Energy-aware Reconfiguration of Sensor Nodes

Andreas Weissel and Simon Kellner

University of Erlangen-Nuremberg
Distributed Systems and Operating Systems
{weissel,kellner}@cs.fau.de

Abstract. Energy consumption is one of the most challenging constraints for the design and implementation of sensor networks. As sensor nodes are commonly battery-driven, the way the available energy is used determines the lifetime of the system. In recent research, a variety of approaches has been investigated to predict energy consumption off-line using simulation, emulation, code analysis and energy models of hardware components. However, for a wide range of applications the run-time behavior is dictated by sensor measurements or external events, allowing only a worst case analysis before deployment.
In this paper, we investigate different approaches to derive energy characteristics during run-time and to adapt or reconfigure the applications running on individual sensor nodes. We discuss prototype implementations based on TinyOS for the BTnode architecture and their overhead regarding energy consumption and code size.

1 Introduction

To address the issue of energy-efficiency, sensor nodes are equipped with low power hardware components. For instance, the Atmel ATmega128 microprocessor found in many sensor nodes features six different sleep modes with reduced power consumption. The energy consumption of a sensor node running a specific application is usually estimated off-line by integrating energy models of system components into simulation or emulation environments. With this information, the application can be tailored to the requirements on system lifetime. However, application scenarios can be envisioned where this approach can lead to inefficient results as the run-time energy consumption is hard to predict.

The following piece of pseudo code shows an exemplary sensor node application. The program polls a sensor and propagates this information to its neighbor if it exceeds a certain critical value (e. g. the temperature level in a habitat monitoring scenario):

```
do forever {
    val = read_sensor()
    if (val > threshold)
        send_packet(val, neighbor)
    sleep(cycle_period)
}
```

Two parameters have an influence on the energy consumption and the corresponding battery lifetime of a node running this simple application: `cycle_period` determines the length of the sleep phase, while the value of `threshold` has an impact on the number of network transmissions.

The energy consumption of the application under different duty cycles can be calculated or derived from simulation. However, the impact of the second parameter on battery lifetime is not known before the node is deployed and sensor readings are available.

Approaches to address this problem aim for structural redundancy or design the application for the worst case, i. e. every sensor measurement exceeds the threshold. Depending on the average case, this solution could result in an over-provisioning of battery capacity, potentially violating other limitations like weight or size constraints.

As an alternative, the application could derive its energy characteristics at run-time and, if necessary, be adapted (either by itself or another node). In the example above, a simple, straightforward solution would be to change one of the two parameters `threshold` or `cycle_period` if the target system lifetime cannot be reached. Therefore, the application has to be aware of its energy consumption.

Dynamic routing algorithms are another application of reconfiguration. As the nodes that are responsible for forwarding network traffic (the so called "covering set") will probably deplete their energy reserves earlier than other nodes, power-aware routing protocols reassign their roles from time to time in order to balance the energy consumption of all nodes and to increase the lifetime of the whole network. Therefore, nodes have to be aware of their remaining battery lifetime and support reconfiguration.

In the next section, we discuss different approaches to characterize the energy consumption of a sensor application during run-time and present solutions for system reconfiguration in section 3. Related work is discussed in section 4.

Strategies for energy-aware code distribution (discussed, e. g., in [10]) are outside the scope of this paper.

All measurements and implementations were conducted on the BTnode platform [12, 2] running TinyOS [5, 8]. The BTnode (rev. 3) is equipped with an Atmel ATmega 128L microcontroller, a Bluetooth subsystem (Zeevo ZV4002) and a low-power radio (Chipcon CC1000).

2 Energy Accounting

In order to decide when and how to reconfigure a running system its energy characteristics, i. e. the capacity of the battery and the energy consumption, have to be known. The energy consumption can be estimated on-line by observing state changes and the occurrence of specific events. However, the remaining system lifetime is not only influenced by the application's energy consumption but also by environmental conditions like the current temperature. Information from a battery sensor can be used to calibrate lifetime estimations based on the accounted energy. In this section, we discuss the different approaches to run-time energy estimation in more detail.

2.1 Battery lifetime estimation

One line of the A/D converter on the BTnode is connected to the batteries, enabling the application to measure the battery voltage with few instructions at runtime. The batteries' discharge characteristics are known, so it is possible to give at least a rough estimation on the remaining lifetime.

In our tests, we used standard alkaline batteries. We did not succeed in finding a simple battery model with easily computable parameters, since their discharge characteristics are not in the form of a simple function. The middle part of these characteristics, which covers most of the lifetime, can be approximated by a linear function. This allows a sufficiently accurate estimation of the remaining lifetime.

If the batteries can be expected to have the same type, the discharge characteristics could be stored on the node, resulting in a more accurate lifetime prediction. These data tables could grow very large, depending on factors like the number of battery types, resolution and temperature.

If the discharge characteristics could be approximated by a simple mathematical function, the development of a battery model would be more feasible. This would eliminate the potentially large tables at the expense of more computation. For example, a combination of one quadratic polynomial and two exponential functions can be fitted manually to the discharge characteristics of our batteries. On the BTnode, however, algorithms for an automatic fitting currently fail due to the low resolution of about 250 distinct values and, on top of that, erratic A/D converter output in the range ± 2.

With the restriction to one battery type and a narrow temperature range it is possible to make a rough estimation on the remaining battery lifetime. One way to counter the poor resolution of the A/D converter is to sum up a number of samples taken in quick succession. The dissipation characteristics of the battery type can be stored in a table (e. g. in flash memory) or even computed since they are approximately linear most of the time.

2.2 Event-based energy accounting

Event-based accounting associates the occurrence of specific events, e. g. the transmission of a network packet, with a certain amount of energy. Time- or state-based accounting measures the time a component (CPU, radio) spends in a specific operating mode. Each operating mode is attributed with a specific energy weight.

The power consumption of the node for different operating modes as well as the energy consumption of specific events can be determined using measurement hardware or based on information from data sheets. In [7] and [11], detailed power measurements of the ATmega128 microprocessor in different sleep modes and the ChipCon radio, both of a Mica2 sensor node, are listed.

operation	power consumption	current draw
xor	31.6 mW	10.8 mA
fmul	29.3 mW	10.0 mA
lds	31.1 mW	10.6 mA

Table 1. Active power consumption of the ATmega128 performing different operations.

We measured the voltage drop at a high-precision resistor in the power lines between the batteries and the node with an A/D-converter operating at 20 kHz with a resolution of 8 bit.

Current processors for desktop and high-end systems show a wide variation of the active power consumption, depending on the functional units accessed or the operations executed [1]. However, our measurements of the ATmega128 indicate that the power consumption of this CPU is rather constant, see table 1. Therefore, energy accounting based on the run-time of tasks in the system should provide sufficient accuracy. This approach has the advantage that the implementation overhead (regarding code size and execution time) is limited.

In order to implement energy accounting, we modified the scheduler routine of TinyOS. All runnable tasks in the system are stored in a scheduler queue. The scheduler function executes all enqueued tasks and then puts the system into sleep mode. An interrupt handler is able to add a task to the scheduler queue. Upon arrival of an interrupt, the sleep state is left and the queue is processed once again. A straightforward approach to implement CPU energy accounting is to measure the time before and after the sleep phase (`sleep()`):

```
do forever {
    while( ! queue.empty() ) run_next_task()
    cpu_accounting()
    sleep()
    cpu_accounting()
}
```

For more accurate timing values, accounting could be started early in each interrupt handler.

In order to be of practical use, energy estimations on a resource-constrained platform like sensor nodes should be derived efficiently and without significant impact on battery lifetime. We measured the overhead of energy accounting for a simple application waking up 4 times a second. The energy consumption of the entire system is increased by 34.4 μW or 4.1 %. This overhead depends on the application's duty cycle and the number of interrupts. Code size is increased by 356 bytes and memory consumption by 6 bytes (a 48 bit counter is used to store the energy consumption).

As applications usually consist of periodic tasks, the required timer can be utilized to account the energy spent in sleep mode. The difference in power consumption of various sleep modes has to be taken into account. Depending on the system configuration (timers and interrupts), the power management module of TinyOS chooses the deepest sleep mode that still offers the required wake up functionality. For the BTnode, the power consumption varies from 14.28 mW in "idle" mode to 0.71 μW in "power down" mode.

To account the energy consumption of peripheral devices like the ChipCon radio a combination of event- and time-based accounting could be used. The occurrence of specific events, for instance the sending of network packets can be counted, see e .g. [9]. The energy weight for each transmission depends on the transmit power. In addition to that, the time the device spends in different states/operating modes, e. g. when listening for incoming packets could be measured similar to the run-time accounting for CPU energy.

3 System Reconfiguration

We evaluated three different methods of system reconfiguration: adaptive applications, multiple pre-installed applications and application upload.

Adaptive applications
A single application supports reconfiguration by switching to other code parts or simply through some parameters held in RAM or EEPROM. In our example (see section 1), the parameters would be `threshold` and `cycle_period`. This method requires neither a communication channel nor re-programming of the flash memory. Reconfiguration can be triggered by the application itself or upon request by another node. This method's overhead and complexity heavily depends on the application and its desired flexibility.

Multiple pre-installed applications
To simplify application design, one could consider storing two or more complete applications (or versions of one application) on the same node. On the BTnode, the 128 kBytes of flash memory provide sufficient storage space. To ease the installation of several programs on a node, we automated the process of linking two or more applications together.

A requirement for switching between interrupt-enabled applications is to update the interrupt vectors. We investigated two solutions: re-programming the interrupt vector table in flash memory and introducing an interrupt dispatcher.

To avoid programming the flash at runtime, the reset and interrupt vectors are redirected to a dispatcher routine which reads an ID from RAM or EEPROM and calls the right handler in the corresponding application. Like the first method, this approach requires neither communication nor re-programming of the flash memory. It allows for more radical changes in applications, as the applications do not have to answer the same interrupts. The application switch is fast: Apart from the reset, only one byte in RAM has to be changed.

Alternatively, the interrupt vector table in flash memory can be re-programmed before switching to another application. This method's runtime overhead is lower at the expense of a higher switching overhead. As we expect applications to switch only infrequently, the lifetime of the flash memory (the ATmega128 is specified for 10,000 write/erase cycles) should not be a concern.

The interrupt dispatcher adds 56 clock cycles at every interrupt. The energy overhead is within the measurement error of our DAQ hardware.

Application upload
The application loads the complete binary of another application, programs the flash memory and triggers a reset, similar to "Xnp" of TinyOS [6]. This method requires a communication channel.

The overhead is considerable:

– periodic listening on the selected communication channel,
– connection setup,
– transfer time (18.9 s for 16.5 kBytes using Bluetooth)
– programming multiple pages in flash memory (20 ms, 1-2 mJ per page)

In spite of its large overhead, this solution can be advantageous over the other methods in that the application can be changed after deployment. A more efficient approach would be to replace only parts of an application as proposed in [3, 4].

The above-mentioned methods of application switching and upload could even save more energy than a single application. For example, a virtualized timer using lists of scheduled events is only needed if it is used by several modules. In a power-saving application where only one module uses the timer, the whole virtualization layer is superfluous and can be left out.

4 Related Work

A multitude of approaches based on simulation or emulation has been proposed to estimate the energy consumption of sensor nodes. *Power Tossim* is a simulator for TinyOS providing a prediction of the energy consumption of Mica2 nodes [11]. *AEON* (Accurate Prediction of Power Consumption) is a tool to estimate the energy consumption of sensor nodes. Measurements of the current draw of Mica2 nodes are fed into a sensor node emulator. The energy consumption is quantified by observing state changes of hardware components.

System reconfiguration of sensor nodes has been extensively studied, however we know of no research project that focuses on energy issues. TinyOS supports remote, in-network programming of (at least) Mica2 nodes, called *Xnp* [6]. A system image in `srec`-format is downloaded using radio communication and stored in flash memory. A "boot loader" residing in a reserved section of the program memory is responsible for reconfiguration. *Contiki* allows the dynamic loading and replacement of individual programs and services on sensor nodes [3]. It supports incremental updates which require less energy as only the application binary and not the entire system has to be transmitted and installed. *SOS* is an infrastructure for software updates [4]. Measurements show that it performs comparably to Xnp of TinyOS in terms of energy usage and performance and better in terms of energy consumption during software updates. Like Contiki, it supports modular updates.

5 Conclusion

Sophisticated power management policies have been proposed for mobile, desktop and high-end computers. For battery-powered, resource-constrained embedded devices like sensor nodes, research has focused on off-line estimation of power consumption using simulation and emulation techniques. In this paper, we discussed different approaches to retrieve the energy characteristics of a sensor node application during runtime and to adapt or reconfigure the running system. An analysis of the overhead in energy and code size for prototype implementations on the BTnode are presented.

References

1. F. Bellosa, A. Weissel, M. Waitz, and S. Kellner. Event-driven energy accounting for dynamic thermal management. In *Proc. of the Workshop on Compilers and Operating Systems for Low Power (COLP'03)*, September 2003.
2. J. Beutel, O. Kasten, F. Mattern, K. Römer, F. Siegemund, and L. Thiele. Prototyping wireless sensor network applications with btnodes. In *1st European Workshop on Wireless Sensor Networks (EWSN)*, January 2004.
3. A. Dunkels, B. Grönvall, and T. Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Proc. of the First IEEE Workshop on Embedded Networked Sensors 2004 (IEEE EmNetS-I)*, November 2004.
4. C.-C. Han, R. K. Rengaswamy, R. Shea, E. Kohler, and M. Srivastava. Sos: A dynamic operating system for sensor nodes. In *Proc. of the Third International Conference on Mobile Systems, Applications, And Services (Mobisys)*, June 2005.
5. J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for network sensors. In *Proc. of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-IX) 2000*, November 2000.
6. C. T. Inc. *Mote In-Network Programming User Reference*, 2003.
7. O. Landsiedel, K. Wehrle, and S. Götz. Accurate prediction of power consumption in sensor networks. In *Proc. of The Second IEEE Workshop on Embedded Networked Sensors (EmNetS-II)*, May 2005.
8. P. Levis, S. Madden, D. Gay, J. Polastre, R. Szewczyk, A. Woo, E. Brewer, and D. Culler. The emergence of networking abstractions and techniques in tinyos. In *Proc. of the First USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI 2004)*, March 2004.
9. S. Madden, M. J. Franklin, and J. M. Hellerstein. Tag: a tiny aggregation service for ad-hoc sensor networks. In *Proc. of the 5th Annual Symposium on Operating Systems Design and Implementation (OSDI)*, December 2002.
10. N. Reijers and K. Langendoen. Efficient code distribution in wireless sensor networks. In *Proc. of the International Workshop on Wireless Sensor Networks and Applications*, September 2003.
11. V. Shnayder, M. Hempstead, B. rong Chen, G. Werner-Allen, and M. Welsh. Simulating the power consumption of large-scale sensor network applications. In *Proc. of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys'04)*, November 2004.
12. E. T. H. Zürich. Btnode - a distributed environment for prototyping ad hoc networks. Web page. Visited 2005-12-06, http://www.btnode.ethz.ch.