abbildung 1. Konzept für die Überwachung der Automobilbatterie

II. Anwendungen

Industrielle Anwendung: Gabelstapler-Traktionsbatterien

Elektrische Flurförderzeuge - insbesondere Gabelstapler - werden von Bleibatterien versorgt, die bis zu 40 Zellen umfassen. Sie wiegen teilweise über 1.5 Tonnen und kosten einige tausend Euro. Abschätzungen ergeben, dass durch besseres Batteriemanagement eine bis zu 30 % höhere Wirtschaftlichkeit und Verfügbarkeit der Batterie erreicht werden könnte. Dies entspricht einem Anwendernutzen von vielen hundert Euro je Traktionsbatterie.

[1]Produziert werden 300-350 Mio Starterbatterien für über 8 Mrd. $ p.a., in Deutschland über 15 Mio. Stück im Wert von ca. 300 Mio Euro, davon ein Drittel für Neufahrzeuge

Fernziel: Automobilbatterie

Die Abbildung 1 zeigt ein Grobkonzept der Verwendung im Automobil, das mit Industriepartnern diskutiert wurde.

Der Ausgangspunkt für die Einführung in das Automobil ist die Einbeziehung der Gesamtstrommessung, weil das bereits bei ersten Fahrzeugen in Serie erfolgt. Dort führt man die Plusleitung der Batterie über einen Shunt durch das Batteriesteuergerät, das wird als „intelligente Batterieklemme“ beworben [4] [9]. Die Anbindung an andere Steuergeräte erfolgt über einen CAN-Bus oder wahrscheinlicher über einen LIN-Bus. Angebunden wird sicherlich das Bordnetzsteuergerät bzw. der Feldregler des Generators, der zukünftig ebenfalls einen Buszugang erhalten wird. Das Batteriesteuergerät mit der drahtlosen Sensorik wird als Erweiterung der „intelligenten Batterieklemme“ gesehen.

Für die Antenne des Steuergerätes soll eine Konstruktion gefunden werden, die kompakt und mit verschiedenen Batteriegrößen und Einbausituationen in den Fahrzeugen kompatibel ist. Die Sensoren selbstsind hingegen konstruktiv vorteilhaft, weil sie ohne Anschlüsse in der Batterie „verschwinden“.

Wirtschaftliche Rahmenbedingungen

Die Traktionsbatterie lässt Lösungen in naher Zukunft technisch machbar erscheinen. Der geschätzte Kostenrahmen von ca. hundert Euro pro System ist mit konventionellen Bauelementen, also für die Stückzahlen bei Gabelstaplern passend, erreichbar. Der Nutzen muss noch durch Feldversuche quantifiziert werden. Für Erprobungen und Experimente bildet die Traktionsbatterie eine gute Plattform.

Für die Automobile der Zukunft muss die Zuverlässigkeit der Versorgung mit elektrischer Energie zwingend verbessert werden. Daher kann ein Kostenbeitrag von den kostensparenden elektrischen Systemen in der Fahrfunktion, dazu gehören die elektrischen Lenkhilfen und die elektromechanischen Bremse (EMB), kommen. Die Automobilbatterie ist dennoch preissensitiv. Nach diskutierten Schätzungen sind Herstellkosten von weniger als 20 Euro für das Gesamtsystem der Batterieüberwachung bzw. von weniger als einem Euro pro Sensor erforderlich. Das wird nur durch vollständig integrierte, maßgeschneiderte Lösungen in Massenproduktion zu erreichen sein. Viele Fragen dazu sind offen, zumindest der Marktumfang spricht nicht dagegen.

III. Ziele und Lösungen

Zuverlässigkeit durch bessere Prognose

Durch die Alterung können die Zellen im Lebensdauerzustand, dem State of Health (SOH), sehr unterschiedlich werden. Einem Batterie-Zustandsschätzer im Fahrzeug oder im Ladesystems steht bisher nur die Gesamtspannung und ggf. der Gesamtstrom zur Verfügung. Dadurch wird es zu einer zunehmenden Fehlbeurteilung des Ladezustandes, des State of Charge (SOC), kommen. Noch entscheidender ist für die elektrische Bremse die Prognose der sog. Hochstromverfügbarkeit, des State of Function (SOF). Nur mit Hilfe der zellenweisen Überwachung kann ein Batteriemodell einzelne Zellen als geschädigt, d.h. nicht mehr hochstromfähig, identifizieren. Damit wird dieser bisher unerkannte, aber kritische Zustand rechtzeitig bemerkt.

Überwachung gegen Schädigungen erhöht Lebensdauer

Für Batteriezellen sind bekanntermaßen drei Zustände schädlich. Das sind der Betrieb bei Übertemperatur, die Überladung und die Tiefentladung.

Durch das Nichterkennen der Tiefentladung wird jedoch die vorgeschädigte Zelle nochmals besonders geschädigt. Dieser Effekt verstärkt sich fortlaufend und führt letztlich zum unerwarteten, scheinbar plötzlichen Ausfall.

Eine Zellenüberwachung anstelle der Gesamtbatterieüberwachung löst das Problem, bisher unerkannte Tiefentladungen können nun rechtzeitig bemerkt und Überladungen verhindert werden.

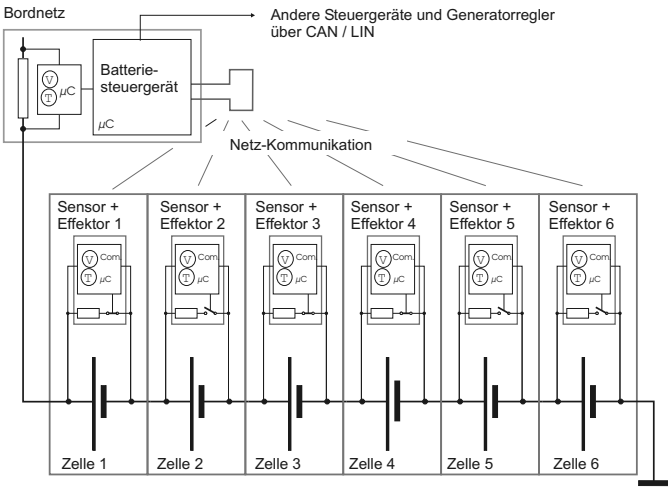


Abbildung 2. Prinzipdarstellung der Ladungsbalancierung: Als Beispiel werden Parallelstrompfade für die Zelle 1,3 und 4 geschaltet, um den Ladezustand (SOC) an die Zellen 2,5,6 anzugleichen

Besonderheit: Ladungsbalancierung durch Effektor im Sensorknoten

Sind ohnehin Sensorknoten in jede Zelle montiert, dann kann dort auch ein einfacher Effektor realisiert werden.

Dieser Effektor ist ein jeweils einschaltbarer Parallelstrompfad zu den Zellenpolen von mittlerer Belastbarkeit und mit Strombegrenzung[2] . Wird dieser Pfad bei einer weniger gealterten Zelle während der Ladung eingeschaltet, dann wird dort mit etwas weniger Strom geladen, die anderen Zelle werden umso stärker geladen. Wird dieser Parallelpfad in der Ruhephase bei Zellen mit hoher Zellenspannung zeitweise eingeschaltet, verlieren sie langsam einen Teil der zuviel gespeicherten Ladung.

[2]Nur bis zu einem Strom von 200mA bis 500mA erscheint ein elektronischer Schalter für den Parallelpfad integrierbar zu sein, es geht also um eine Langzeit-Beeinflussung.

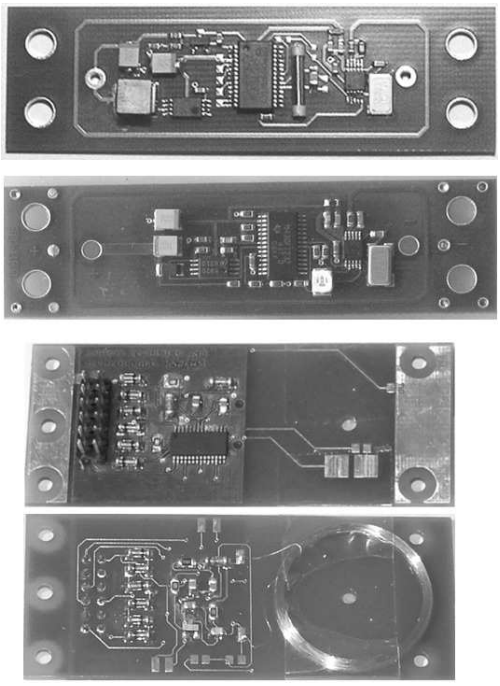


Abbildung 3. Versuchsmuster für Zellensensoren

oben: **Version UHF/LF**
Sensorknoten mit UHF-Uplink und mit LF-Downlink, noch nicht verwendet
darunter: **Version UHF/–**
Sensorknoten mit UHF-Uplink und ohne LF-Downlink
Muster wurden eingesetzt für Vorversuch Traktionsbatterie [13]
darunter/unten: **Version LF/LF**
Sensorknoten mit LF-Downlink und LF-Uplink (Vorder-/Rückseite)
Muster erprobt in Laborversuchen [7][1]

Der Vorteil eines solchen Ausgleiches liegt darin, dass der nächste Lade-/Entlade-Zyklus auf gleichem oder zumindestens besser angeglichenem Ladungsniveau beginnt. Batteriefachleute nennen das Verfahren Symmetrierung oder Chargebalancing. Bei stationären Industriebatterien wird das Balancieren durch das Wartungspersonal vorgenommen. Bisher war das Verfahren für eine Automobilbatterie kaum denkbar. Der drahtlose Zugang zur Zelle könnte das nun ermöglichen.

Die Einbringung eines Effektors ist noch vertieft zu untersuchen und zu bewerten. Dazu muss für die Sensorcontroller ein dezentrales, ein kooperatives oder ein zentrales Entscheidungsmodell erarbeitet werden.

IV. Praktische Vorarbeiten

Die Vorarbeiten führten bisher zu drei Varianten:

Version UHF/LF: Der Sensor-Controller MSP430F1232 mit 10-bit-ADC und integriertem Temperatursensor kommuniziert mit einem UHF-Uplink[3] durch einen mit OOK gesteuerten PLL-Schaltkreis (433MHz ISM) und einer angepassten Schleifenantenne auf der Trägerplatine. Der LF-Downlink mit 125 KHz Träger benutzt eine Spulen-Antenne mit Ferritkern, die auch für RF-ID-Transponder (Wegfahrsperre) genutzt wird. Dieser Down-Link ist primär für eine einfache Wake-up-Funktion und Synchronisierung konzipiert.

Durch einen Step-Up-Converter ist der Abfall der versorgenden Zellenspannung sogar bis unter 0.5 Volt erlaubt.

Version UHF/–: Es wurde auf den Downlink verzichtet, um den Aufwand für eine erste Versuchsserie zu reduzieren[13]. Die Übertragung hat sich als sehr robust erwiesen, sowohl unter ungünstigsten Ausbreitungsbedingungen (Batteriedeckel und Trog aus Metall), als auch in der Störsicherheit (Antriebsmotoren, Ladegerät).

Version LF/LF: Dieser Sensor nutzt Kommunikations-Lösungen, die in der RF-ID-Technologie verbreitet sind. Es wird ein gemeinsamer Antennenschwingkreis für beide Richtungen benutzt. Für den Downlink und Uplink wird ein 125 kHz Träger amplitudenmoduliert. Der Uplink benutzt die Modulation durch Belastung des Antennenkreises (2-ASK). Als Besonderheit ist zu nennen, dass keine Betriebsspannung aus der Zelle entnommen werden muss, diese wird durch Gleichrichtung des Downlink-Signals gewonnen. Auch diese Lösung liefert über einige Dezimeter Reichweite verlässliche Messwerte. Die Datenübertragung ist im Down- und Uplink bis zu 5 kBit/s brutto möglich [7]. Die Energiebilanz des Controllers und der Kommunikation ist den Versionen zuvor überlegen. Von Vorteil ist sicher die niedrige Trägerfrequenz, welche im metallischen Umfeld weniger Abschattung und Nullstellen hat.

Entscheidung für proprietäre Lösungen

Für die Übertragung wurde in den genannten Versionen kein standardisiertes Verfahren oder kommerzielles Modul benutzt, sondern eine Eigenimplementation. Der verwendete Controller kann die Mess- und Übertragungsaufgaben auch noch bei energiesparenden 80 kHz Takt durchführen.

Alle Varianten benutzen im Basisband den ungenauen und temperaturdriftenden integrierten RC-Oszillator [7], ein Quarz sollte hier bewusst eingespart werden. Die eigene Lösung lässt ein Einmessen des Empfängers auf die Zeitfehler des Senders zu (sog. Run-In-Verfahren).

Die Versionen mit UHF-Uplink benötigen jedoch zwingend einen Quarz-Oszillator für die Sende-PLL, um den ISM-Regularien zu genügen. Für automotive Anwendungen qualifiziert, ist ein solcher Quarz teurer als das Die eines kleinen Controllers (s. auch [14]). Die Kosten würden sich mit der Zellenanzahl multiplizieren, also sich mindestens sechsfach auswirken. Die LF/LF-Version benötigt keinen Quarz im Sensor, jedoch eine vergleichsweise aufwändige Antennenspule. Optimierte Step-Up-Converterspulen der ersten beiden Versionen besitzen ebenfalls nennenswerte Kosten. Für die Anwendung auf Traktionsbatterien sind diese Bauelementekosten noch erträglich, jedoch wird für die Automobilbatterie eines günstigeres technisches Konzept gefordert.

Bei den Antennenkosten hat die UHF-Übertragung klare Vorteile. Für die LF-Varianten muss eine größere Spulenan-

[3]Uplink bedeutet hier von den Sensoren zum Steuergerät, Downlink vom Steuergerät zu den Sensoren.

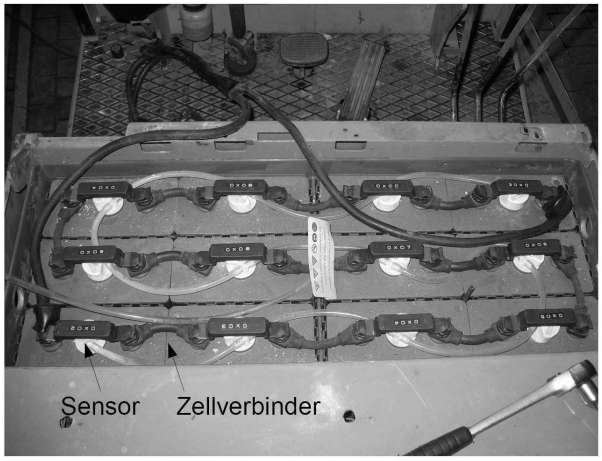


Abbildung 4. Drahtlos kommunizierende Zellensensoren auf der Batterie des Gabelstaplers der HAW Hamburg [13]

tenne oder Ferritantenne am Steuergerät genutzt werden, die ähnlich der eines RF-ID-Readers aufgebaut ist.

Erste Ergebnisse

Durch die Versuche an der Traktionsbatterie wurden erste Erfahrungen gewonnen, siehe Abbildung 4. So zeigte sich ein starker Unterschied im Lebensdauerzustand (SOH) durch Alterung der Zellen. Für eine Zelle wird die aktuelle Kapazität (SOC) bis zur Tiefentladung ausgeschöpft, während die anderen Zellen noch über nutzbare Kapazität verfügen, siehe Abbildung 5.

Die Temperaturwerte der Zellen sind von erheblichem Wert für die Batteriemodelle. Die Temperaturmessung der Controller über integrierte Dioden besitzt große Streuungen im Anfangswert und im Temperaturkoeffizienten. Daher wurde eine einfache Kalibrierung für den Sensor entwickelt. Prinzipiell sind die Sensoren, die auf der Zelle montiert werden, denen in der Zelle bezüglich der Temperaturbestimmung unterlegen.

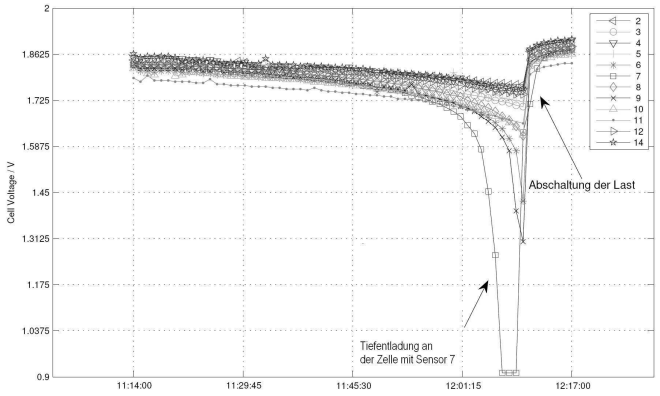


Abbildung 5. Zellenspannungen während 75 Minuten Entladung mit Vollast des Stapler-Antriebsmotors[13]. Der einbrechende Spannungsverlauf an einen Sensor zeigt eine Zelle mit schädigender Tiefentladung, dieses Problem war bisher an der Spannung der Gesamtbatterie nicht erkennbar.

V. Zusammenfassung und Bewertung

Bisher wurde für die Anwendung gezeigt:

- die Zellensensoren liefern wertvolle Informationen und sind gut in bestehende Batteriesysteme integrierbar
- die drahtlose Arbeitsweise ist stabil und für die rauhe Umgebung vorteilhaft
- Standardübertragungsverfahren oder Funkmodule, die einen Quarz im Sensor erfordern, waren nicht notwendig
- auch handelsübliche Controller sind geeignet, um wie RF-ID-Transponder ohne Energiequelle zu arbeiten
- dort können Systemtakte unter 100 kHz ausreichen, wenn auf Softwareoverhead verzichtet wird
- eine niedrige Betriebsfrequenz kann Vorteile haben, z.B. für Wakeup-Funktionen oder die Synchronisierung

Viele Aspekte sind noch offen, unter anderem:

- die Erfassung von schnellen Laständerungen, zunächst hat die Werterfassung statistischen Charakter
- die Nutzung des Sensor-Controllers, um autark Bewertungen und Entscheidungen zu treffen (z.B. Messregime bei schnellen Ereignissen ändern)
- Konstruktive Fragen (z.B. Umspritzung), Feldversuche

Der Ansatz, dass Sensornetze für große Batterien gerade durch die drahtlose Auslegung Vorteile bieten, ist durch die Arbeiten bestärkt worden. Favorisiert werden von den Autoren Lösungen, die sich am RF-ID-Bereich orientieren.

Literatur

[1] Eger, Torsten; Entwicklung von Hard- und Software eines Readers für drahtlose Sensorik mit Resonanzabgleich; Diplomarbeit an der HAW Hamburg 2008
[2] Finkenzeller, Klaus, RFID-Handbuch, 4. Auflage, Hansa-Verlag, 2006
[3] Form, T., Vorlesung Fahrzeugelektronik , TU Braunschweig, 2006 inbs. Teil 5 Teil 8 und Teil 9 über http://www.ifr.ing.tu-bs.de erreichbar
[4] Heim, A. (BMW Group München) Intelligenter Batteriesensor: Schlüsselkomponente für das Energiemanagement der Zukunft VDI-Berichte, Elektronik im Kraftfahrzeug, Baden-Baden, 2003
[5] Heinemann, D., Strukturen von Batterie- und Energiemanagementsystemen mit Bleibatterien und Ultracaps, Dissertation, TU Berlin 2007
[6] Hetzler, Ullrich, Batterie- und Energiemanagement in Kraftfahrzeugen-Anwendungen von ISA-ASIC und Präzisionsshunt bei BMW-Fahrzeugen E&E-Kompendium (Referenzbook Elektronik und Entwicklung) 2005/2006 www.isa-asic.de/files/2005-06-eekompendium.pdf
[7] Krannich, Tobias Experimentalsystem für einen Sensor-Controller mit drahtloser Energie- und Datenübertragung; Diplomarbeit an der HAW Hamburg 2008
[8] Leohold, Jürgen, Die elektrische Infrastrukur für zukünftige Fahrerassistenzsysteme, 5. Braunschweiger Symposium, Automatisierung und Assistenzsystem für Transportmittel 2004
[9] N.N. Der Elektronische Batteriesensor, Firmenschrift Bosch, 2007
[10] N.N., Transponder Cookbook, Part 4, FAE Training Identifikation Firmenschrift Philips, 1993
[11] N.N. HellaKGaA Hueck & Co., Energiemanagement - Technische Informationen (Broschüre) 2008 im Internet www.hella.com/.../ti_em_d.pdf
[12] Notten, P. (Philips Research/TU Eindhoven) Battery Management Methods, pers. Kommunikation
[13] Plaschke, Stephan; Experimentalsystem für drahtlose Batteriesensorik; Diplomarbeit an der HAW Hamburg 2008
[14] Polityko D.-D., Niedermayer M, Guttowski S, John W., Reichl H.; Drahtlose Sensornetzwerke: Aspekte der Hardwareminiaturisierung 2. GI/ITG KuVS Fachgespräch Drahtlose Sensornetze, 2004
[15] Pop, V., Bergveld, H., Danilov, D., Regtien, P.., Notten, P. Battery Management Systems, Accurate State-of-Charge Indication for Battery-Powered Applications Philips Research Book Series 2008
[16] Riemschneider, K.-R. Wireless Battery Management System, Pat.appl. WO2004/047215A1, US020060152190A1, EP000001573851A1

Wireless Energy Meter

Philip Rings, Lasse Thiem, Thomas Luckenbach
Fraunhofer Institute for Open Communication Systems
FOKUS
10589 Berlin, Germany
[philip.rings, lasse.thiem, thomas.luckenbach]@fokus.fraunhofer.de

Abstract— **This paper describes the ongoing development of an Energy Meter (EM) with an interface to a wireless sensor network based on IEEE 802.15.4 technology.**

I. INTRODUCTION

In the age of ecological awareness, and the conclusion that energy should not be wasted, it is necessary to investigate the places where electrical energy is consumed. The effective energy usage of a device can often be determined by monitoring the according electrical devices over a long space of time. The measuring can easily be done by simple Energy Meters from the local do-it-yourself store. But sometimes the limited functionality of these meters also limits their area of application. If, for example, a refrigerator shall be measured, the power cord may not be easily accessible. And most of the "standard" Wireless Energy Meters show their calculations on a small display directly on the meter. A solution is a Wireless Energy Meter which is able to transmit its measured data wirelessly to a computer. There the data can be interpreted and maybe even compared to data from other wirelessly connected devices.

II. RELATED WORK

In [1] an AC Meter (ACME) development for IP based Wireless AC Energy Monitors is presented. The developed platform consists of four main sections: power supply, signal filtering, Wireless Energy Metering and microcontroller. All embedded Software developments are done with TinyOS. In [2] an electronic system is described to measure the active, apparent and reactive energies delivered to a load of an AC voltage line. The proposed system is directly connected to a PC running visualization software of the power consumption. In [3] a case study of an inexpensive student designed power monitoring instrument for campus submetering is described. This system is build up with an Energy Metering IC connected to a PIC controller. Over a serial interface measured values can be accessed.

III. HARDWARE OVERVIEW

The Wireless Energy Meter (see also Figure 1) consists of two major elements to measure the different kinds of the power consumption. These two elements are the IEEE 802.15.4 [4] ICradio Module 2.4G [5] and the Energy Measurement IC ADE7753 [6] which will be introduced below.

A. *IEEE 802.15.4 - ICradio Module 2.4G*

The ICradio Module 2.4G is a compact, flexible deployable radio module, which is required for operation in wireless sensor networks (WSN) like in IEEE802.15.4. The 2.4G module consists of an ATMega1281 AVR microcontroller and the AT86RF230 2.4GHz radio chip from Atmel. It is completely compatible with Atmel's free IEEE802.15.4 MAC software.

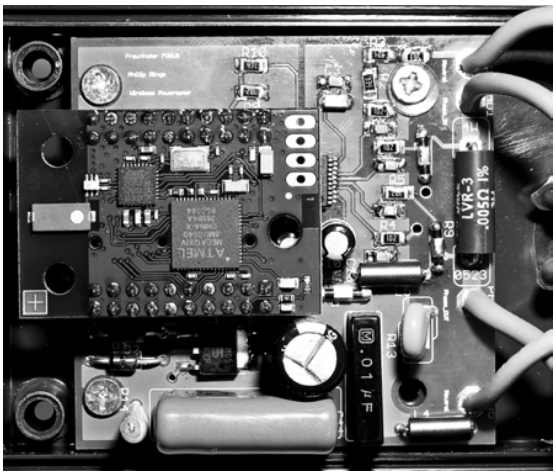


Figure 1. Wireless Energy Meter (EM)

B. *ADE7753*

The ADE7753 allows the control and the read out of several energy registers over a serial interface which is compatible to SPI. Through this registers it is possible to read out the accumulated values of active and apparent energy, the rms values of current and voltage and for example get information about the line period. In addition, it is possible to write to the ADE7753 registers to configure for example gains or compensate offsets among all the operations.

C. *Wireless Energy Meter Hardware Design*

The needed voltages for the two main hardware components, the ADE7753 and the ICradio Module 2.4G, are provided by a power supply which uses direct rectification. For the current sensing a shunt resistor is used, over which a comparatively low voltage drops when a current flows. The voltage sensing is done by a simple voltage divider, which scales the line voltage from 230Vrms to about ±0.5V. Two low-pass-filters at the analog inputs of the ADE7753 prevent

the measured signals from containing aliasing effects which could produce error components during the calculations. The analog-to-digital conversion of the voltage and current metering and afterwards all signal processing for the energy accumulation is done by the ADE7753. This IC is therefore controlled by the microcontroller which also is responsible for controlling the transceiver and keeping contact to the IEEE802.15.4 network. To make the node's hardware scalable, the ICradio Module 2.4G is connected to the invented circuit board via contact plugs. So a firmware update or the connection of additional hardware components can easily be done (see also Fig. 1).

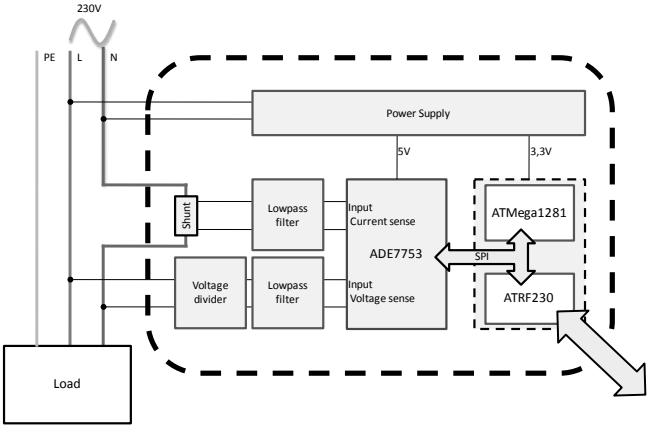


Figure 2. FOKUS Wireless Energy Meter hardware design

D. SPI Communication

The entire communication process to the ADE7753 takes exactly 225us (see also Figure 3). During this time frame the transceiver has no SPI connection to the microcontroller because only one device at time can transfer data to the microcontroller via SPI. Thus, the EM node is temporary not part of the wireless network. In multi-hop networks the routing protocol has to take this into account. In this current setup the communication is done by a pull mechanism of the node itself to a sink node in a peer-to-peer star network.

Read operation	t=0,0s	t=33,6us	t=86,7us	t=138,5us	t=190,4us
MOSI	3F 00	02 00 00 00	17 00 00 00	16 00 00 00	27 00 00
MISO	00 02	00 0B 07 5F	00 20 53 46	00 1C 4D 9E	00 22 10
Adress	0x3F	0x02	0x17	0x16	0x27
Name of register	DIEREV	AENERGY	VRMS	IRMS	PERIOD
Size	8 Bit	24 Bit	24 Bit	24 Bit	16 Bit
Content (Hex.)	0x02	0x0B075F	0x205346	0x1C4D9E	0x2210
Content (Dez.)	2	722783	2118470	2118470	8720

Figure 3. Results of the SPI communication analysis

IV. VISUALIZATION

The visualization of the measured consumed power of all devices in a home for example allows a real time feedback to the residents of a home.

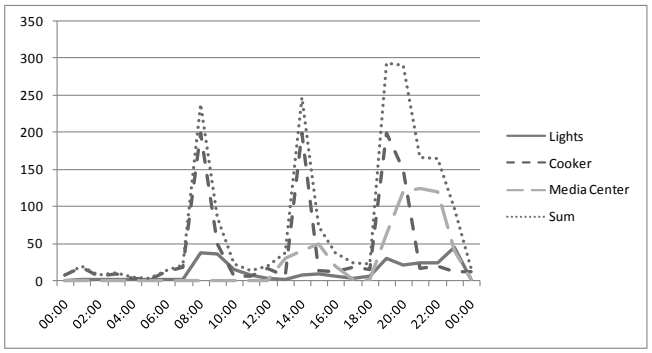


Figure 4. Typical daily power consumption

V. MEASUREMENT

For evaluation purposes some measurements with the hardware design were done to prove the system concept. Two of these measurements are described in the following to sections.

a) Energy

Analyses of the accuracy of the Wireless Energy Meter have shown that the system produces a failure rate of less than 0.1% (see also Figure 5). Several measurements with known parameters of the load have approved the accuracy.

Measurement A	expected [Wh]	determined [Wh]	difference [Wh]	failure (%)
A1	6,27898	6,32245	0,04348	0,00273
A2	6,36768	6,41180	0,04412	0,00281
A3	6,34710	6,39281	0,04571	0,00290
A4	6,34710	6,39646	0,04936	0,00313
A5	6,34710	6,37913	0,03202	0,00203

Measurement B	expected [Wh]	determined [Wh]	difference [Wh]	failure (%)
B1	0,62465	0,62046	0,00419	0,00003
B2	0,62465	0,61807	0,00659	0,00004
B3	0,62465	0,61846	0,00619	0,00004
B4	0,62465	0,62183	0,00282	0,00002
B5	0,62458	0,63625	0,01167	0,00007

Figure 5. Measurements about the accuracy of the Energ Meter

VI. CONCLUSION AND FUTURE WORIK

In this paper a wireless energy metering device is described which allows the visualization of power consumption of electrical devices like a TV or washing machine in real time. The current system enables residents to have an immediate overview about the actual and short term history power consumption. Future work will include developments towards automatic and remote control of devices. A further approach could be that once a day accumulated data is sent from the household to the energy provider. This gives the energy provider the opportunity to better calculate the needed

capacities of their power plants. For the customer this offers the possibility to directly get information about better pay scales or to let the provider dynamically change the pay scale.

ACKNOWLEDGMENT

The presented work is part of the AutHoNe project which is being partly funded by German Federal Ministry of Education and Research under grant agreement no. 01BN0702. The project is being carried out as part of the CELTIC initiative within the EUREKA framework.

REFERENCES

[1] X. Jiang, S.Dawson-Haggerty, J. Taneja, P. Dutta and D. Culler, Demo Abstract: Creating Greener Homes with IP-Based Wireless AC Energy Monitors, Proceedings of the 6th ACM conference on Embedded network sensor systems, SENSYS 2008, Pages 355-356

[2] D. R. Muñoz, D. M. Pérez, J. S. Moreno, S. C. Berga, E. C. Montero, Design and experimental verification of a smart sensor to measure the energy and power consumption in a one-phase AC line, Elsevier Journal of the International Measurement Confederation (IMEKO), Measurement 42 (2009) 412–419

[3] P. M. Jansson, J. Tisa, W. Kim, Instrument and Measurement Technology Education—A Case Study: Inexpensive Student-Designed Power Monitoring Instrument for Campus Submetering, IEEE TRANSACTIONS ON INSTRUMENTATION AND MEASUREMENT, VOL. 56, NO. 5, OCTOBER 2007

[4] [1] IEEE 802.15.4.4 – Homepage, The Institute of Electrical and Electronics Engineers, Inc., February 2007, http://www.ieee802.org/15/pub/TG4.html IEEE 802.15.2 Standard

[5] ICradio Module 2.4G Product Homepage, In-Circuit GmbH, http://www.ic-board.de/product_info.php?info=p4_ICradio-Module-2-4G.html, May 2009

[6] ADE 7753 Product Homepage, Analog Devices, Inc, http://www.analog.com/en/analog-to-digital-converters/energy-measurement/ADE7753/products/product.html, May 2009

Entwicklung eines Sensornetzwerks für den Einsatz in Schiffsmaschinenräumen

Timo Schröder[#], Tobias Pilsak[*1], Jan Luiken ter Haseborg[*2]

[*]*Technische Universität Hamburg-Harburg, Institut für Messtechnik*
Harburger Schloßstraße 20, 21079 Hamburg
[1]Pilsak@tuhh.de
[2]terHaseborg@tuhh.de

[#]*Helmut Schmidt Universität, Institut für elektrische Messtechnik*
Holstenhofweg 85, 22043 Hamburg
t.schroeder@hsu-hh.de

***Abstract*— Dieser Bericht behandelt den Einsatz eines drahtlosen Sensornetzwerkes (WSN) im Schiffsmaschinenraum. Im Maschinenraum müssen eine Vielzahl von physikalischen Größen überwacht werden. Die dafür benötigten Sensoren übertragen ihre Messwerte bislang kabelgebunden. Um eine Installation von Sensoren an schwer zugänglichen Stellen zu ermöglichen, Installationskosten zu senken oder das Nachrüsten von Sensoren zu vereinfachen, sind drahtlose Sensornetzwerke interessant.**

Im folgendem wird die verwendete Hard- und Software vorgestellt. Es folgt eine kurze Beschreibung grundlegender Multihop Routing Protokolle. Weiter werden die benötigten Eigenschaften eines Routing Protokolls für den Schiffsmaschinenraum erläutert. Das hieraus entwickelte Routing Protokoll wird präsentiert und erste Ergebnisse der Erprobung an Board eines modernen Kreuzfahrtschiffes vorgestellt.

Keywords: WSN, Routing, TinyOS, nesC, MCFA, DD

I. EINLEITUNG

Auf modernen Schiffen befinden sich bis zu 15000 Sensoren, die einen fehlerfreien Betrieb der Maschinen gewährleisten. Die Verkabelung dieser Sensoren ist ein wichtiger Kostenfaktor. Um die Kosten für die Verkabelung zu reduzieren ist der Einsatz eines drahtlosen Sensornetzwerkes denkbar. Weitere Einsatzgebiete ist die Nachrüstung von Systemen oder das Positionieren von Sensoren an nur schwer zugänglichen Orten, wie z.B. dem Pod-Antrieb, der sich mehrfach um 360° drehen kann.

Um physikalische Messgrößen zu erfassen und drahtlos zu übertragen, wird ein Funkknoten benötigt, der mindestens mit einem Sensor und einem Kommunikationsmodul ausgestattet ist. Zur Vergrößerung des abzudeckenden Bereiches werden Multihop Verbindungen (Verbindung über Zwischen-Funkknoten) verwendet. Um Datenpaket koordiniert durch das WSN zur Senke zu führen, sind die Paketpfade mittels eines Routing-Algorithmus zu definieren.

Abbildung 1: Maschinenraum eines Kreuzfahrschiffes

II. FUNKKNOTEN UND BETRIEBSSYSTEM

In dieser Arbeit wurde der Funkknoten „IRIS 2.4-GHz“ der Firma Crossbow Technology verwendet [1]. Dieser ist mit der Plattform XM2110CA, bestehend aus einem Mikrocontroller, Transceiver (IEEE 802.15.4) und Flashspeicher, ausgestattet.

Zur direkten drahtlosen Kommunikation wird das Übertragungsprotokoll IEEE 802.15.4 [2] für LR-WPAN (Low data Rate- Wireless Personal Area Network) verwendet, um eine spätere Interoperabilität zu gewahrleisten. Es wurden dazu nur die physikalische Schicht und die MAC Schicht verwendet. Die physikalische Schicht des Protokolls stellt 16 Kanäle im 2.4-GHz-ISM-Band bereit. Die Datenrate ist 250 kBit/s. Als IEEE 802.15.4-2003 kompatiblen Transceiver wird der AT86RF230 von Atmel verwendet [3].

Bei der Messdatenerfassung sind Sensoren für Temperatur, Luftdruck und Feuchtigkeit über ein 51-polige Stecker mit dem Mikrocontroller verbunden. (Abbildung 2)

Zur Programmierung des Funkknoten wurde das komponentenbasierte und ereignisgesteuerte Open-Source-Betriebssystem TinyOS 2.x (TOS) verwendet. Dessen Vorteile liegen bei der Portierbarkeit auf unterschiedlichen Hardwareplattformen und der Bibliothek an vorhandenen Basisfunktionen. Weitere Anwendung findet es im Forschungsschwerpunkt SomSed der TUHH [4].

Ein TinyOS-Programm ist als ein Konstrukt von über Schnittstellen verbundenen Komponenten zu verstehen. Dabei wird die eigentliche Anwendung und TinyOS zu einem Programm, welches den Funkknoten betreibt.

Programmiert wird im eigens für TinyOS entwickelten C-Dialekt nesC. Das komponentenbasierte nesC definiert durch die Auflösung des Programmes in einzelne elementare asynchrone und synchrone Abläufe ein Verarbeitungsmodell [5].

Abbildung 2: Iris Funkknoten mit Sensorboard

III. SZENARIO - INTERFERENZEN IM SCHIFFSMASCHINENRAUM

Es wurden Messungen des elektromagnetischen Spektrums des ISM-Bands mittels eines Spektrumanalysators frequenzauflösend im „Max-Hold"-Modus an mehreren Punkten des Maschinenraums auf dem Kreuzfahrtschiffes „Celebrity Solstice" durchgeführt (Abbildung 3).

Für die hier beschriebene Anwendung ist das 2,4-GHz-ISM-Band von besonderer Bedeutung, da dort die Funkknoten innerhalb eines Kanals ihre Daten übertragen.

Es bestand die Aufgabe, ein WSN basierend auf IEEE 802.15.4 im Schiffsmaschinenraum einzurichten. Dieses setzt hinsichtlich der Veränderung der Topologie keine hohen Ansprüche. Jedoch ist zu erwarten, dass infolge der Frequenzselektivität nicht alle Kanäle die gleiche Qualität besitzen.

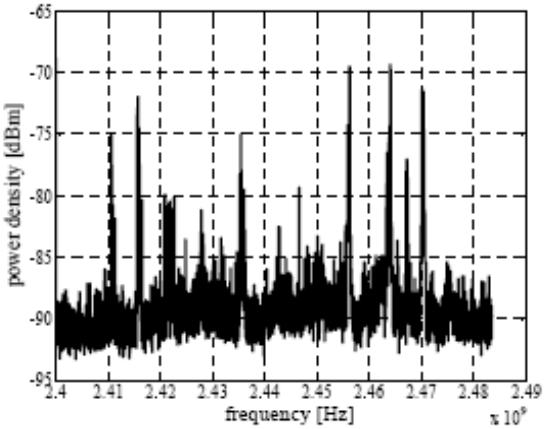


Abbildung 3: Elektromagnetisches Spektrum des ISM-Bandes im Maschinenraum

IV. MULTI-HOP ROUTING PROTOKOLL

Bei der Auswahl der notwendigen Eigenschaften eines WSN im Schiffsmaschinenraum sind mehrere bereits existierende Routing-Prokolle anhand ihrer Eigenschaften analysiert worden. Dabei haben sich vier Eigenschaften als wichtig herauskristallisiert.

a.) Die Anforderungen an eine effiziente Nutzung der Speicherressourcen, wobei die Anwendung, das Protokoll und die Treiber der Sensorik sich den Speicher teilen müssen.

b.) Eine energieeffiziente Programmierung, um eine lange Lebensdauer der Funkknoten zu gewährleisten. Dazu gehört neben der Hardwareausstattung und der Sensorik auch das Routing-Protokoll, welches das energieintensive Empfangen und Senden reduziert.

c.) Nach der Initialisierung sollte das WSN in der Lage sein mittels Selbstkonfiguration die Pakete über Hops zur Senke zu übermitteln.

d.) Das Protokoll muss robust gegenüber Veränderungen des Funkkanals sein und Übertragungsstörungen kompensieren können.

A. Prinzipien der Routing-Protokolle

Im Folgenden werden kurz die entschiedenen Eigenschaften der ausgewählten Multihop Routing Protokoll kurz beschrieben. Es wurden mehr Routing-Protokolle untersucht, jedoch werden hier nur die wichtigsten kurz beschrieben.

A.A.1 Flooding

Flooding beschreibt eine sehr einfache Methode Pakete ohne Routingtabelle, bidirektional mittels Multi Hopping zu übertragen. Das Paket wird vom Funkknoten an alle Nachbarn gesendet; als Broadcast bezeichnet. Die aktiven Funkknoten leiten dieses ebenfalls per Broadcast an deren Nachbarn weiter. Somit breitet sich das Packet wie eine Flut im gesamten Netz aus und erreicht die Senke.

A.A.2 *Tabellenbasiertes Routing*

Jeder Funkknoten im Netz besitzt entsprechend dem eigentlichen Protokoll eine oder mehrere Tabellen mit Informationen für das Routing (z.B. ID der Nachbarn, Verbindungsqualität, etc), welche nach der entsprechenden Routing-Metrik geordnet sind. Diese Tabellen beschreiben zur Übertragung den Pfad eines Paketes zur Senke.

A.A.2.1 Minimum Cost Forwarding Algorithmus

Ein einfacher unidirektionaler Routing-Algorithmus zur Minimierung des Energieverbrauches bei der Übertragung von Paketen ist der Minimum Cost Forwarding Algorithm [6] (MCFA). Dieses proaktive Verfahren leitet die Pakete entlang des, bei Initialisierung gefunden, kürzesten Pfades von Quelle zur Senke. Jeder Funkknoten speichert dazu Routing-Informationen, z.B. Adresse des nächsten Funkknotens auf dem Pfad zur Senke.

A.A.2.2 Directed Diffusion

Der reaktive Routing-Algorithmus Directed Diffusion (DD) Routing [7] beschreibt ein Verfahren zur datenzentrischen Übertragung. Das bedeutet, dass kein einzelner Funkknoten, sondern eine Gruppe von Funkknoten angesprochen wird. Es entsteht somit eine 1 zu N Beziehung zwischen einer Senke und N Funkknoten. In den Routing-Tabellen der Funkknoten sind passend zum Datentyp die Adresse des nächsten Funkknotens zur Senke eingetragen. Ein beliebiger Funkknoten kann mittels Interessenbekundung als Senke für einen Datentypen, während des Betriebes fungieren. Falls ein Eintrag zu einem Zielknoten in der Routingtabelle nicht existiert, z.B wegen abgelaufener Gültigkeitsdauer oder keinen empfangenen Interessen-Paket, leitet der Funkknoten das entsprechende Paket an alle Nachbarn weiter.

V. GEGENÜBERSTELLUNG DER ROUTING-PROTOKOLLE

Das Flooding ist hinsichtlich des überproportional hohen Datenaufkommens zur Übertragung eines Pakets für größere WSN nicht geeignet. Der aufgebaute Baum des MCFA von Routing-Pfaden ist eine sehr gut geeignet zur Paketübertragung, auch in unzugänglichen Bereichen des Schiffes. Die Idee einer Gruppenbeziehung des Directed Diffusion Protokolls bei der unidirektionalen Kommunikation zwischen Funkknoten und Senke ist geeignet, um die Kommunikation mittels Differenzierung von Messdaten im Schiffsmaschinenraum effizienter zu gestalten.

In der Zielanwendung ist davon auszugehen, dass es eine fest installierte Senke gibt, welche für die Daten verarbeitende Einheit als Gateway fungiert.

VI. ENTWICKELTES ROUTING-PROTOKOLL

Das im Rahmen dieser Arbeit entwickelte Routing-Protokoll ist somit eine Kombination aus den Eigenschaften der Routing-Protokolle MCFA und Directed Diffusion.

Die Pakete werden entlang eines bei der Initialisierung berechneten Baumes übermittelt, dessen Pfade die kürzesten Wege mit der besten Verbindungsqualität zur Senke beschreiben. Wie bereits in Kapitel III. erwähnt, werden keine sehr großen Veränderungen in der Topologie des WSN erwartet. Um trotzdem auf Veränderungen im Funkkanal reagieren zu können, kann der Baum von der Senke, während des Betriebes reinitialisiert werden. Jeder Baum, siehe Abbildung 2, steht für einen Datentyp.

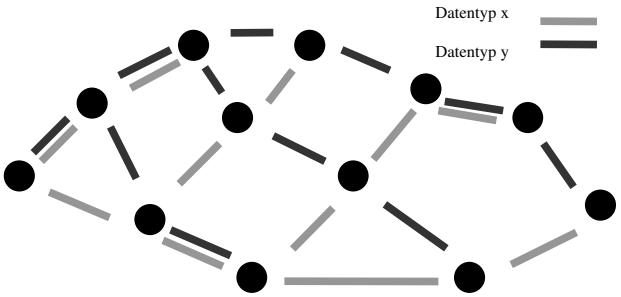


Abbildung 4: Routing-Pfade verschiedener Datentypen

VII. TESTUMGEBUNG

Der Schiffsmaschinenraum befindet sich in den unteren Decks eines Schiffes. Abbildung 1 zeigt den Maschinenraum mit zwei der vier Hauptmaschinen eines Kreuzfahrtschiffes.

Um den Ablauf zu kontrollieren, wird eine Vielzahl von Sensoren eingesetzt, die Temperaturen, Drücke, Durchflüsse oder Schalterstellungen überwachen.

Diese Sensoren wurden für die Erprobung mit einer Sensorplatine nachgebildet, welche zusammen mit dem IRIS Funkknoten in Abbildung 2 gezeigt ist. Die Sensorplatine verfügt über einen Feuchtigkeits-, Temperatur- und Luftdrucksensor.

Es wurde zur Überprüfung der Funktion des entwickelten Routing Protokolls und der Funkknoten in der Umgebung des Schiffsmaschinenraums installiert. Die Verbindungsstruktur des Netzwerkes ist in Abbildung 5 gezeigt. Die Pfeile zeigen den vom System gewählten Baumaufbau. Es ist zu erkennen, dass zwei Funkknoten mittels eines Zwischenknoten Verbindung zur Senke haben. Die Verbindungsstruktur ist natürlich auch von der Verbindungsqualität zwischen den Funkknoten abhängig, wie bereits in VI. erwähnt. Es wurde beobachtet, dass sich die Verbindungsstruktur über die Zeit leicht verändert. Es bestand beispielsweise nicht in jedem Baum eine Verbindung zwischen Funkknoten 42 und 40. Somit diesem Fall wurden Paket von Funkknoten 42 automatisch über die Funkknoten 29 und 40 zur Senke geleitet.

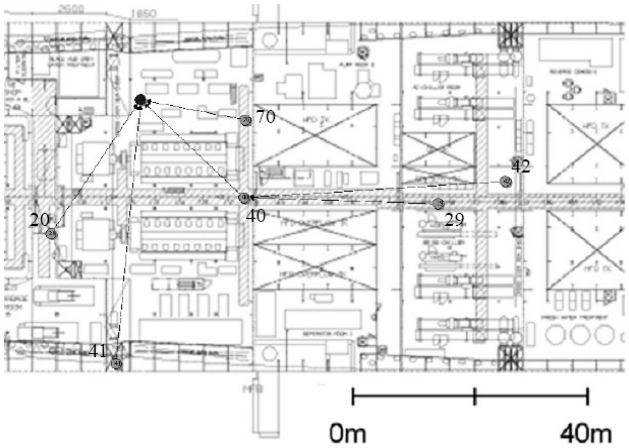


Abbildung 5: Verbindungsstruktur des Sensornetzwerkes im Schiffsmaschinenraum (dunkel: Senke, hell: Funkknoten)

Jeder Funkknoten misst physikalische Daten, wie zum Beispiel die Temperatur und übermittelt diese Daten durch den Baum zu Senke. Das gezeigte Netzwerk arbeitete während einer mehrtägigen Probefahrt. Ein PC speicherte alle Sensordaten die er von der Senke erhielt zusammen mit einem Zeitstempel und Informationen über den Topologieaufbau. Ein beispielhafter Verlauf der Umgebungstemperatur über die Zeit gemessen ist in Abbildung 5 gezeigt. Eine detailliertere Beschreibung der Messergebnisse ist in [8] zu finden.

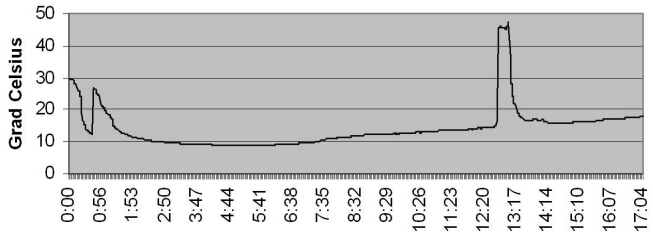


Abbildung 5: Temperaturverlauf

VIII. ZUSAMMENFASSUNG

In diesem Bericht wurden verschiedene Routing Verfahren und ein speziell für die Anwendung im Maschinenraum entwickeltes Baum-Routing Protokoll vorgestellt. Das entwickelte System wurde im Schiffsmaschinenraum erfolgreich getestet.

DANKSAGUNG

Spezieller Dank gilt der Meyer Werft in Papenburg für die freundliche Unterstützung [9].

REFERENZEN

[1] Crossbow Technology, Inc., “IRIS 2.4GHz Datasheet”, www.xbow.com

[2] IEEE Std. 802.15.4a-2007: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs), IEEE Standard for Information technology- Telecommunications and information exchange between systems- Local and metropolitan area networks- Specific requirements.

[3] ATMEL Corporation, AT86RF230 Datasheet, www.atmel.com

[4] S. Georgi, C. Weyer, M. Stemick, C. Renner, F. Hackbarth, U. Pilz, J. Eichmann, T. Pilsak, H. Sauff, L. Torres, K. Dembowski, F. Wagner, „SomSeD: An Interdisciplinary Approach for Developing Wireless Sensor Networks”, 7. GI/ITG KuVS Fachgespräch Drahtlose Sensornetze, Berlin, 2008

[5] Philip Levis, David Gay, “TinyOS Programming”, Book, Cambridge University Press; 1 edition (April 13, 2009)

[6] Fan Ye, Alvin Chen, Songwu Lu, Lixia Zhang, “A scalable solution to minimum cost forwarding in large sensor networks”, UCLA Computer Science Department, Los Angeles, CA 90095-1596

[7] Chalermek Intanagonwiwat, Ramesh Govindan, Deborah Estrin “Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Network”

[8] “Somsed on Ships”, SomSed Workshops, Hamburg, Germany, unpublished.

[9] Meyer Werft in Papenburg, www.meyer-werft.de

IEEE 1451.0 Sensor Interoperability Experiment

Jesper Zedlitz
Christian-Albrechts-University
Kiel, Germany
Email: jze@informatik.uni-kiel.de

Norbert Luttenberger
Christian-Albrechts-University
Kiel, Germany
Email: nl@informatik.uni-kiel.de

I. Motivation

Interoperability is the ability of two or more heterogeneous systems to exchange information and use this data in a reasonable way. For oceanographic data acquisition several approaches exist, how interoperability between different sensor systems can be achieved. A promising interoperability protocol for sensor networks is IEEE 1451.0. Up to now no practical experience exists in using the protocol in the field.

To truly evaluate the use of an interoperability standard it is necessary to really implement it in a demonstration with numerous involved parties and different software implementations of the specification. Only then it is possible to find hidden ambiguities and obstacles that hinder the data exchange in a given specialist area.

An interoperability experiment together with MARUM Bremen, Universitat Politècnica de Cataluny, Monterey Bay Aquarium Research Institute (MBARI) and National Institute of Standards and Technology (NIST) should clarify to what degree IEEE 1451.0 is applicable for oceanographic observations. At the end of the experiment a live demonstration of the system was presented at the Ocean Innovations Workshop in Saint Johns in October 2008. The parties involved in the experiment are shown in figure 1.

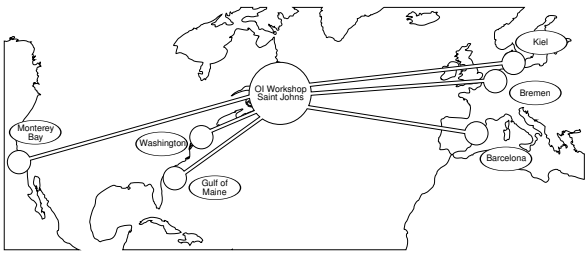


Fig. 1. Parties involved in the interoperability experiment

II. IEEE 1451 Protocol Family

IEEE 1451 is a family of interface standards. It is developed with the aim to access transducer (sensor/actor) data through common open and network-independent communication interfaces. For the user it should not be noticeable whether the transducers are connected directly to a system or accessed via networks – wired or wireless. A key component is the so called *Transducer Electronic Data Sheet* (TEDS) that contains meta data about the transducer including device identification, manufacturer information, calibration curves, measurement ranges etc.

IEEE 1451.0[1] defines common operations for transducers and also protocols how to access transducer data via network using the HTTP. IEEE 1451.0 uses a client/server model. A *Network Capable Application Processor* (NCAP) with one or more *Transducer Interface Modules* (TIM) – these are sensors or actors – attached takes the role of the server. Clients can send requests to the NCAP to retrieve measured values. The server answers with XML documents (figure 2). As an alternative a simpler ASCII format for responses is specified in IEEE 1451.0.

```
<?xml version="1.0" encoding="utf-8" ?>
<TransducerDiscoveryHTTPResponse>
   <errorCode>
      0
   </errorCode>
   <ncapId>
      1
   </ncapId>
   <timId>
      3
   </timId>
   <numberOfChannels>
      3
   </numberOfChannels>
   <channelIds>
      1,2,3
   </channelIds>
   <transducerNames>
      pressure,
      temperature,
      conductivity
   </transducerNames>
</TransducerDiscoveryHTTPResponse>
```

Fig. 2. XML document containing the response of a TransducerDiscovery query

In addition to the IEEE 1451.0 standard the IEEE 1451 family also contains other specifications that describe the communication of transducers connected directly to the NCAP:

- **IEEE 1451.2** – point-to-point connections including RS232, I^2C and USB
- **IEEE 1451.3** – multi-point connections
- **IEEE 1451.5** – wireless communication including IEEE 802.11 (WiFi), IEEE 802.15.1 (Bluetooth) and IEEE 802.15.4 (ZigBee)
- **IEEE 1451.6** – CANopen network interface
- **IEEE 1451.7** – Radio Frequency Identification (RFID) systems

Some of these specifications are still under development or revision. Therefore they were not part of the interoperability experiment in which we only tested the communication between IEEE 1451.0 capable nodes. In the context of this experiment five operations have been evaluated (figure 3).

Module	Operation	Parameters	Task
TimDiscovery	reportTims	NCAP id TIM id	*list of sensors attached to the NCAP*
	reportChannels	NCAP id TIM id Number of channels Channel id Channel name	*list of channels of one sensor*
TransducerAccess	readData	NCAP id TIM id Channel id Transducer data	*read the current measurement value of a channel*
TedsManager	readTeds *(GEO TEDS)*	NCAP id TIM id TEDS type *(=14)* TEDS data	*read the geographic position (latitude, longitude, height) of the sensor system*
	readTeds *(Meta−Id TEDS)*	NCAP id TIM id TEDS type *(=2)* TEDS data	*read vendor, model, version, serial number, description of the sensor system*

Fig. 3. The five operations from IEEE 1451.0 used in the experiment

III. IEEE 1451.0 Server

For the experiment a Java implementation of an IEEE 1451.0 server has been developed in Kiel. It receives IEEE 1451.0 requests via HTTP from clients. If necessary it triggers attached sensors to perform measurements and read the results. Other kinds of sensors constantly report values to the NCAP – in that case it is only necessary to select the latest values. The measured values are prepared and returned to the client in IEEE 1451.0 format. An overview of the system in Kiel is shown in figure 4. Attached to the central NCAP are three TIMs having several *channels* each. The weather station at IfM-Geomar, the weather station at Kiel lighthouse and measured values from a CTD[1] were used as data sources.

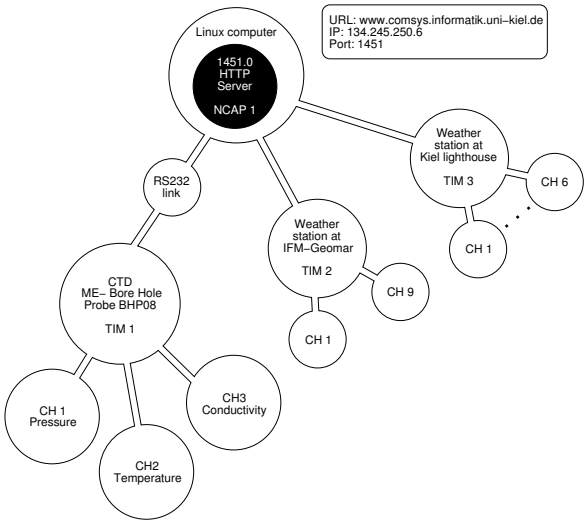


Fig. 4. Overview of the IEEE 1451.0 system in Kiel

[1]a oceanographic instrument measuring conductivity, temperature and depth/pressure

The Java source code for the server developed in Kiel was also used by the partners in Bremen and Barcelona as basis for their software development. They wrote their own code for the communication between NCAP and instruments and the conversion of measured values into IEEE 1451.0 format. MBARI on the other hand used a completely own implementation including the IEEE 1451.0 command interpreter. These parallel developments made it possible to compare different servers and to find and fix software bugs but also helped finding ambiguous text passages in the IEEE 1451.0 specification.

Several IEEE 1451.0 clients written by the participants of the experiment were used to query the various IEEE 1451.0 servers. A simple but quite useful client with a graphical user interface (fig. 5) has been developed in Kiel. It was used frequently during the experiment to check the responses from the servers.

Fig. 5. GUI of the IEEE 1451.0 client

IV. Results of the Experiment

During the experiment, several flaws of IEEE 1451.0 became visible. It appeared that further studies are necessary, for example on how to deal with low bandwidth links and links with discontinued operation – both limitations of satellite links, which are often used in the communication with oceanographic instruments. Other problematic points are the treatment of units of measurement and timestamps for measurements.

This shows how important interoperability experiments are: They help to assess the usability of specifications for the in-the-field use in a given specialist area.

References

[1] IEEE STD 1451.0-2007, Standard for a Smart Transducer Interface for Sensors and Actuators – Common Functions, Communication Protocols, and Transducer Electronic Data Sheet (TEDS) Formats, IEEE Instrumentation and Measurement Society, TC-9, The Institute of Electrical and Electronics Engineers, Inc., New York, N.Y. 10016, SH99684, October 5, 2007.