Power monitoring and testing in Wireless Sensor Network Development

Matthias Woehrle, Jan Beutel, Roman Lim, Mustafa Yuecel, Lothar Thiele
Computer Engineering and Networks Lab, ETH Zurich,
8092 Zurich, Switzerland
firstname.lastname@tik.ee.ethz.ch

***Abstract*—Wireless Sensor Networks (WSNs) are becoming widely used in scientific applications as they allow for instrumenting the environment in previously impossible ways revealing unprecedented insight and detailed data. Long term monitoring of environmental processes and the harsh reality of remote and inaccessible terrain demand longevity from such WSN deployments. While functional correctness is typically the developers prime concern, unattended operation without the possibility to scavenge energy nor to exchange batteries demand applications to be significantly optimized for power consumption. Thus, power consumption of the application becomes a chief concern of system design where development necessitates appropriate means for monitoring, profiling and analysis. We propose the monitoring and testing of power consumption in an integrated testing infrastructure. We additionally present a novel methodology for automatic power profile testing for embedded systems in the context of WSNs.**

I. Introduction

Environmental monitoring uses WSN technology, since it allows for tether-less, minimally invasive instrumentation of phenomena under observation. Typical deployments for environmental monitoring require intricate and expensive installations often on remote sites or terrain which is not easily accessed such as a mountain. Such deployments must provide a considerable network lifetime. Apart from the functional correctness of the application and communication intricacies, the biggest challenge for WSN applications is to provide guarantees on the longevity of the system by elaborate optimization and validation of the power consumption of individual nodes.

In the TinyOS architecture, the Null application is used to check the lowest power mode on a given platform. However, as you can see in Fig. 1, the currently measured current for Null on a TinyNode is around 1 mA, while in a data collection application designed for the TinyNode, the lowest power modes achieved is substantially less. This is not just a singular problem but actually a frequent complaint on the TinyOS mailing list which can only be detected by appropriate power profiling methods. Different results of measurements can be attributed to detrimental code modifications or a slightly differing tool chain also resulting in differing binary code images. In a recent discussion it was reported that even using the older code base certain nodes were yielding a value greatly differing from the targeted power consumption. The sleep current was magnitudes off the typical value. Since these were specific nodes, a hardware change was assumed and the manufacturer was contacted. A detailed analysis revealed that

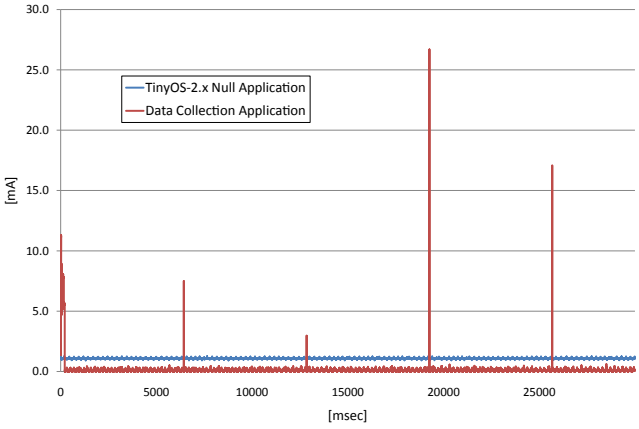

Figure 1. Low power data application versus Null on a TinyNode: Null is not initialized properly to transition into the lowest power state possible. The low power data application however enters a low power sleep state.

in fact no hardware change (schematic change) had occurred but a minor device variation in the MSP430 processor used, possibly due to the revving of the silicon at the vendors side, had lead to reduced performance (possibly even malfunction) with the current implementation of the driver used. This is not the only example of device variations impacting system performance as can be seen in an excerpt from the TinyOS-Devel mailing list [1]:

>> I just tested some almost unused MicaZ motes and they show the
>> expected low power consumption (2uA measured with the Extech).
>> The one that shows high consumption was used in a real deployment
>> for several months. I also tested 6 other motes from the same
>> deployment and they also show a low power consumption. [· · ·]
>>Here are the tests using the McuPowerOverride. I used 3 good
>> MicaZ (G1-G3) and the bad one (B). The results are:

>>	G1	G2	G3	B
>> IDLE	3.53	3.46	3.63	3.88
>> ADC_NR	1.15	1.13	1.18	1.36
>> EXT_STANDBY	0.12	0.12	0.12	0.29
>> POWER_SAVE	2.6u	1.9u	2.2u	0.16
>> STANDBY	0.12	0.12	0.12	0.29

>> All the values are in mA except for POWER_SAVE for G1-G3.

Had this node actually gone bad over the years? Or was it due to the environment it had been subjected too? The error

sources are hard to pinpoint in such a complex system where the observability is very limited. Both cases show that the application performance can change although the developer assumes that there has not been any change. Thus, detailed profiling the power consumption at different design times is of utmost importance for applications relying on low power consumption.

To this end, we argue for testing and monitoring power consumption during the development process. As deploying a correct system focussing on vital non-functional properties, i. e. power consumption, is of utmost importance, it is vital to integrate power consumption or current draw[1] respectively, into the testing process. This integration has substantial benefits. Long-term trending allows the analysis of power consumption and identifying the effects of code changes on power consumption. Power unit tests based on power traces allow for verifying that the power consumption of a tested system is in accordance with its specification at any instant in time. The integration of power consumption and logical traces allows for enhanced modeling of WSN nodes.

We provide in this paper the following contributions:

- We present an architecture for monitoring power consumption for WSN development,
- We present methods for the automated analysis of power consumption and its use in integration testing,
- We present results of power profiling in WSN application development.

II. Related Work

A. Energy-focussed Simulation and Testing

Test platforms for WSNs have so far concentrated on simulation on different abstraction levels and testbeds:

A prominent tool for simulation is TOSSIM [2], an event-driven simulator for TinyOS [3] platforms, which replaces components accessing hardware with simulated components. An extension to simulation is tracking the power state of each component during the simulation as performed in PowerTOSSIM [4]. Instrumentation of the simulated hardware components and using a additional power tracker, PowerTOSSIM calculates power consumption using measured current draw from Mica2 nodes as well as providing the execution time of components in cycles. Back-end tools are provided focussing on analysis and on data visualization.

In [4], the authors also describe in-situ measurements on a testbed: Motelab [5] provides in situ power measurement for a single node (Mote 118) using a digital multimeter. The timestamped data is available for users in a log file (e. g. for experimental validation). No further analysis support is provided. A tool like SPOT [6], which allows for in-situ measurement of sensor node power and energy, allows for large scale instrumentation of nodes in testbeds allowing for extensive distributed power analysis and testing. Our work provides the groundwork for a distributed, global approach for testing power consumption.

[1]We use power consumption and current draw interchangeably. When assuming a nearly constant supply voltage considering a single test execution, the power consumption is only dependent on the current draw.

AEON [7] provides accurate power consumption prediction for WSN nodes. It is built on top of Avrora [8], a simulation and analysis toolbox for programs written for the AVR microcontroller produced by Atmel and the Mica2, and often used for instruction-level simulation. AEON extends Avrora with an energy model, similar to PowerTOSSIM's additions to TOSSIM and also uses previous measurements of current draw for hardware characterization of the individual components.

Dunkels et al. [9] discuss on-line energy monitoring for WSN nodes in the Contiki operating system. The authors instrument device drivers to timestamp activation and deactivation of components and multiply the on-time with an empirically determined current draw.

All these tools allow for modeling power consumption on various abstraction levels and allow for analysis of the power consumption or energy consumption of software on an individual test. Our approach extends these available tools, since we focus on the testing and automated analysis of specific power consumption profiles of given executions on real hardware. We focus in our work on using a realistic test platform, i. e. a testbed. However, the power unit tests presented in the following is applicable for any test platform and could be used with any of these tools, e. g. in the context of an automated, multi test platform framework [11].

B. Background: Continuous Integration

Continuous integration (CI) [12] is a methodology, which promotes frequent integration, i. e. after each code check-in, in order to provide rapid feedback to developers. This facilitates identifying software defects, typically subject to recent changes. Up to this point, CI focusses on the integration of enterprise scale software projects designed by large teams and is a common and well known methodology, e.g. in agile development [13]. For integration of software in a team, CI allows for communication with the code repository, a build tool-chain to compile the software, software analysis tools and a test platform running unit tests such as JUnit for Java projects. The overall status of the project as well as the details of all associated builds are presented on a webpage. Regular builds in CI are either triggered upon code changes, by user requests or periodically. Builds are referenced to a specific version of the code base in the repository using a unique build id.

III. The Power Monitoring Architecture

A. Testing Architecture

Our approach for testing WSN software combines established methods from software engineering combined with WSN-specific tools, i. e. the execution of the software on the target platform in a realistic environment and the profiling of power consumption by means of node current draw measurements. This is established by integrating a testbed with measurement devices for power consumption into an off-the-shelf continuous integration framework. In this setup, we integrate the Deployment Support Network (DSN) [10] with CruiseControl (CC) [14], a prominent open-source CI

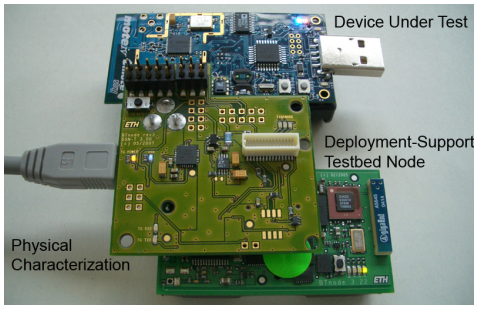

Figure 2. The testbed setup consists of a monitoring node, an interface board for power measurements, a supply and the DUT.

framework. Power measurements are performed with an off-the-shelf power analyzer, an Agilent N6705A providing ethernet connection. By using a testbed and realistic deployments scenarios, many factors that cannot be captured in simulation and analysis methods, can be used to narrow in on a realistic sensor network deployment and simultaneously extracting behavioral trace data from the system under test.

Test jobs consist of a compilation process that is handled by the CI framework: (i) job formation with subsequent submission to a testbed, (ii) distribution of code to target devices, (iii) synchronous start of all target devices and (iv) log file and power measurements collection. Depending on the context of the test, analysis can take place online, e.g. for the monitoring of operation or in more detail offline after completion of a test job.

Through the integrated approach, execution is greatly simplified and data from all test jobs is logged in a repository that references the actual software code version under test. This assures a maximum of transparency and the ability for a comprehensive post-execution analysis and evaluation.

B. Physical Parameter Extraction

For the physical characterization of motes, we employ two different approaches: On the one hand we observe long-term trends to determine the development process effects on the characteristics. With detailed snapshots of individual software builds, we perform an in-depth analysis allowing for regression testing of physical parameters.

1) Long-Term Trending: Current testbeds only have means of profiling power consumption on selected nodes [5]. A supervision of all nodes under test is mandatory for assuring reliable operation and sufficient test coverage. We are using a combination of a DSN [10] node pair with a custom power monitoring board (cf. Figure 2) that uses the internal ADC on the ATmega128l and a current sense amplifier in combination with the network logging tool Cacti. The device under test (DUT) can be powered from different sources (battery, line power) and the power status is sampled on request, e. g. currently once every 5 minutes. Although limited in precision and accuracy, basic long term trending with coarse granularity is very helpful for testbed supervision on long test sequences, giving an initial impression.

2) Detailed Physical Parameter Characterization: For detailed characterization of the system performance and especially to pinpoint specific behavior detailed traces incorporating the variation of the power consumption are a vital resource. In order to characterize device variations, but also to understand the interplay of the power supply (battery, regulated power, solar) and the system under varying load conditions, instrumentation with a fine resolution, e. g. > 1000 samples/sec, is required. Since such equipment is both costly and bulky it is currently not feasible to instrument every node in every testbed. With an approach like the SPOT, extensive node measurements could be integrated into our approach. For the moment, we instrument only a subset of nodes for detailed analysis and automate the process using an interface to CI.

3) Presentation: For each build and its associated tests, all data generated, logs, build artifacts and test results are stored in a central data structure available through a web based reporting interface [15]. The graphical reporting interface helps to present an overview of the most critical aspects from the wealth of information and contexts. Current consumption plots (Fig. 1) and average power consumption history (Fig. 3) are invaluable to analyze the application's behavior during a test. Long term trending information allows for following the evolution of the average power consumption of an application over the development process.

As an example consider Harvester [16], a typical WSN application running on TinyOS 2 (T2). It collects sensor data to monitor the environmental conditions and forwards the data to one or multiple base stations acting as gateways. Harvester is designed for long-lasting deployments, thus critically optimizing power consumption by using low-power listening and an adapted TinyOS Collection Tree Protocol. Harvester was designed particularly for Tmote Sky sensor nodes with a Chipcon CC2420 radio. Harvester is in its final development stage, where in-depth optimizations try to minimize power consumption wherever possible.

Figure 3 shows collected power consumption data over the development cycle. The continuous integration allows for tracking power consumption changes along the timeline. Thus, effects of bug fixes (in rectangles) and optimizations (in circles) are easily trackable and analyzable. Change 2 on the 22nd of December introduced a better estimation of the clock drift, resulting in a considerable decrease in power consumption, while a functional enhancement of adding real-valued sensor values and a fix for the Sensirion sensor driver resulted in an increase power consumption on the 9th of January (change 4). Harvester is tested for each new build on a small network of a base station and two sensing nodes. One of the sensing nodes is monitored. The measurement period is currently 5 minutes. It includes start up conditions, since the initialization phase is an important part in its use for building monitoring. The mean current consumption over the whole monitoring window is determined and presented for each build.

Another feature provided by our testing architecture is the ability for parameterized application projects, where the application is automatically built, tested and profiled for a set of differing parameter settings. In the particular example of Harvester, different values for the sleep cycle parameter are set. For each of the parameter, the average power consumption

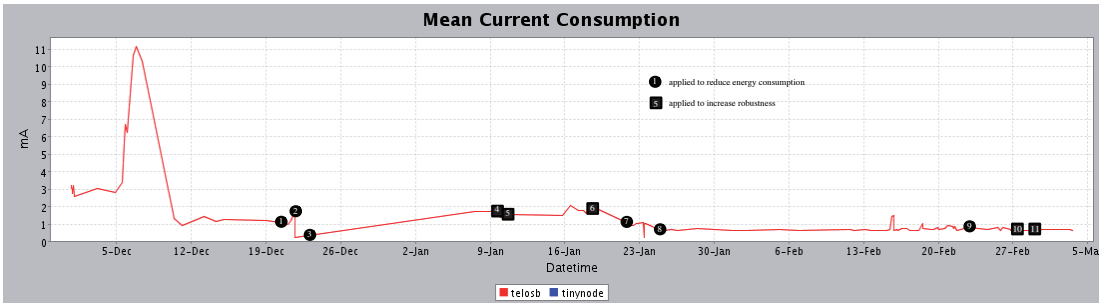

Figure 3. Harvester Evolution. Each mark indicates a code change. Code changes are either functional fixes or protocol enhancements.

for a test is recorded and its history displayed as shown in Figure 4. This allows for tracking and comparing the varying impact of changes on the parametrized application.

IV. Testing for Power Consumption

When trying to verify a comprehensive system, a test can nevertheless focus on an individual aspect of the execution. Focussing on the power consumption of the system or a part thereof such as a single node, *power unit tests* can be devised to determine if the power consumption, which is monitored during execution, is acceptable for the given test. While in standard unit testing, the test is performed on individual functional units, the power unit test is performed focussing on an individual physical parameter.

Power consumption is a crucial metric for sensor nodes, since energy-efficiency is of utmost importance for battery-operated motes. However it is not sufficient to merely look at average values. Sleep states and duty cycle patterns require intricate analysis of detailed power consumption traces. Employing our testing infrastructure, we can formulate power unit tests on the node measurements.

A. Power Unit Tests

The formulation of a power unit test is based on a testcase execution for a given application. The testcase is deterministic and allows for formulating a representative reference function. Accounting for noise on measurements and variations in hardware or environmental conditions, we determine a set of bounds for a given reference function. These bounds allows for formulating a boolean valued checker that can be processed in the context of a larger testing framework by providing a single pass or fail result [11].

A reference may be determined in different ways: using a model, e. g. derived from a specification, and simulating the test case on the model or by using a golden measurement of the testcase on the target device. Reference functions differ in level of detail and accuracy, e. g. physical characterization of the sensor nodes may be incomplete.

1) Bounds for Power Unit Tests: The checking of a test measurement against a reference must allow for some variation in measurements. For hardware differences due to manufacturing variations or for differing temperature levels, we allow an offset of the range values in a specified interval. In order to compare a reference to an observation, we need to allow for a temporal shift of the reference function in time in a given interval due to test run start and stop time variances. To this end, we perform a least square analysis of the difference between the current observation and the reference trace. We compute a best fitting reference to determine shift and offset to the test specific reference in the specified intervals.

Starting point for our power unit test of the power trace is the shifted reference start. The checking of the power consumption occurs after the reference start to be able to include a test initialization phase, not considering transient effects and power-on or boot effects.

In order to check the power trace, the determined reference allows for determining an upper and a lower bound, which are the boundaries an acceptance region. An error condition is asserted, if a window of consecutive readings lies outside this region. A small number of readings outside the acceptance region may be a spurious measurement device artifact. In the following we give a formal definition of the power unit test, i. e. the reference function and the upper and lower bound of the acceptance region and how they are derived for a given test scenario.

Definition 4.1: A *reference function* is a function $f : \mathbb{R} \to \mathbb{R}$ of time t of a measured physical quantity, e.g. current dissipation. $f(t)$ is a piecewise, typically discontinuous function, which is composed of sub-functions $f_i : \mathbb{R} \to \mathbb{R}$ with discontinuities $t_i \in \mathbb{R}$, $i \in \mathbb{N}$ at the sub-interval boundaries of f.

This reference function is hulled by two bounding functions (f^+, f^-), which are the boundaries of the acceptance region. In order to compute these bounding functions, we first determine intermediate upper and lower bound functions $f_{y,i}^{(+/-)}$, which only account for the variance in value in each interval, but do not consider the sub-interval boundaries t_i.

$$f_{y,i}^{-}(t) = \begin{cases} f_i(t) - \Delta y_i^- & \text{if } t \in [t_{i-1}, t_i) \\ 0 & \text{if } t \notin [t_{i-1}, t_i) \end{cases}$$

The upper bound $f_{y,i}^{+}$ follows accordingly with adding bound values Δy_i^+. Variable bounds per interval allow for different granular checking. A transmitting node may have larger variation in its power than a node which is sleeping. Additionally, we account for uncertainties in time with a symmetric variability in time Δt around the discontinuities t_i. Thus, the *lower bound* of a reference function is defined as:

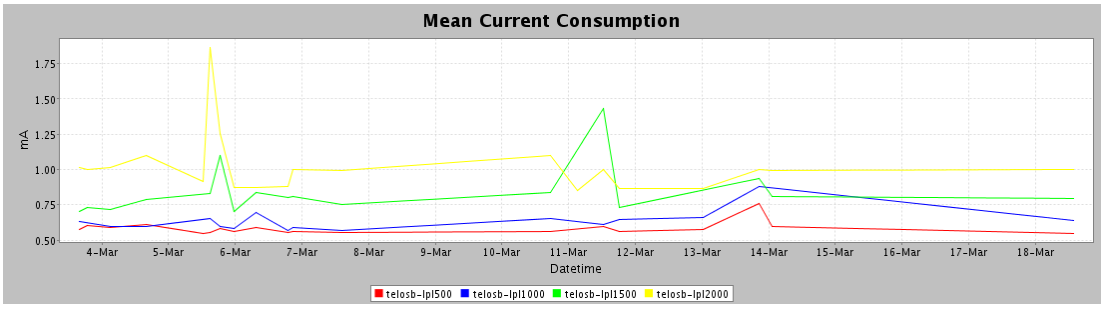

Figure 4. Parameterized Harvester Project. Application changes have differing effects for the different parameter settings.

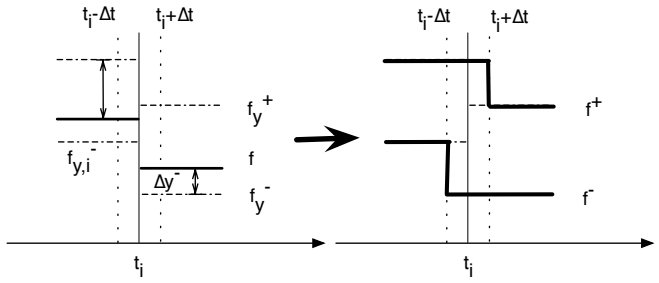

Figure 5. Illustration of bounds derivation of the acceptance region given a reference function.

$$\forall \tilde{t} \in [-\Delta t, \Delta t], i \in \mathbb{N}:$$
$$f^-(t+\tilde{t}) = \begin{cases} f_y^-(t_i^-) & \text{if } -f_y^-(t_i^-) + f_y^-(t_i^+)) \geq 0 \\ f_y^-(t_i^+) & \text{if } -f_y^-(t_i^-) + f_y^-(t_i^+)) < 0 \end{cases}$$

The *upper bound* follows accordingly:

$$\forall \tilde{t} \in [-\Delta t, \Delta t], i \in \mathbb{N}:$$
$$f^+(t+\tilde{t}) = \begin{cases} f_y^+(t_i^-) & \text{if } -f_y^+(t_i^-) + f_y^+(t_i^+)) \leq 0 \\ f_y^+(t_i^+) & \text{if } -f_y^+(t_i^-) + f_y^+(t_i^+)) > 0 \end{cases}$$

Depending on the function value in the neighboring sub-interval in $[-\Delta t, +\Delta t]$ around each discontinuity t_i, either the function value approaching from the left (t_i^-) or the right (t_i^+) is chosen.

Figure 5 illustrates the process of generating the bounds for a power unit test given intervals of the reference function.

2) Implementation Example: In the following, we look at the TinyOS Application MultihopOscilloscope. It is a multi-hop data collection application available in the TinyOS 2 distribution, which is regularly built on our test infrastructure.

The reference is described using an XML specification as depicted in Listing 1. It specifies the start and stop time of the reference checking. The reference function is specified using individual data points to define linear segments in the intervals. For MultihopOscilloscope, the period of the reference function is $1s$, divided into 2 distinct segments: (1) for radio idle and (2) for transmitting. We define a global time variance for a reference function of $\Delta t = 0.05s$. We use a variable bound for the reference function of $\Delta y_{1/2^+} = 1.2mA$ and $\Delta y_{1^-} = 0.5mA$ and $\Delta y_{2^-} = 1.2mA$.

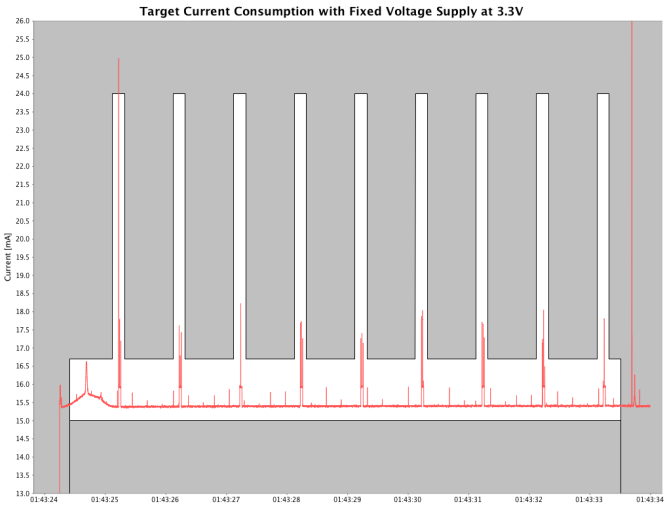

Figure 6. Test data from TinyOS 2.x MultihopOscilloscope on TinyNode: The white area denotes the reference bounds.

```
<referenceTrace name='MultihopOscilloscope'>
   <start>0.4</start>
   <stop>9.5</stop>
   <xVariance>0.05</xVariance>
   <period>1.0</period>
   <points>
      <point>
         <time>0.0</time>
         <value>15.5</value>
         <yVarianceMinus>0.5</yVarianceMinus>
         <yVariancePlus>1.2</yVariancePlus>
      </point>
      ...
   </points>
</referenceTrace>
```

Listing 1. XML specification for MultihopOscilloscope

The resulting bound for the reference curve is displayed in Fig. 6. It also displays the start and the stop time at $0.4s$ and $9.5s$ respectively. Although a single reading is outside the reference bound, no error is asserted, as a single outlier may also be attributed to a measurement device artifact.

B. Power Modeling

In order to determine the reference function of a given system, we model the power consumption of a WSN node as a composition of components with a given power consumption. Total power consumption is determined by the constituent components $C_i, i \in \{1, \ldots, m\}$. Each component C_i has a

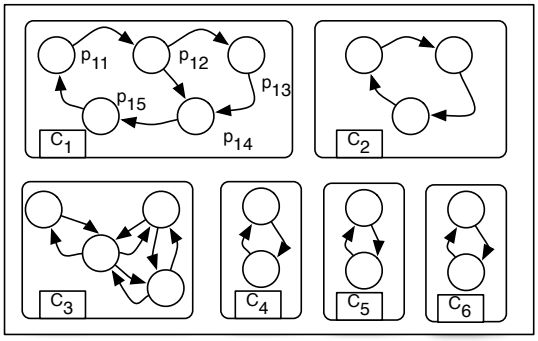

Figure 7. Model of a node for Power consumption. Component C displays a model for a radio component with 4 states: Off, On, Tx, Rx. Component E is a simple LED.

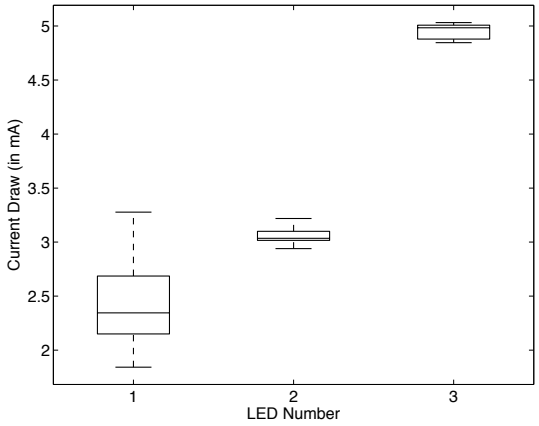

Figure 8. Boxplot of the results for the analysis of ten different Tmote Sky Sensor nodes.

set of power states $S_i = \{s_{i,1}, \ldots, s_{i,n}\}$. Figure 7 displays 6 components, and their corresponding set of power states.

The power consumption of a component is a function of the state to a real value: $P_i : S_i \rightarrow R$. The total power consumption P is the composition of the power consumption of the individual components:

$$P = \sum_{i=1}^{m} P_i(s_{i,j})$$

In this model, transients due to capacitive loading or discharge are ignored.

The model parameters can be derived using a linear model (lm) approach:

$$Y = X\beta + \epsilon,$$

where Y is the power consumption measurement, x_j is the predictor for a given state of one of the components and β describes the contribution of a a particular component state on the power consumption.

Given an execution of the system and synchronized logging data of power consumption (Y) and functional instrumentation data X, we can determine the parameters for the linear model of the power consumption where each test execution is a time series of observations.

This model is independent of the actual application running on the node; the predictors only concern the actual hardware. Different applications and observations may be used in order to fit the model. We instrumented the Tmote Sky Platform and measure the power consumption with an Agilent N6705A Power Analyzer. Synchronized state information of the radio, is available by exposing internal state via connector pins and measuring voltage in parallel to the current consumption of the sensor node on the four available slots.

However, a linear model is based on the assumption that different observations are uncorrelated. This is not the case for our current measurements, since these are time series data. We could use a generalized least squares (gls) approach. An analysis of the autocorrelation of the time series shows an autoregressive process $AR(k)$ of an order $k \leq 6$, which renders gls computationally extremely expensive. In this work, we choose the linear model and accept the simplification error.

Using the linear model, we can determine the predictors using R [17]. Figure 8 shows the boxplots for LED models based on measurements on ten different nodes. The boxplot shows, that the current draw for the LEDs is fairly stable across the different nodes. However one node in our measurement considerably differs in its current draw for LED1.

The model of the LEDs serves to generate the reference curve for the TinyOS 2 Blink application, which is continually built on our testing infrastructure. Thus a power unit test can be formulated as described above.

V. Summary and Outlook

In this work, we have presented a new testing architecture for power consumption of WSN nodes. The integrated platform allows for novel approaches for testing WSNs. The presented power unit tests allows for integrating tests focussing on power consumption into an automated build process.

Widespread integration of power and energy monitoring tools such as the SPOT result in a wealth of execution information, which requires novel tools and methodologies for modeling, testing and analysis. The testing architecture and power unit testing are first efforts in this direction.

VI. Acknowledgements

The work presented here was supported by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant number 5005-67322. We kindly acknowledge the support by Anne Auger and Tim Hohm.

References

[1] R. Musaloiu-E. *et al.*, "[tinyos-devel] more bugging about current consumption on micaz in tinyos-2.x," http://mail.millennium.berkeley.edu/pipermail/tinyos-devel/2006-July/001250.html, July 2006.

[2] P. Levis, N. Lee, M. Welsh, and D. Culler, "TOSSIM: Accurate and scalable simulation of entire TinyOS applications," in *Proc. 1st ACM Conf. Embedded Networked Sensor Systems (SenSys 2003)*, Nov. 2003, pp. 126–137.

[3] TinyOS, "http://tinyos.net/." [Online]. Available: http://tinyos.net/

[4] V. Shnayder, M. Hempstead, B. rong Chen, G. W. Allen, and M. Welsh, "Simulating the power consumption of large-scale sensor network applications," in *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, 2004, pp. 188–200.

[5] G. Werner-Allen, P. Swieskowski, and M. Welsh, "MoteLab: A wireless sensor network testbed," in *Proc. 4th Int'l Conf. Information Processing Sensor Networks (IPSN '05)*, Apr. 2005, pp. 483–488.

[6] X. Jiang, P. Dutta, D. Culler, and I. Stoica, "Micro power meter for energy monitoring of wireless sensor networks at scale," in *IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks*. New York, NY, USA: ACM, 2007, pp. 186–195.

[7] O. Landsiedel, K. Wehrle, and S. Gotz, "Accurate prediction of power consumption in sensor networks," in *Proc. 2nd IEEE Workshop on Embedded Networked Sensors (EmNetS-II)*. IEEE Computer Society, 2005, pp. 37–44.

[8] B. L. Titzer, D. K. Lee, and J. Palsberg, "Avrora: scalable sensor network simulation with precise timing," in *Proc. 4th Int'l Conf. Information Processing Sensor Networks (IPSN '05)*, 2005, p. 67.

[9] A. Dunkels, F. Osterlind, N. Tsiftes, and Z. He, "Software-based on-line energy estimation for sensor nodes," in *EmNets '07: Proceedings of the 4th workshop on Embedded networked sensors*, 2007, pp. 28–32.

[10] M. Dyer, J. Beutel, L. Thiele, T. Kalt, P. Oehen, K. Martin, and P. Blum, "Deployment support network - a toolkit for the development of WSNs," in *Proc. 4th European Workshop on Sensor Networks (EWSN 2007)*, 2007, pp. 195–211.

[11] M. Woehrle, C. Plessl, J. Beutel, and L. Thiele, "Increasing the reliability of wireless sensor networks with a distributed testing framework," in *Proc. 4th IEEE Workshop on Embedded Networked Sensors (EmNetS-IV)*, June 2007.

[12] A. G. Paul Duvall, Steve Matyas, *Continuous Integration: Improving Software Quality and Reducing Risk*. Addison-Wesley, 2007.

[13] D. Cohen, M. Lindvall, and P. Costa, "An introduction to agile methods." *Advances in Computers*, vol. 62, pp. 2–67, 2004.

[14] CruiseControl, "Cruisecontrol home," April 2008. [Online]. Available: http://cruisecontrol.sourceforge.net/

[15] Computer Engineering and Networks Lab - ETH Zürich, "Cruisecontrol at tik42x.ee.ethz.ch," April 2008. [Online]. Available: http://tik42x.ee.ethz.ch:8080/

[16] ——, "Harvester in tinyos 2 contrib," 2008. [Online]. Available: http://tinyos.cvs.sourceforge.net/tinyos/tinyos-2.x-contrib/ethz/snpk/apps/

[17] R Development Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2008, ISBN 3-900051-07-0. [Online]. Available: http://www.R-project.org