Universität Karlsruhe (TH)
Forschungsuniversität • gegründet 1825

Fakultät für **Informatik**

GI/ITG KuVS Fachgespräch Systemsoftware und Energiebewusste Systeme

11. Oktober 2007, Universität Karlsruhe (TH)

Herausgeber: Prof. Dr. Frank Bellosa

Interner Bericht 2007-20

ISSN 1432-7864

Universität Karlsruhe - Fakultät für Informatik - Bibliothek - Postfach 6980 - 76128 Karlsruhe

Inhaltsverzeichnis

SANDMAN: an Energy-Efficient Middleware for Pervasive Computing

Gregor Schiele
Information Systems II
Universität Mannheim
Schloss, 68131 Mannheim, Germany
gregor.schiele@uni-mannheim.de

Christian Becker
Information Systems II
Universität Mannheim
Schloss, 68131 Mannheim, Germany
christian.becker@uni-mannheim.de

ABSTRACT

Energy efficiency in pervasive computing is crucial for devices operated by battery. To provide energy efficiency we created an energy efficient middleware, called SANDMAN. In this paper we present an overview on the past research done in the SANDMAN project and the current and future directions of our work.

1. INTRODUCTION

Energy is a crucial resource in pervasive computing systems with mobile devices. These devices are often embedded into everyday items and can not be provided with a large battery or a fixed connection to the power grid. Thus, the efficient operation of devices with respect to energy is a major challenge of such systems. When designing our pervasive computing middleware BASE [2], we experienced this challenge and decided to integrate algorithms and mechanisms for energy-efficiency in our middleware.

When starting our work, we looked at the main sources of energy consumption. The first thing we learned is that while a lot of work has been done to lower the energy consumption for sending and receiving data, a large additional amount of energy is consumed by idle devices waiting to be used. As an example, it takes 805 mW to keep an IEEE 802.11 network interface up and running without sending data [3]. Idle devices provide currently unneeded and thus unnecessary resources and consume energy by doing so. This energy can be saved by temporarily switching such devices in a low-energy sleep mode. However, doing so results in a number of challenges that must be addressed to keep the system operational, e.g., network connectivity and service discovery.

In this paper we report on the challenges we met when designing an energy-efficient middleware for pervasive computing that allows to power down currently unused devices. We also discuss solutions to overcome these challenges and present the current and future work done in the project.

The paper is structured as follows. First, we define our system model and assumptions. After that we describe the features of our existing middleware BASE that are needed to understand our approach. Following to this we present our approach towards an energy-efficient middleware and evaluate our middleware briefly. Finally, we discuss current work and how we plan to proceed from the current project state, give the related work, and finish the paper with a short conclusion.

2. SYSTEM MODEL AND ASSUMPTIONS

Our targeted system class consists of a number of battery-operated mobile devices. Each device is equipped with one or more network interfaces. Using these interfaces the devices form a number of wireless mobile ad hoc networks (MANETs). Basic MANET functionality, e.g., addressing, remote execution, is offered by our pre-existing middleware, which we assume to be installed on each device. In addition, we assume that each device has two operational modes: a fully operational AWAKE mode and an energy-efficient SLEEP mode. While in SLEEP mode, the device can not perform any operations or communicate. To switch back to AWAKE mode, the device has an internal timer, e.g., a watch dog timer, which reactivates the device after a predetermined time. This simple model can be extended to multiple different operational modes. However, in this paper we restrict ourselves to the simple model to ease the description of the main concepts used in our approach.

3. BASE

We designed our energy-efficient middleware as a number of extensions to our existing middleware BASE. This allowed us to concentrate on the new challenges imposed by our need to save energy while reusing a lot of previous work. Before describing our extensions in more detail, in this section we present the parts of the basic middleware that are needed to understand the extensions.

BASE is designed to be a minimal yet flexible communication middleware for pervasive computing. It does not rely on any external infrastructure, enabling the devices to cooperate with each other in a peer-to-peer fashion.

The architecture of BASE is shown in Figure 1. The middleware is structured as an extensible micro broker. The broker itself manages interactions with remote devices and synchronizes them with respect to the application's desired interaction model. To communicate, (semantic, serializer, modifier and transceiver) plugins are used to add support for different communication technologies and protocols. As an example, to access a CORBA service, the device developer only has to integrate an IIOP plugin into the BASE configuration. The management of these plugins is the responsibility of the plugin manager. At runtime the middleware detects nearby devices (with a discovery plugin), negotiates communication abilities with them and allows the local application to access other devices using a service abstraction. Once an interaction takes place, BASE automatically builds a suitable protocol stack by selecting and integrating multiple plugins. To adapt to networking changes, BASE is able

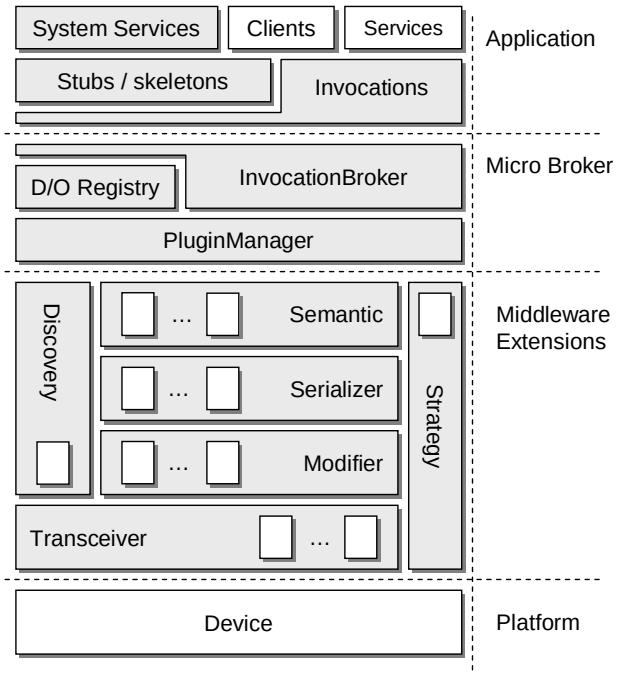


Figure 1: BASE Architecture

to reselect the used plugins dynamically. More information about BASE can be found in [2].

4. SANDMAN

To add energy efficiency support to our existing middleware BASE, we extended it with a number of system services to allow an energy-efficient operation of each device. The resulting new middleware system is called SANDMAN. It stays fully compatible with existing BASE installations. SANDMAN is designed around three main concepts to save energy:

1. Reduce the energy consumed by transferring data by selecting the most energy-efficient communication protocols that are available in a given situation.

2. Switch idle devices to their low power SLEEP mode to reduce unnecessary standby energy consumption.

3. Allow clients to select the most energy efficient service to create energy-efficient application configurations.

The first concept can be realized easily with BASE using its existing ability to select plugins dynamically. To do so, the plugin descriptions must be enhanced with information about their energy consumption and a suitable selection strategy must be provided and integrated. Whenever a new protocol stack is selected, the selection strategy accesses the plugin descriptions and selects the most energy efficient configuration. The second concept, the deactivation of idle devices presented us with a number of challenges, which we discuss in the following sections. The third concept, the selection of energy-efficient services is subject to current and future work and discussed later.

4.1 Transition Scheduling

The first challenge when deactivating idle devices is to schedule deactivations properly. Often, it is not easy to decide whether a device is unused and can be deactivated. It may be idle at the moment but play a crucial role in the execution of an application in the near future. As an initial approach, we relied on a transition strategy with a fixed inactivitiy threshold. Such approaches are well known from the area of Dynamic Power Management. They can be implemented very efficiently even on resource-restricted devices, as they only require a timer to operate. In addition, we added an interface to the middleware that allows application code, e.g., service implementations, to explicitly specify that the device is currently in use and should not be deactivated. Further information is provided by the BASE microbroker, which notifies SANDMAN about incoming and pending requests, as well as currently used local services. This rather simple approach works well in cases where specific usage patterns are difficult to determine. In other scenarios, more complex idle detection mechanisms, e.g., based on statistical approaches, could be beneficial. Finally, BASE handles each interaction between a client and a service individually. While this results in a very flexible system, we decided to add an additional abstraction for service usages, so-called *sessions*. Using a session, a client can specify that it currently uses a given service. This information is then forwarded to SANDMAN which will not deactivate the device offering the service, even if there is no client interaction for some time. To cope with suddenly disappearing clients, leases are used. In addition, sessions can be used by clients to negotiate with the service that the latter may sleep even while the client is using it, e.g., because the client can cope with a given latency. Client and server can also negotiate synchronization times, i.e., they will communicate at given times only, allowing both to temporarily sleep. In our implementation, the ability to open a session is provided but negotiation strategies are subject to future work and must be provided by application developers at this time.

4.2 Service Discovery

Before using them, clients must first discover devices and their services. However, existing discovery approaches like UPnP or Jini cannot handle deactivated devices and wrongly assume that they have left the system. Thus, before deactivating a device, we must make sure, that it stays discoverable. To do so, we developed a self-adaptive discovery protocol that can handle deactivated devices. Our approach works as follows: at startup time, each device operates autonomously and answers discovery requests from remote clients directly. In this state the system resembles a classical UPnP discovery system. During the system operation, the devices cluster themselves with neighboring devices that have the same mobility pattern as themselves. This ensures that the resulting clusters are highly stable, which is necessary to achieve long sleep times without introducing errors in the discovery process. Otherwise, devices that left the communication range of their clusters while sleeping could lead to phantom discoveries. Each cluster has a single leader, the so-called cluster head (CH). Once a cluster is formed and a CH elected, all devices in the cluster switch their discovery system to a registry-based approach, resembling Jini. The CH collects information about all services in its cluster and answers discovery requests from clients for

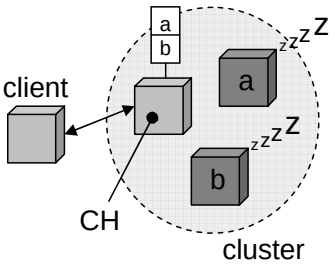


Figure 2: SANDMAN Approach

them as shown in Figure 2. This allows all other devices in the cluster to switch to their SLEEP mode, while the CH keeps advertising them. In addition to this, the CH can act as a proxy to detect new services for sleeping client devices. To accept new client requests or receive information about newly detected services, each device in a cluster awakes regularly.

An overview of the protocol used by SANDMAN to put devices to their SLEEP mode is given in Figure 3. In this example, we assume that a cluster consisting of two devices n_1 and n_2 has already been formed and omit the messages necessary for cluster management. At the beginning, n_2 starts its inactivity threshold timer. If n_2 is idle for t_is, it decides to go to sleep and sends a $SLEEP\_ANC$ message to its CH n_1, including the desired sleep time t_{sd}. The CH can modify this sleep time to allow cooperative scheduling algorithms as discussed later. It stores the new sleep time t_s in its local database for n_2 and sends back a $SLEEP\_ACK$ message with the sleep time. After receiving this message, n_2 configures its internal watch dog timer to reawake after t_ss and transitions to its sleep mode. Meanwhile, a client device n_3 contacts the CH to search for services. The CH finds that n_2 offers a service suitable for n_3 and announces this to the client device. In this message, it includes the service descriptions, the plugins that can be used to contact the device as well as the remaining sleep time of n_2 (zero if the device is awake). The client waits for the specified time until n_2 awakes. Then, it contacts n_2 directly and uses its service. A special case arises, if the device wants to sleep shortly after a client device discovered one of its services. The CH cannot know, if the client will contact the device and thus denies any sleep requests from a device, if the time between its last discovery and the sleep request is smaller than a given threshold t_a. Once a service is no longer used, its device restarts its inactivity threshold timer and the algorithm starts anew. More information about the service discovery approach and the protocols used (e.g., for clustering) can be found in [9] and [8].

4.3 Connectivity Preservation

In addition to keeping the devices discoverable, the network connectivity must be maintained. If we deactivate devices at will, we will most likely lower the connectivity of the network. We may even induce network partitioning. Luckily, we can reuse the solution chosen for the discovery and put the responsibility for routing on the CHs. In addition, we have to make sure that the CHs form a connected overlay network and can reach all nodes. To do so we design our clustering approach such that it not only uses the mobility patterns of devices for its clustering decision but also the current neighbor graph of the devices. Two devices are clustered iff they have the same neighbors. This makes sure that each one of them can act as CH and will be able to reach all neighbors. An example for this are devices carried by the same user. These devices are nearby and typically have the same neighbors. Note that to really ensure this property, we have to recheck it regularly to cope with later connectivity changes.

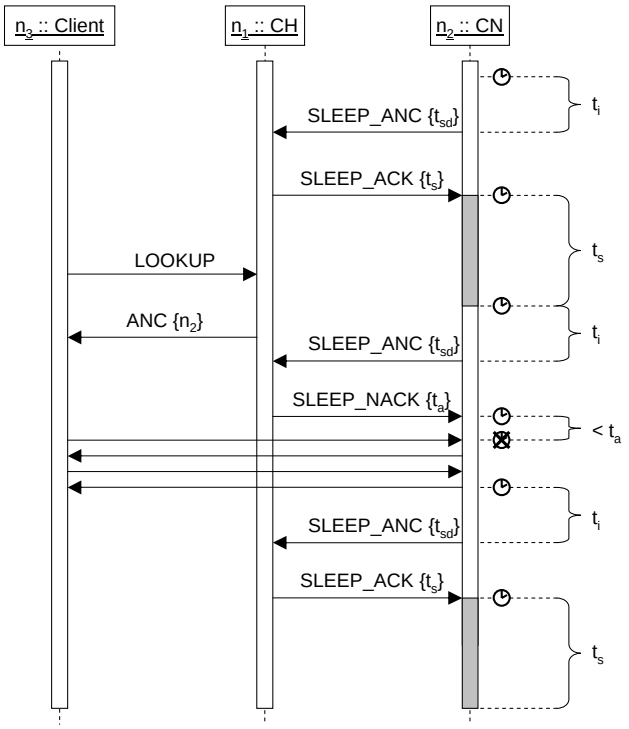


Figure 3: SANDMAN Protocol Overview

This approach consumes additional energy, first because fewer devices are clustered and second to perform the regular check. If we can accept a certain (small) loss in connectivity, we can schedule the rechecks to occur only rarely or omit them altogether. In addition we can accept a certain amount of difference in neighboring sets when clustering devices, e.g., we cluster devices when their neighborhoods overlap by at least 90%. Using these parameters we can adapt the system behaviour between more connectivity preserving and more energy-efficient as needed.

4.4 Interaction Latency

When a device is asleep it cannot be reactivated preliminarily, e.g., to handle an unexpected request by the user. Clearly, in some cases the user might be able to manually reactivate a device prior to its scheduled awake time by pressing a special button, etc. However, we do not assume that this is always possible or even the normal case. Thus, a client wanting to use a sleeping device must wait until the device awakes on its own. This slows down the client's execution and may consume additional energy. Therefore, it is important to lower the experienced interaction latency. To do so, we propose to cooperatively schedule the sleep times of all devices in a cluster. To realize this, SANDMAN allows

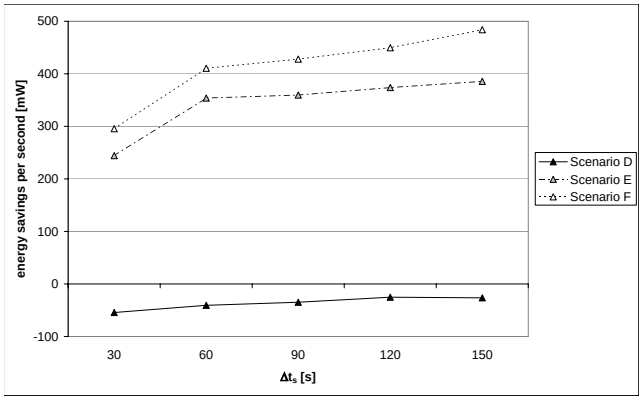


Figure 4: Energy Savings

CHs to manage the sleep schedule of its whole cluster locally and to coordinate all devices accordingly. Currently, we are examining two cooperative scheduling algorithms: the first interweaves the sleep times of devices offering the same service such that the time until one of these devices awakes is minimized. The second keeps one device awake all the time, allowing clients to use a service without any additional delay. The device that must stay awake is chosen by the CH in a round-robin fashion. Our current implementation includes only a simple scheduling algorithm that operates on isolated devices. Cooperative scheduling algorithms are subject to current and future work.

4.5 Evaluation

To evaluate the energy savings that can be achieved by putting idle devices into their SLEEP mode, we performed a number of experiments using the Network Emulation Toolkit (NET) [5]. NET is a Linux-based emulation environment developed at Stuttgart University for testing and evaluating network protocols in both stationary and mobile environments. For our experiments we defined scenarios with different mobility characteristics, e.g., device speed and device group size. Figure 4 shows the resulting energy savings for three scenarios with a device speed of 2 m/s and three different group sizes, single devices (Scenario D), groups of 4 (Scenario E) and groups of 10 devices (Scenario F). Clearly, a group size of one leads to the well known random waypoint model. The results are shown for different sleep times Δt_s and are averaged over all devices in a single cluster, i.e. they include the overhead experienced by the CH.

In Scenario D, the devices consume more energy than without SANDMAN. This is due to the fact that devices are clustered rarely and the message overhead due to clustering consumes more energy than is saved by sleeping devices. Therefore, for this scenario, SANDMAN is not beneficial and should not be used. However, for larger group sizes, the devices are able to save up to 484 mW per node for Δt_s=150 s and a group size of 10. For the chosen continuous device consumption of 805 mW, this is a saving of approximately 60% per device, including CHs and unclustered devices. For scenarios with other movement speeds the results are accordingly, while total values for higher speeds are lower. This is the case, as with higher mobility, clusters become less stable and devices must recluster more often. We can observe the same effect when comparing scenarios with identical group sizes but different movement speeds. The achieved energy savings are lower for higher speeds. A much more elaborate evaluation of our approach, including message overhead, savings, delays and discovery errors can be found in [8].

5. CURRENT AND FUTURE ACTIVITIES

SANDMAN so far provides basic functionalities to enable energy-efficient device operation. However, different possibilities to further enhance the achievable energy savings exist and are currently evaluated by us. The most promising ones are discussed in the following sections.

5.1 Transition Scheduling

The transition scheduling strategy currently implemented in SANDMAN does not take into account other devices in a cluster. Instead it operates completely isolated. In addition, we use fixed, preset values for the parameters involved in the strategy, e.g., the inactivity threshold and the chosen maximum sleep time. Clearly, there are a number of possibilities to enhance this approach. First, we can enhance the inactivity threshold strategy by using dynamic parametrization. Second, we are currently examining more advanced transition strategies, e.g., based on statistics, to provide us with better predictions of future device usages. Third, we already developed a first cooperative transition scheduling algorithm, which takes into account all services in a cluster when computing sleep times. This algorithm must be evaluated and analyzed further.

5.2 Service Selection

A major issue in service oriented systems is the selection of suitable services by clients. From an energy efficiency point of view, this selection should depend heavily on the resulting energy usage. Thus, if multiple services are available, the client should use the one which leads to the most energy efficient application configuration. However, without system support, the client cannot decide which one is this. The resulting energy consumption depends on many factors and cannot predetermined with total certainty. As some prominent examples, the energy consumption depends on the amount and frequency of communications between client and server, the local execution cost of the service on its device, and the additional consumption if the service uses additional services to provide its functionality. In addition, the stability of the resulting configuration must be taken into account. A service might be highly energy efficient but is expected to become unavailable in short time, leading to another application reconfiguration with additional costs.

In the future, we want to provide additional support for selecting energy-efficient services. To do so, we plan to develop additional algorithms to model and predict the resulting energy usage of different configurations. First, we can use an analytical model to compute an estimated consumption. Second, we can rely on historical data, i.e. measurements taken for past configurations (see, e.g., [6]). In reality, we expect solutions that combine these two approaches to provide the best trade off between complexity and energy-efficiency.

5.3 Session Negotiation Strategies

As described before, sessions allow clients and services to negotiate energy saving strategies, e.g. by specifying com-

mon synchronization points. Although SANDMAN already allows such negotiations, additional support to do so would be beneficial. Most importantly, different strategies must be developed and analyzed to help application developers to decide on the best strategy for their code.

5.4 Adaptive Service Discovery

The Consumptions for discovery and usage must be carefully weighted against each other. It may be beneficial (at least from the system's perspective) to use a slightly worse service that was discovered with much less effort. Again, the precondition to follow this approach is the provision of exact and efficient models to estimate the energy consumption of a future service usage. Once this information is available, the system can adapt its discovery efforts depending on the achievable savings. As an example, if an application uses a very energy consuming service at the moment, SANDMAN can increase the frequency and range of discovery requests to find a better service. On the other hand, if a nearby energy-efficient service is used, the need for additional discoveries is low and the middleware can decide to stop searching for alternative services, until the used one becomes unavailable.

6. RELATED WORK

There are a number of existing energy-efficient middleware systems complementing our approach. The GRACE project [7] aims at reducing the energy-consumption of mobile devices that process multimedia data. It combines system functions like process scheduling, CPU power management and data encoding to enable global adaptation. The MillyWatt project [10] enables battery-powered devices to run for a predefined period of time. To do so, active devices are deactivated periodically for a specific fraction of time. In contrast to this, we deactivate idle devices, only. The Power Aware Reconfigurable Middleware (PARM) [4] and the Remote Processing Framework (RPF) [6] enable energy savings on mobile battery powered devices by shifting energy intensive tasks to resource rich devices. Our approach is able to do this by modelling such tasks as services that can be executed remotely. However, we currently do not support clients in selecting whether a given service should be executed locally or remotely. Another approach is taken by MagnetOS [1]. Through the continuous redistribution of application parts across the available devices of a mobile ad hoc network, MagnetOS reduces the communication cost by reducing the length of data paths.

Regarding energy-efficient service discovery, the DEAP-Space system enables devices to safely deactivate their communication adapters. To keep devices discoverable, it uses synchronized time windows to broadcast service announcements in a single hop environment. Our approach is aimed at multi hop networks and does not require synchronized devices, enabling optimized interaction latencies.

7. CONCLUSION

In this paper, we have presented our energy-efficient middleware SANDMAN. SANDMAN is realized as a number of extensions to BASE, our minimal and adaptive communication middleware for peer-based pervasive computing environments. It supports energy-efficient communication by selecting energy-efficient protocol stacks, and deactivates idle devices to reduce the idle standby energy consumption. To do so, SANDMAN clusters devices dynamically depending on their mobility patterns and neighboring devices. This allows to deactivate devices while preserving the network connectivity and the discovery of the devices and their services. Work is going on in different directions. Most prominently, we expect energy efficient service selection to emerge as a major enhancement to save additional energy. To do so, the future energy consumption of different application configurations must be predicted, e.g. using suitable analytical models or historical measurements.

8. REFERENCES

[1] Barr, R., Bicket, J., Dantas, D., Du, B., Kim, T., Zhou, B., and Sirer, E. On the need for system-level support for ad hoc and sensor networks. *ACM SIGOPS Operating Systems Review 36*, 2 (April 2002).

[2] Becker, C., Schiele, G., Gubbels, H., and Rothermel, K. Base – a micro-broker-based middleware for pervasive computing. In *Proceedings of the IEEE international conference on Pervasive Computing and Communications (PerCom)* (Mar. 2003).

[3] Feeney, L. M., and Nilsson, M. Investigating the energy consumption of a wireless network interface in an ad hoc networking environment. In *Proceedings of the 20th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2001)* (Anchorage, AK, USA, April 2001).

[4] Mohapatra, S., and Venkatasubramanian, N. PARM: Power aware reconfigurable middleware. In *Proceedings of the IEEE 23rd International Conference on Distributed Computing Systems (ICDCS)* (May 2003).

[5] NET - Network Emulation Testbed. Webpage. http://net.informatik.uni-stuttgart.de/.

[6] Rudenko, A., Reiher, P. L., Popek, G. J., and Kuenning, G. H. The remote processing framework for portable computer power saving. In *Proceedings of the ACM Symposium on Applied Computing (SAC99)* (Feb. 1999), pp. 365–372.

[7] Sachs, S., Yuan, W., Hughes, C., Harris, A., Adve, S., Jones, D., Kravets, R., and Nahrstedt, K. GRACE: A hierarchical adaptation framework for saving energy. Technical Report UIUCDCS-R-2004-2409, University of Illinois, February 2004.

[8] Schiele, G. *System Support for Spontaneous Pervasive Computing Environments*. PhD thesis, University of Stuttgart, 2007. Adviser-Kurt Rothermel and Adviser-Christian Becker.

[9] Schiele, G., Becker, C., and Rothermel, K. Energy-efficient cluster-based service discovery. In *Proceedings of the 11th ACM SIGOPS European Workshop* (September 2004).

[10] Zeng, H., Fan, X., Ellis, C., Lebeck, A., and Vahdat, A. ECOSystem: Managing energy as a first class operating system resource. In *10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS X)* (Oct. 2002).

Energiemanagement in REFLEX

Karsten Walther

Oktober 2007

Tief eingebettete Systeme wie z.B. Sensornetzknoten beziehen aktuell ihre Energie aus Batterien bzw. sollen zukünftig Umgebungsenergie nutzen [5]. Prinzipiell steht wenig Energie zur Verfügung und das Gerät und insbesondere der verwendete Mikrocontroller muss möglichst sparsam mit dieser Ressource umgehen. Dies ist eine nicht triviale Aufgabe und die Steuerung ist typischerweise tief im System verwurzelt. In dem Vortrag wird ein Energiemanagementkonzept für ereignisgetriebene Systeme am Beispiel von REFLEX vorgestellt. Die Energiesparmassnahmen erfolgen weitgehend implizit und ermöglichen eine einfache Entwicklung energieeffizienter Programme. In dem Konzept werden weiterhin die Architektureigenschaften typischer Mikrokontroller berücksichtigt.

1 Einleitung

Mikrocontrollerbasierte Systeme sind schon seit langer Zeit Bestandteile des täglichen Lebens und ihren Anwendung scheint noch lange nicht erschöpft. Ein dennoch bestehendes Problem ist die Energieversorgung bei neuen Anwendungen wie Sensornetzwerke oder kabellosen Automatisierungslösungen. Im Gegensatz zu leistungsfähigeren mobilen Geräten sollen die dort eingesetzten Systeme bei gleichzeitig wesentlich kleinerer Batteriekapazität über mehrere Monate funktionieren .

Dies führt dazu, dass möglichst kleine sparsame Mikrocontroller eingesetzt werden. Die Verwendung solcher Mikrocontroller alleine reicht jedoch nicht für Systemlaufzeiten von mehreren Monaten oder Jahren aus. Dafür müssen die vorhandenen Energiesparmodi der Hardware auch genutzt werden. Dabei stellen sich 2 große Probleme. Erstens die Diversität der Hardware, d.h. es muss eine geeignete Abstraktion gefunden werden, welche ein plattformunabhängiges aber dennoch effektives Energiemanagement möglich macht. Zweitens sollten die Mechanismen weitestgehend implizit wirken, damit der Anwendungsprogrammierer sich auf die Lösung seines eigentlichen Problems konzentrieren kann.

Im Folgenden wird ein Konzept zum Energiemanagement in tief eingebetteten Systemen und der beispielhaften Umsetzung für REFLEX [7] vorgestellt. Grundlegend ist dabei die ereignisgetriebene Abarbeitung, welche gut geeignet für typische tief eingebettete Systeme ist [6, 1].

2 Grundlagen

2.1 Technische Grundlagen

Mikrocontroller haben mehrere für das Energiemanagement relevante Eigenschaften. Sie werden z.B. mit wesentlich geringeren Taktfrequenzen betrieben als moderne Mikroprozessoren und haben auch keinen Cache. Weiterhin verfügen Mikrocontroller über integrierte Module auf dem Chip wie z.B. AD-Wandler, welche schnell an- und abgeschaltet werden können. Und es existieren meist mehrere Schlafmodi in denen ganze Modulgruppen des Controllers abgeschaltet werden können [3]. Bei dem Texas Instruments MSP430 wie auch beim Freescale HCS12 sinkt die Leistungsaufnahme im tiefsten Schlafmodus zum Beispiel auf weniger als ein Promille der Leistungsaufnahme des aktiven Modus [1].

In vielen leistungsfähigeren Geräten werden Techniken wie Dynamic Voltage Scaling (DVS) und/oder Dynamic Frequency Scaling (DFS) genutzt. Untersuchungen dieser Techniken für Low-End Mikrocontroller z.B. [4, 2] haben gezeigt, dass der Einsatz dieser Techniken dort nicht sehr vielversprechend ist. Das liegt an der geringen Last, die solche Systeme verursachen, welche die für DVS notwendige externe Beschaltung inneffektiv werden lässt. Im typischen Leistungsbereich der Plattformen liegt der Wirkungsgrad bei ca. 70%. Ein Festspannungsregler hingegen kommt auch im Niedriglastbereich auf über 90% Wirkungsgrad.

Gegen DFS spricht der nur linear zur Taktfrequemz sinkende Stromverbrauch, welcher somit nur in Kombination mit DVS Einsparung bringt aber gleichzeitig das System komplexer macht. So beruhen viele Abläufe, wie z.B. die Kommunikation mit externen Sensoren über proprietäre Schnittstellen, auf engen zeitlichen Grenzen. Schwankende Taktfrequenzen würden die Programmierung solcher Treiber nur weiter erschweren.

Daher ist die Nutzung der von den Mikrocontrollern zur Verfügung gestellten Stromsparmodi der vielversprechenste Weg [4]. Insbesondere da im Gegensatz zu leistungfähigeren Systemen mit externen Komponenten die Chip-internen schnell an- und abgeschaltet werden können.

2.2 Ereignissflussmodell

Das für die Untersuchungen verwendete Betriebssystem REFLEX stellt ein Ereignissflussmodell als Programmierschnittstelle zur Verfügung. Typische Anwendungen können damit wie in Abbildung 1 dargestellt werden.

[1] siehe Datenblatt

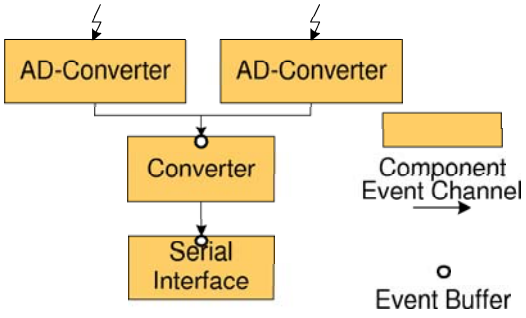


Figure 1: Event Flow Scheme of REFLEX

In diesem Modell existieren Softwaremodule, welche über Ereigniskanäle miteinander kommunizieren. In REFLEX sind das vom Laufzeitsystem verwaltbare passive Objekte. Dabei gibt es zwei Arten von Modulen. Dies sind Interrupthandler, welche infolge eines Hardwareereignisses ausgeführt werden und Aktivitäten, welche nach einem Softwareereignis gemäß einer Schedulingstrategie aufgerufen werden. Die Ereigniskanäle entkoppeln die zeitliche Ausführung von verbundenen Aktivitäten/Interrupthandler voneinander. Dazu müssen die Ereignisse, welche typischerweise mit Daten verknüpft sind, in den Kanälen gepuffert werden und können damit asynchron von dem empfangenen Modul ausgewertet werden.

REFLEX und andere ereignisgetriebene Systeme sind inherent stromsparend, da eine Verarbeitung im System nur durchgeführt wird, wenn tatsächlich etwas zu tun ist. Dies gilt insbesondere in Systemen mit wenig Last, z.B. Sensornetzwerkapplikationen.

3 Das Energiesparkonzept

Grundidee ist die Nutzung von Wissen aus der ereignisbasierten Abarbeitung eines Programms. So werden z.B. Aktivitäten im System nur ausgeführt, wenn an deren Eingängen Ereignisse aufgetreten sind, welche allesamt direkt oder indirekt von Interrupts abhängig sind. Dadurch kann ausschließlich durch das Steuern der Interruptquellen der Energieverbrauch des Systems entscheidend beeinflusst werden.

3.1 Interruptgruppen

Zur gezielten Steuerung werden Interruptquellen in Gruppen eingeordnet, welche je nach Systemmodi aktiviert oder deaktiviert werden. Vordefinierte Gruppen sind *SECONDARY*, *DREAM*, *RTI* und *UPDATE*. Dabei feuern sekundäre Interruptquellen nicht selbsständig, sondern erst in Folge von Softwareroutinen. Die Treiber solcher Komponenten sind daher selbst für das An- und Abschalten verantwortlich. Ausgabeinterrupts sind typische Vertreter dieser Gruppen. Interrupts der Gruppe *DREAM*

werden benutzt, um aus dem Traummodus (wake-on-anything-wanted) aufzuwachen, jeder nicht sekundäre Interrupt kann dieser Gruppe zugeordnet werden. Interrupts der Gruppe *RTI* werden durch die Hardware bestimmt. Normalerweise sind das die Real-Time-Clock, der Watchdog und RESET. Diese Interrupts können den Mikrocontroller auch aus typischerweise vorhandenen Tiefschlafmodi wecken, in welchen nur eine Echtzeituhr in Betrieb ist. Diese 3 Interruptgruppen werden durch das im System integrierte Energiemanagementsystem genutzt.

Die letzte Gruppe *UPDATE* ist bereits anwendungsorientiert und bestimmt einen Satz von Interruptquellen, welche während einer bestimmten Programmphase aktiv sein sollen. In diesem Fall sind das alle Interrupts, welche während einer Aktualisierung der Knotensoftware über eine I/O-Schnittstelle benötigt werden. Weitere solche Gruppen können vom Anwender angelegt werden. Beim Umschalten zwischen diesen Gruppenmodi werden entsprechend dem gewünschten Modus Interruptquellen an- bzw. abgeschaltet.

3.2 Schlafmodi

Die vordefinierten Gruppen werden vom System für allgemein verwendbare Schlafmodi genutzt. In REFLEX werden die folgenden 3 Schlafmodi angeboten, welche explizit verwendet werden können.

- *dream* (wake on anything)
- *sleep* (wake on RTC)
- *stop* (wake on RESET)

Dream dient dabei als Modus, in welchem Interrupts der Gruppe *DREAM* den Controller wieder wecken können. Der *sleep*-Modus berücksichtigt, dass die meisten Low-End Mikrocontroller einen speziellen Schlafmodus unterstützen in welchem nur ein kleiner Clock-Schaltkreis in Betrieb bleibt. Daher kann auch nur ein RTC-Interrupt (Real-Time-Clock) den Controller wecken. Dabei kann optional noch angegeben werden, wie viele RTC-Ticks das System schlafen soll. Der letzte Modus schaltet den gesamten Controller ab. Dieser Ruhezustand kann dann nur über externe Interrupts wie *RESET* wieder verlassen werden. Dies ist z.B. nützlich für batteriebetriebene oder kapazitiv arbeitende Schalter.

Der Clou an der Schnittstelle ist jedoch, dass sie nicht synchron funktioniert, d.h. der Controller wird nicht adhoc schlafen gelegt. Bevor dies geschieht, lässt das Energiemanagementsystem erst alle noch ausstehenden Aktivitäten zu Ende ausführen, lässt aber keine Interrupts mehr zu, welche im Tiefschlafmodus deaktiviert sind. Somit muss der Anwendungsprogrammierer nicht selbst überprüfen und sich eventuell benachrichtigen lassen, wann der Schlafmodus betreten werden kann.

3.3 Nullasterkennung

Die letzte Komponente im Energiemanagement ist die Nullasterkennung im System. Diese ist wichtig, da bei allen Schlafmodi von Mikrocontrollern der Rechenkern deak-

tiviert ist und daher keine Berechnungen durchgeführt werden können. Daher kann nur in Nullastsituationen in Schlafmodi gewechselt werden. In ereignisgetriebenen Systemen ist Nullast gleichzusetzen mit einer leeren ready-list im Scheduler. Der Mikrocontroller kann dann in den gewünschten Schlafmodus geschalten werden. Standardmäßig wird der maschinespezifische halt-Befehl aufgerufen, da dies dort meist der tiefstmögliche Schlafmodus ist. Dies geschieht für den Anwendungsprogrammierer vollkommen implizit.

4 Zusammenfassung

Es wurde ein Energiemanagementsystem für tief eingebettete, ereignisgetriebene Systeme präsentiert, welches die technischen Merkmale dieser Systeme, wie z.B. Verfügbarkeit von Schlafmodi, berücksichtigt. Weiterhin bietet das System eine sehr einfache Nutzerschnittstelle und nutzt durch die ereignissflussbasierte Programmierung ohnehin vorhandenes Wissen zur internen Ablaufsteuerung. Somit ist das System sehr leichtgewichtig aber verspricht dennoch eine hohe Effizienz. In den laufenden Arbeiten soll dieses durch Messungen in realen Systemen bestätigt werden.

References

[1] A.Burg. The eventflow model - a concept for real-time control of intelligent autonomous systems. Techreport, TU Passau, 1997.

[2] Y. Cho, Y. Kim, and N. Chang. Pvs: passive voltage scaling for wireless sensor networks. In *ISLPED '07: Proceedings of the 2007 international symposium on Low power electronics and design*, pages 135–140, New York, NY, USA, 2007. ACM Press.

[3] V. Ekanayake, I. Clinton Kelly, and R. Manohar. An ultra low-power processor for sensor networks. In *ASPLOS-XI: Proceedings of the 11th international conference on Architectural support for programming languages and operating systems*, pages 27–36, New York, NY, USA, 2004. ACM Press.

[4] R. Ghattas and A. G. Dean. Energy management for commodity short-bit-width microcontrollers. In *CASES '05: Proceedings of the 2005 international conference on Compilers, architectures and synthesis for embedded systems*, pages 32–42, New York, NY, USA, 2005. ACM Press.

[5] A. Kansal, J. Hsu, S. Zahedi, and M. B. Srivastava. Power management in energy harvesting sensor networks. *Trans. on Embedded Computing Sys.*, 6(4):32, 2007.

[6] D. B. Stewart, R. A. Volpe, and P. K. Khosla. Design of dynamically reconfigurable real-time software using port-based objects. *IEEE Trans. Softw. Eng.*, 23(12):759–776, 1997.

[7] K. Walther and J. Nolte. A flexible scheduling framework for deeply embedded systems. In *In Proc. of 4th IEEE International Symposium on Embedded Computing*, 2007.

Energieoptimiertes Scheduling für Mikrocontroller mit SDL

Marc Krämer, Alexander Geraldy

{kraemer, geraldy}@informatik.uni-kl.de
Technische Universität Kaiserslautern, Deutschland

ZUSAMMENFASSUNG

Für die Entwicklung und Einsetzbarkeit von ubiquitären Rechnersystemen und Sensornetzen spielt die energiesparende Arbeitsweise der mobilen Hardware eine elementare Rolle. Denn die heute noch zu kurze Batterielebensdauer verhindert die sinnvolle Anwendung dieser neuen Techniken noch in vielen Anwendungsbereichen. Um Energie zu sparen, ist jedoch das Zusammenspiel zwischen energiesparender Hardware auf der einen Seite und der Berücksichtigung der Hardwareaspekte in der Protokollentwicklung auf Anwendungsebene andererseits fast unabdingbar.

Wir werden im Folgenden zeigen, wie mit Hilfe eines energieorientierten Schedulers Energie gespart werden kann, ohne dass der Protokollentwickler Detailkenntnis der Hardware besitzen muss. Wir werden diesen Scheduler anhand der in der Forschung weit verbreiteten MICAz-Plattform vorstellen und evaluieren.

1. EINLEITUNG

Die Einsparung von Energie ist besonders für batteriebetriebene mobile Knoten, wie sie im ubiquitären Netzen und Sensornetzen verwendet werden, ein wichtiges Thema. Die Netzwerkknoten sollen ohne einen Eingriff, insbesondere ohne Batteriewechsel, möglichst lange arbeiten. Gleichzeitig wird versucht, die Entwicklung mobiler Applikationen auf einer hohen Entwicklungs- und Abstraktionsebene voranzutreiben, um die Entwurfskomplexität zu reduzieren. Hierbei fallen zu Verfügung stehende Möglichkeiten zur Energieeinsparung aber oft der Abstraktion zum Opfer, weil die benötigten Hardwaredetails und -funktionen durch Gerätetreiber verdeckt werden.

Diese Abstraktion durch Schichten erschwert es jedoch sowohl auf Applikations- als auch auf Treiberebene, Energieentscheidungen für den gesamten Knoten zu treffen. Es ist also notwendig, eine Integration auf einer Zwischenschicht (Betriebssystem) zu erstellen, die sowohl Informationen von der Applikation als auch Zustandsinformationen der Hardware erhält. Diese Zwischenschicht besitzt so einerseits die Information, welche Energiesparmodi verfügbar sind, und andererseits die Kenntnis, wann die Applikation einen bestimmten Energiesparzustand toleriert. Auf dieser Schicht kann also zu jedem Zeitpunkt der aktuell optimale Energiezustand für das System gefunden werden.

Wir werden nun kurz auf die zugrunde liegende Hardware und die von uns verwendeten Entwicklungstools eingehen. Anschließend stellen wir unseren energie-orientierten Scheduler vor und zeigen, dass mit ihm wertvolle Energie eingespart werden kann.

2. GRUNDLAGEN

Als Plattform für unsere Protokollentwicklung dient der MICAz, der weite Verbreitung in der Forschung gefunden hat. Für diesen MICAz wurde von uns eine Entwicklungskette entwickelt, der uns ermöglicht, Protokolle abstrakt in der formalen Spezifikationssprache SDL zu spezifiziern und automatisch, d.h. ohne weitere Eingriffe des Entwicklers, auf dem MICAz umzusetzen. Der Implementierungsschritt verläuft dadurch unmittelbar und in einer vordefinierten Weise, so dass ein direkter Zusammenhang zwischen der Protokollspezifikation und dem Verhalten des MICAz vorliegt. Die gleiche Toolkette kann verwendet werden, um weitere Plattformen, wie beispielsweise auch eine Simulationsplattform, anzusprechen.

Wir werden nun den MICAz sowie die Spezifikationssprache SDL vorstellen.

MICAz. Der MICAz [6] basiert auf dem 8 Bit-Mikrocontroller ATMega128L von Atmel [3] und dem Transceiverchip CC2420 von Chipcon [5]. Der Mikrocontroller wird mit ∼8 MHz getaktet und verwendet als zusätzlichen Zeitgeber einen externen Quartz mit 32.768 kHz. Der ATMega128L stellt verschiedene Energiesparmodi zur Verfügung. Einige der Modi können benutzt werden, um über einen Timerinterrupt zu einer bestimmten Zeit wieder aufzuwachen. In anderen Modi lässt sich der Knoten nur durch externe Interrupts, wie z.B. einen Tastendruck, wieder aufwecken [8]. Als Speicherausbau besitzt der ATMega 4 kB RAM sowie 128 kB Flash. Der Transceiver des MICAz arbeitet auf dem 2.4 MHz-Band und ist weitgehend frei programmierbar, so dass nicht unbedingt das vorgesehene ZigBee-Protokoll[2] verwendet werden muss, sondern auch eigene Protokolle entwickelt werden können. Lediglich auf die maximale ZigBee-Rahmengröße von

127 Bytes bleibt man festgelegt. Zur Einsparung von Energie stellt der Transceiver neben dem *Shutdown*-Modus noch den *Idle*-Modus zur Verfügung. Der Shutdown-Modus schaltet dabei den Transceiver vollständig aus und benötigt eine sehr lange Wiedereinschaltzeit. Im Idle-Modus ist der Wechsel in den Sende-/Empfangsmodus sehr schnell möglich, er bringt jedoch einen höheren Energieverbrauch mit sich. In beiden Modi ist kein Empfang oder Senden möglich. Nach jedem Sendevorgang wechselt der Transceiver automatisch in den Empfangsmodus zurück und zeigt damit ein ähnliches Verhalten, wie es von kabelgebundenen Halbduplexverbindungen zu erwarten ist.

SDL. Die formale Spezifikationssprache Sprache SDL [1] ermöglicht die strukturierte Beschreibung von verteilten endlichen Automaten, die über Kanäle verbunden kommunizieren. Zusätzlich lassen sich SDL-Systeme auch als offene Systeme spezifizieren, welche Signale mit ihrer Umgebung austauschen. SDL-Timer ermöglichen das Auslösen bestimmter Ereignisse zu einer vorherbestimmten Zeit. Diese Eigenschaften machen SDL zu einer hervorragende Beschreibungssprache für typische Anwendungen mit Mikrocontrollern. Zusammen mit dem Kodegenerator CMicro von Telelogic Tau [9] und der SDL Umgebungsplattform SEnF (SDL Environment Framework) [7] lässt sich aus den SDL-Systemen automatisch C-Kode generieren, der mit Hilfe existierender C-Compiler dann für die entsprechende Plattform übersetzt und ausgeführt werden kann.

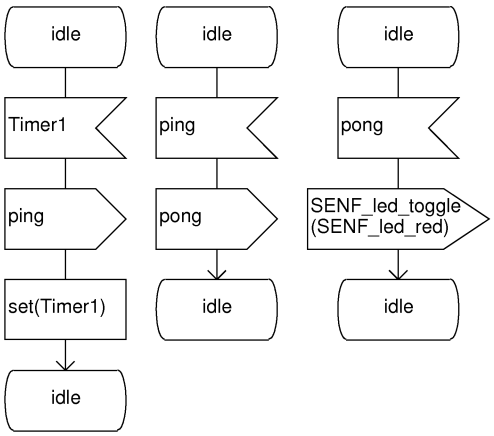


Abbildung 1: Einfaches SDL System

In Abbildung 1 ist ein einfaches (verkürztes) SDL-System gezeigt, das nach Ablauf eines Timers das Signal `ping` generiert und den Timer erneut aufzieht. Wird das Signal `ping` empfangen, wird daraus das Signal `pong` erzeugt. Bei Empfang dieses Signals wird auf dem MICAz die rote Leuchtdiode ein- bzw. ausgeschaltet.

Für die Prozessorzuteilung wird der Scheduler aus der CMicro-Implementierung von Telelogic verwendet. Dieser dient als Grundlage für die Entwicklung unseres energieorientierten Schedulers.

3. ENERGIE-SCHEDULING FÜR DEN MICAZ

Das Ziel des energie-orientierten Schedulings besteht darin, dass Protokolle energieeffizient ausgeführt werden können, ohne dass Hardwaredetails bei der Spezifikation der Protokolle einfließen müssen. Die Entscheidung über Energieressourcen soll viel mehr im Scheduler auf Betriebssystemebene getroffen werden können.

Hierzu muss die Anwendung dem Betriebssystem Details über ihr Verhalten mitteilen damit dieses Entscheidung zur Aktivierung eines Energiesparmodus geeignet treffen kann. Der Scheduler entscheidet über den nächsten zur Auswahl stehenden Prozess, kennt seine Ressourcenanforderung und weiß außerdem bei der Unterbrechung eines Prozesses, welche Ressourcen wieder frei werden. Die Verwaltung der Ressource Energie wird genau aus diesem Grund in den Scheduler integriert.

Wir werden nun am Beispiel der Hardwareplattform des MICAz unsere Erweiterung des energieoptimierten Scheduling vorstellen. Der MICAz ist hierbei nur als Stellvertreter für eine Vielzahl unterschiedlicher Hardwareplattformen zu sehen. Die an diesem Beispiel gezeigten Details lassen sich in ähnlicher Weise auch auf andere Plattformen übertragen. Zunächst werden wir auf die Hardwaredetails des MICAz eingehen, soweit sie Möglichkeiten zur Energieeinsparung betreffen. Schließlich stellen wir den SDL-Scheduler vor, sowie eine Evaluation des Schedulers.

3.1 Hardwareplattform

Systemzeit. Da SDL zeitbasierte Ereignisse (Timer) anbietet, muss von der Hardwareplattform eine Zeitbasis zur Verfügung gestellt werden. Diese Zeitbasis sollte möglichst gleichmäßig und kontinuierlich laufen. Wir verwenden für unsere Kommunikationssysteme einen interner Timer des Mikrocontrollers, der durch den externen Quartz bei 32,768 kHz getaktet wird. Die Genauigkeit der systeminternen Zeit ist damit auf $\frac{1}{32768}$ s $\approx 30\,\mu$s beschränkt. Im normalen Betriebsmodus wird der Mikrocontroller nach Ablauf der $30\,\mu$s per Interrupt unterbrochen und die Systemzeit inkrementiert.

Wegen dieses Mechanismus wäre die Energiesparphase auf $30\,\mu$s beschränkt. Atmel hat bereits vorgesehen, den Timer selbständig ein Register hochzählen zu lassen, ohne das gerade laufende Programm zu unterbrechen. Bei Erreichen einer definierten Schwelle des Zählers wird dann der Interrupt ausgelöst. Bei der 8-Bit Architektur des MICAz lässt sich die maximale Schlafzeit damit auf $\frac{254}{32768}$ s $\approx$ 7.8 ms erhöhen. Eine weitere Verlängerung der Schlafzeit wäre möglich, indem der Zeitgeber mittels Prescaler um einen Faktor verlangsamt wird. Da es während der Schlafphase zu jedem Zeitpunkt zu einer Unterbrechung durch einen externen Interrupt kommen kann und der Stand des Prescaler nicht auslesbar ist, führen diese längeren Schlafphasen jedoch zu einer Verschlechterung der Zeitgenauigkeit. Je nach Anwendungsfall ist dieses Verhalten störend oder kann im schlimmsten Fall sogar zur Fehlfunktion des Systems führen. Daher werden wir im Folgenden von einer maximalen Schlafdauer von 7,8 ms ausgehen.

Modus	Aufwach-zeit	Leistung	I/O-Clock	Inter-rupt
active	-	28 mW	X	X
idle	kurz	15 mW	X	X
ext. standby	lang	3 mW	-	X

Tabelle 1: Energieverbrauch des ATMega128L

Auswahl des Energiesparmodus. Jeder der Energiesparmodi besitzt unterschiedliche Eigenschaften. Generell gilt jedoch: Je weniger Energie der Modus verbraucht, desto weniger Funktionen stehen auf dem Sensorknoten zur Verfügung. Dadurch ist es nicht möglich, zu jedem Zeitpunkt in den Modus mit dem geringsten Verbrauch zu wechseln. In [8] wurden bereits Messergebnisse zum Energieverbrauch des MICAz veröffentlicht die wir als Grundlage verwenden. Für die Auswahl des Modus müssen zu jedem Zeitpunkt die (Anwendungs-)Anforderungen an das System bekannt sein. Wird beispielsweise die UART-Schnittstelle benötigt, kann nur noch der Idle-Modus verwendet werden. Jeder andere Energiesparmodus deaktiviert die I/O-Clock, die für die Kommunikation über die UART-Schnittstelle benötigt wird. Sobald jedoch auf dem Knoten bekannt ist, wann eine Kominikation über UART durchgeführt wird, kann der Knoten in der Zwischenzeit Energie sparen, indem er in einen Modus wechselt, der die I/O-Clock deaktiviert. Die Auswahl eines optimalen Energiesparmodus erfordert also eine sehr genaue Kenntnis der zugrundeliegenden Hardware und der Anforderungen der Anwendung. Tabelle 1 zeigt für den Prozessor einen Auszug der Funktionalität, den Energieverbauch sowie die Aufwachzeit.

Transceiver. In [8] wurde gezeigt, dass der Transceiver einer der größten Verbraucher auf dem MICAz ist. Das Hauptproblem ist hierbei, dass er im Empfangsmodus genausoviel Energie benötigt, wie im Sendemodus. Daraus folgt, dass beispielsweise eine Minimierung der Übertragungszeit keine Energieeinsparung bewirkt. Ein Energiesparmodus darf nur dann aktiviert werden, wenn keine Nachrichten von Nachbarknoten zu erwarten sind. Eine Anwendung, welche die notwendige genaue Synchronisation mit den Nachbarknoten anbietet (hier wird u.a. die bereits erwähnte genaue Zeitbasis benötigt!), und dem Scheduler die notwendige Information über mögliche Schlafzeiten anbietet, erlaubt es, drastische Energieeinsparungen vorzunehmen. Hier wird ganz deutlich, dass die Hardware oder Treiberschicht nicht ohne Anwendungs-/Protokollwissen entscheiden kann, wann welche Energiesparmodi erlaubt sind.

3.2 Energieoptimiertes Scheduling

SDL. SDL erlaubt eine Vielzahl an Auslösern zur Ausführung einer Transition. Zu diesen gehören neben den nachrichten- und timergesteuerten Transitionen u.a. auch die spontane Transition und eine bedingte ereignislose Ausführung (continuous signal). Diese beiden speziellen SDL-Konstrukte werden von CMicro aus Effizienzgründen nicht unterstützt und von uns auch bei der Optimierung des Schedulings ausgeklammert. Der Aufgabenbereich des so reduzierten SDL-Schedulers gliedert sich damit in drei Teile:

- Überprüfen, ob Timer abgelaufen sind und in Folge dessen zugehörige Prozesse aktivieren.
- Befinden sich Signale in einer Warteschlange, so werden die entsprechenden Prozesse abgearbeitet.
- Außerdem wird die Warteschlange der externen Signale auf neue Einträge überprüft.

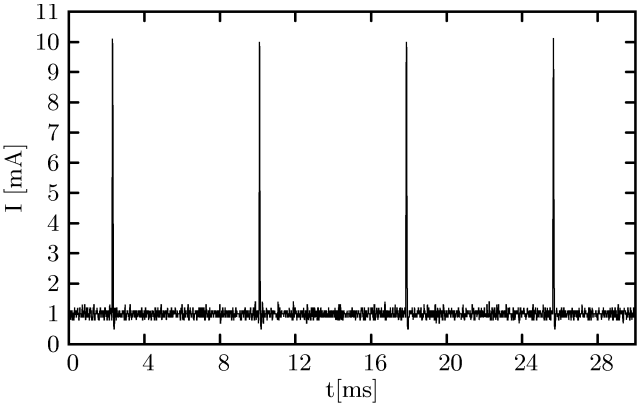


Abbildung 2: Energieverbrauch des MICAz

Der Knoten kann also bei einer leeren Signalwarteschlange bis zum Auftreten des nächsten Timers schlafen. Externe Ereignisse erfordern eine Interaktion des Prozessors, lösen also einen Interrupt aus. Die daraus resultierenden SDL-Signale werden in die externe Signalwarteschlange eingefügt. Sobald der Prozessor aus der Interrupt-Routine zurückkehrt muss er diese immer überprüfen.

Im besten Fall lässt sich die Schlafzeit auf dem verwendeten Knoten auf 7.8 ms ausdehnen. Nach dieser Zeit muss die Systemzeit weitergezählt werden und der Knoten kann ggf. weiterschlafen. In Abbildung 2 ist dieser Zyklus für ein leeres SDL-System gezeigt. Wie deutlich zu erkennen ist, beträgt der Stromverbrauch des MICAz im Betrieb ∼10 mA und im Schlafmodus ∼1 mA. Nur durch das Hinzufügen von Kommunikationskomponenten (UART, Transceiver) steigt der Energieverbrauch, ohne das sie im System verwendet werden. Deshalb wird die Modellierung dieser Komponenten im weiteren genauer betrachtet.

Paketbasierte Kommunikation (Transceiver). Statt nun die Energiemodi des Transceivers per Treiberaufruf zu steuern, was im Falle des UART so nicht umzusetzen wäre, gehen wir beim Transceiver, wie auch beim UART, den Weg über den expliziten Empfang. Dies bedeutet, dass nicht nur Sendewünsche an den Treiber mit den zu versendenden Daten weitergereicht werden, sondern auch Empfangswünsche an den Treiber geschickt werden. Der Treiber wartet ab diesem Zeitpunkt so lange, bis Daten eingehen und reicht diese dann an die Anwendung weiter. Die Zeit bis zum eigentlichen Empfang der Daten sollte dann so kurz wie möglich sein, um hier nicht unnötig Energie zu verbrauchen. Danach schaltet sich der Treiber wieder ab bis der nächste Empfangswunsch gesendet wird. Diese zwar sehr einfache, aber effektive Änderung ermöglicht es dem Treiber direkt zu entscheiden, in

welchem Modus er sich befindet und der Scheduler kann aufgrund dieser Informationen einen günstigeren Energiesparmodus auswählen. Zusätzlich sind bereits in der Spezifikation der Protokolle Übertragungszeitpunkte gekennzeichnet und können dort auch dokumentiert werden. Statische Analysen oder Simulationen des Systems können so sehr einfach Protokollfehler, die durch den Wechsel in Energiesparmodi entstehen können, aufdecken.

Serielle Kommunikation (UART). Die UART-Schnittstelle zeigt im Vergleich zum Transceiver einige Unterschiede. Es handelt sich um eine bidirektionale serielle Schnittstelle. Die Bidirektionalität der Schnittstelle drückt sich durch den weiteren Zustande „Senden + Empfangen“ aus. Die Tatsache, dass es sich um serielle Daten handelt, die kontinuierlich gesendet werden, macht den Einsatz des expliziten Empfangens zunächst unmöglich. Meist werden über dieses Schnittstelle auch Informations-Blöcke ausgetauscht. Zur Erkennung müssen entweder die Protokolle, die für die Schnittstelle eingesetzt werden, derart abgeändert werden, dass sich Blockgrenzen erkennen lassen und danach der UART auf nicht empfangen umgeschaltet. Hierfür wäre der Zugriff auf diese Protokolle nötig und die Implementierung eines Teilprotokolls würde im Hardware-Treiber erledigt. Eine andere Möglichkeit Blockgrenzen zu erkennen besteht in der Annahme, dass zwischen zwei Blöcken immer eine Pause entsteht, die länger ist, als die Zeit die sich rechnerisch aus der Übertragung zwischen zwei Bytes ergibt. Bei einer Übertragungsrate von 9600 Baud/s ist die Dauer bis das nächste Zeichen eintrifft $\sim$1 ms. Ist also nach 2 ms kein weiteres Byte eingetroffen kann man von einer Blockgrenze ausgehen. Durch diese Annahme verhält sich die serielle Kommunikation wie eine paketbasierte Vollduplexverbindung. Es kann damit aus SDL der Empfang und das Senden eines Paketes zu einem bestimmten Zeitpunkt spezifiziert werden.

3.3 Evaluation

Im Rahmen einer studentischen Arbeit [4] wurden die beschriebenen Änderungen am SDL-Scheduler von CMicro implementiert und getestet. Dabei wurden die zusätzlichen Signale der Anwendung an die Treiber weitergegeben und für den Scheduler zugänglich gemacht. Dadurch bleibt die Kapselung der Aufgaben und der Zustände weiterhin in der jeweiligen Komponente erhalten. Zusätzlich kann der Scheduler den aktuellen Zustand eines Treibers auswerten und aufgrund dieser Informationen eine Entscheidung zur Abschaltung der Komponente treffen.

Zur weiteren Evalution wurde als Anwendungsszenario ein Alarmierungssystem angenommen, wobei an den MICAz zwei Tasten angeschlossen wurden, über die sich ein Alarm an einer Basis-Station auslösen bzw. wieder ausschalten lässt. Die Basis-Station war hierbei mit einer permanenten Stromversorgung ausgestattet, wodurch die Synchronisation mit dem MICAz entfällt. Der MICAz muss sich jedoch alle 5 Minuten bei der Basis melden, um anzuzeigen, dass er sich noch innerhalb der Sendereichweite befindet. Die Basis-Station kann nach diesem Signal ihrerseits innerhalb eines Zeitfensters von zwei Sekunden Daten an dem MICAz senden, um bspw. eine der Leuchtdioden einzuschalten.

Für den Ruhezustand, also ohne Alarmauslösung, lässt sich der Energieverbrauch rechnerisch relativ leicht für den MICAz bestimmen. Bei einer Laufzeit von 11 Minuten ergeben sich 3 Synchronisationszeitpunkte zu denen der Transceiver jeweils maximal 3 Sekunden aktiv sein muss. Während der Laufzeit muss der Knoten $\sim$ 85000 mal geweckt werden, um die Systemzeit weiterzuzählen. Wenn pro Inkrement die Zeit mit 1000 Prozessorinstruktionen abgeschätzt wird, ergibt sich bei der Laufzeit die Gesamtzeit zu 11.5 Sekunden. Der MICAz verbraucht mit eingeschaltetem Transceiver 29 mA, ohne aktivierten Transceiver 9 mA und im Standby-Modus 2 mA. Damit lässt sich die gesamt verbrauchte Energie rechnerisch bestimmen:

$$\begin{aligned} 9\,\mathrm{s}\cdot 29\,\mathrm{mA} + 11.5\,\mathrm{s}\cdot 9\,\mathrm{mA} + & \\ (11\cdot 60\,\mathrm{s} - 9\,\mathrm{s} - 11.5\,\mathrm{s})\cdot 2\,\mathrm{mA} & = 1643.5\,\mathrm{mAs} \\ 1643.5\,\mathrm{mAs}\cdot 3.1\,\mathrm{V} & = 5095\,\mathrm{mJ} \end{aligned}$$

In diesem Szenario würde sich der maximal mögliche Energieverbrauch zu

$$11\cdot 60\,\mathrm{s}\cdot 29\,\mathrm{mA}\cdot 3.1\,\mathrm{V} = 59334\,\mathrm{mJ}$$

ergeben. Der minimale Energieverbauch für obiges Szenario kann um $11.5\mathrm{s}\cdot 9\,\mathrm{mA}\cdot 3.1\,\mathrm{V} = 320\,\mathrm{mJ}$ geringer ausfallen als das Ergebnis des vorgestellten Schedulers. Es kann hierbei also lediglich das Inkrementieren des Timers entfallen, da in diesem Szenario die Genauigkeit des Timers nicht nötig wäre.

4. ZUSAMMENFASSUNG UND AUSBLICK

In dieser Arbeit wurde die Funktionsweise eines energieoptimierenden Schedulers für SDL am Beispiel der MICAz-Plattform vorgestellt. Durch die Einführung von Signalen, die dem Betriebssystem Kommunikationszeitpunkte anzeigen, ist es nun möglich, auf Betriebssystemebene Entscheidungen für die Einsparung von Energie zu treffen, welche die Anforderungen der Anwendung einbeziehen. Die automatisierte Einsparung von Energie ist besonders für dynamische und große Systeme eine unabdingbare Hilfe. Während die Spezifikation der Anwendung weiterhin hardwareunabhängig erfolgen kann, bietet der energieorientierte Scheduler eine automatische Ausnutzung möglicher Energiesparmodi und verhilft damit der Anwendung zu einer drastisch verlängerten Laufzeit.

In Zukunft soll der hier vorgestellte energieorientierte Scheduler um weitere Aspekte erweitert werden, die zunächst weitere Energieeinsparungen zur Folge haben. In Zunkft soll es aber auch möglich sein Abschätzungen über den Energieverbrauch des laufenden Systems anzugeben und damit die Laufzeit sehr exakt vorrauszusagen. Diese Voraussage könnte genutzt werden, um kostbare Rechenleistung sinnvoll auf Prozesse verschiedener Priorität zu verteilen oder Arbeiten auf energiestärkere Rechnerknoten im Netzwerk zu verteilen.

5. REFERENZEN

[1] CCITT Specification and Description Language (SDL), 1995. ITU-T Recommendation Z.100 (03/93) + (10/96) + (1998).

[2] IEEE standard 802.15.4-2006. `http://standards.ieee.org/getieee802/download/802.15.4-2006.pdf`, 2006.

[3] Atmel. Datasheet ATMega128L. http://www.atmel.com/dyn/resources/prod_documents/doc2467.pdf, 2006.

[4] A. Becker. Entwicklung eines Energie-Schedulers für SDL-Systeme. Projektarbeit, Technische Universität Kaiserslautern, 2006.

[5] Chipcon. Chipcon CC2420 datasheet. http://www.chipcon.com/files/CC2420_Data_Sheet_1_3.pdf, 2005.

[6] Crossbow. Crossbow datasheet on MicaZ. http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICAz_Datasheet.pdf, 2006.

[7] I. Fliege, A. Geraldy, S. Jung, T. Kuhn, C. Webel, and C. Weber. Konzept und Struktur des SDL Environment Frameworks (SEnF). Technischer Bericht 341/05, Fachbereich Informatik, TU Kaiserslautern, 2005.

[8] M. Krämer and A. Geraldy. Energy Measurements for MicaZ Node. In *5. GI/ITG KuVS Fachgespräch „Drahtlose Sensornetze“*, page 61ff. Pedro José Marrón, 2006.

[9] Telelogic AB. Telelogic Tau SDL Suite Homepage. http://www.telelogic.com/products/tau/sdl/index.cfm, 2006.

Energy Accounting Support in TinyOS

Simon Kellner
kellner@ira.uka.de

Frank Bellosa
bellosa@ira.uka.de

System Architecture Group
Universität Karlsruhe

ABSTRACT

Energy is the most limiting resource in sensor networks. This is particularly true for dynamic sensor networks in which the sensor-net application is not statically planned. We describe three components of our energy management system for nodes in such dynamic sensor networks: A flexible energy model and an accounting infrastructure for making sensor nodes energy-aware, and Resource Containers for managing the energy accounting information.

1. INTRODUCTION

Energy still is the most critical resource in sensor networks. Current energy supplies already take up most of a sensor node's space, but can provide the desired node lifetimes of years only when sensor-net application designers give a high priority to a long sensor-net lifetime. Sensor-Net Operating Systems (OSes) like TinyOS [2] encourage energy saving by not providing a convenient CPU-abstraction such as threads, which could, for example, tempt application developers into creating CPU-intensive waiting loops and thus into wasting energy.

Database interfaces to sensor nets like TinyDB [5] make it easy for users to retrieve sensor data: A sensor-net application is formulated as a request in an SQL-like language and interpreted by the sensor network until the request expires. The program on the sensor nodes only needs the ability to interpret and execute such requests. This eliminates the need to reprogram sensor nodes and allows multiple queries to be processed simultaneously.

Such dynamic systems can support multiple users in a sensor net, each with his own set of queries. In this scenario it is desirable to account the energy consumption of each query, e.g. to bill users based on their sensor-net usage, or to find the query with the highest energy consumption and cancel it before it wears down the energy supplies.

In this paper we describe several parts of a solution to energy management on sensor nodes that addresses multi-user dynamic sensor networks. The presented solution is currently being implemented for MICAz nodes in TinyOS 2. First we describe our energy model for a sensor node, then the infrastructure used in taking measurements and accounting the energy. As a third part, we describe the concept of Resource Containers, which will be employed to fairly distribute the accounted energy in our dynamic sensor-net setting.

2. RELATED WORK

The management of energy in sensor networks has received a significant share of research over the last years, as it concerns the primary resource of such networks.

PowerTOSSIM[6] is perhaps the approach most similar to our own model and accounting infrastructure. It instruments OS components or simulations thereof to track power states and uses an energy model to compute energy consumption for one or more sensor nodes. However, there is no implementation for current versions of TinyOS. Its energy model only considers hardware states, not the transitions in-between. The most significant difference is in the intended use: Our instrumentation and model are designed to be used in on-line energy accounting as opposed to off-line simulation.

AEON[4] is the energy model used in the AVRORA[7] simulator. It models the hardware states of a MICA2 node. Our model is based primarily on the MICAz node and additionally considers transitions between hardware states.

Energy measurements of a MICAz node can be found in [3]. The measurements of the AtMega128 controller are detailed, but the measurements of the ZigBee controller (CC2420) severely lack details. Our measurements show a real difference between the listen and the transmit state, regardless of the programmed output power in the latter.

Resource Containers are an OS-abstraction introduced by Banga, Druschel and Mogul [1] in 1999 for accounting on web-servers and consist basically of OS-provided storage for accounting data. The idea is to separate OS abstractions for CPU and resource accounting, so one can base accounting on other, more suitable abstractions. In the PC world, for example, Resource Containers (RCs) give administrators and users the ability of accounting all activity connected to a user request, which usually has a higher significance than process-based accounting.

3. ENERGY MODEL

Our sensor node energy model is designed to be used both for off-line simulations and on-line accounting. To this end, the model is specified in a more formal manner than usual.

Our energy model is based on finite state machines that closely model the hardware's power states and transitions. To account for the concurrency possible on typical hardware, each subsystem on a sensor node is modeled by its own

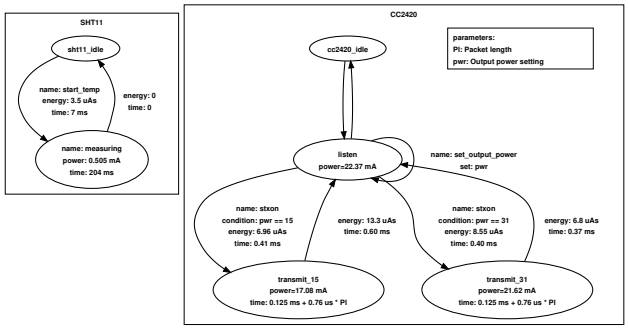


Figure 1: Energy models for the CC2420 and SHT11 chips

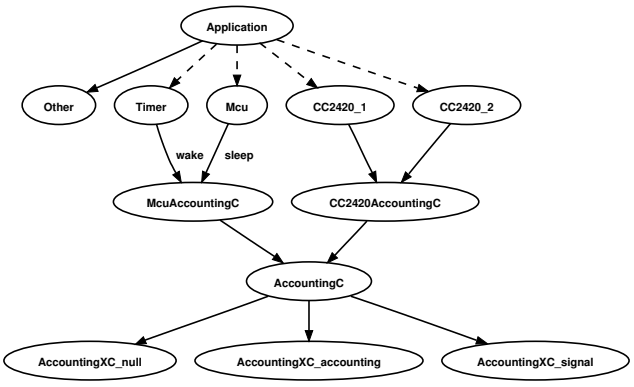


Figure 2: Accounting infrastructure overview

state machine. For example, the OS can finish instructing the radio controller to send a packet, start taking a measurement on a sensor, and get interrupted by a preset timer. In this example, the radio controller, the sensor, and the microcontroller each are modeled by one finite state machine.

States and transitions of these machines are attributed with their physical characteristics such as time (duration), energy, or power. These characteristics often depend on parameters from outside such as the battery voltage or packet length. Values for these parameters have to be supplied by each program that uses this model.

The model states describe the hardware states in as much as they can be distinguished by their power consumption. A small part of such an energy model is shown in Figure 1. Simple hardware like sensors or LEDs can be modeled by a small number of states. The model for the SHT11 chip in Figure 1, for example, covers the whole process of measuring the temperature. For other chips like the ZigBee controller CC2420 this approach can result in multiple *transmit* states, one for each selectable transmission power. Of the 32 available and 8 documented power settings, only two are shown in Figure 1 due to space constraints. Furthermore, the transitions from *cc2420_idle* to the transmit states have been omitted to avoid cluttering.

Transitions in the model describe the time and energy spent on changing the hardware state. A transition can be named or unnamed. Unnamed transitions are used for state changes which are predictable from the hardware layer. For example, upon completing a transmission the CC2420 controller automatically switches back into the *listening* state where it could then receive an acknowledgment message. The time spent transmitting is known from the length of the packet, which is known, as a packet must be stored in the CC2420's transmit buffer prior to transmission. Named transitions, on the other hand, describe changes that are not predictable from a hardware viewpoint, e.g., when the software sends a command to the radio chip to transmit a packet that is currently in the chip's buffers. In the example, *start_temp*, *set_output_power* and *stxon* are of this type.

Some models are further equipped with parameters that can be used to reduce the number of model states or to calculate energy consumption. The packet length parameter in Figure 1 determines the time the CC2420 spends transmitting, so the energy consumed for the transmission directly depends on the packet length. The other parameter, output power setting, is not strictly necessary for modeling. Its primary use is to reduce the number of states in the model: Without it, the number of states would almost triple: As the output power can be programmed in the *idle* and *listen* states, there would have to be 32 versions of both states with bidirectional transitions between all versions of a state. So not only would the number of states nearly triple, the number of transitions would grow quadratically with them.

4. ACCOUNTING INFRASTRUCTURE

Information required to use the model for on-line energy accounting has to come from various places deep inside a sensor node operating system. Usually, this means places where a driver issues commands to or receives interrupts from the hardware it manages. Due to the high degree of modularization in TinyOS 2, the relevant code places for one hardware chip could be scattered over several directories. The purpose of our accounting infrastructure is to collect this information in a consistent manner.

The process of gathering the accounting information should also be transparent to the sensor-net application. This allows developers to base energy-aware applications on traditional, statically planned applications without the need to change a lot of code.

Energy-relevant code fragments are instrumented by an additional function call. These function calls are routed first to a module for the subsystem and then to a central module named `AccountingC`, as shown in Figure 2. Here they arrive as (*component*, *event*) pairs, where *component* refers to a finite state machine of the energy model, and *event* to a named transition in this machine. The central module then uses a back-end to process the gathered information. The choice of back-end is configurable from the command line at compile time. The default back-end (`AccountinXC_null`) simply discards the information. Thus, in spite of introducing these additional function calls in the source code, this does not result in additional object code, as the compiler inlines all involved functions.

Aside from the actual accounting process, the infrastruc-

ture has shown its flexibility in combination with the TinyOS build system by providing convenient options to indicate the energy-relevant events using a GPIO pin to the measurement hardware outside. For this scenario, the central accounting component is given a different back-end module (`AccountingXC_signal`) to use, which in turn uses a platform-dependent signaling module to access a GPIO pin. This proved to be very helpful for extracting the physical characteristics of our energy model from measurements of sensor nodes. Parts of the data analysis could be automated, as phase lengths and transitions can be detected more easily by these additional signals.

5. RESOURCE CONTAINER

To make energy accounting work not only on a single node but in a large network, we employ the concept of Resouce Containers (RCs). RCs can be used to account energy consumption for each query being executed on the node and aggregate this information throughout the network.

In the following we assume that the sensor-net application responds to user-generated queries and that all packets related to one query carry the same unique ID.

5.1 Normal Resource Containers

A normal RC is associated with a query. As soon as an application learns the ID of the query currently being processed, it informs TinyOS that it wishes to switch to the RC associated with this query. The selected RC is then bound to the current TinyOS `task`.

The energy consumption of all further activities coming from this TinyOS `task` is accounted to the selected RC. If the TinyOS `task` posts a new TinyOS `task` or sets up a new timer, this binding can be stored by the scheduler or timer system, and can be used to switch back to the stored RC automatically on the corresponding wake-up call.

The OS here clearly depends on the application for correct accounting, but this is both feasible and necessary in a sensor-net application. It is necessary to prevent producing hard-to-maintain code, and it is feasible because there should be only few places where this RC-switching occurs, namely when a sensor-net application starts processing a query.

5.2 Anonymous Resource Containers

Since TinyOS applications spend most of their time sleeping and perform only minimal amounts of processing, energy consumed during interrupt handling is not negligible. For example, a timer interrupt may cause the activation of a communication device, which is subsequently used to send stored sensor data to other nodes. The sensor node is not aware of the query ID until it accesses the packet it is about to send. In the meantime, the activation of the communication device can consume a substantial amount of energy that cannot be assigned to the correct RC at that moment.

As a solution, the interrupt handler can allocate a temporary, anonymous RC and use it to account both its own energy consumption and the device activation. Later, when the application becomes aware of the query ID, it can switch to the RC associated with the query, causing the temporary RC to be merged and released.

5.3 Special Resource Containers

It may be necessary to employ special RCs to provide additional information or to handle cases where the correct RC is not known.

5.3.1 Root Resource Container

One RC worth mentioning is the RC for the whole node. It is used to collect the amount of energy consumed by the whole node, regardless of queries. This information is of interest to the nodes themselves in order to estimate the amount of remaining energy. It can also be regarded as another data source and can itself be the target of a query.

5.3.2 Idle Resource Container

Some energy consumption can not be clearly accounted to a query, e.g., the energy spent during sleep (*idle energy*). We call the problem of accounting this energy consumption in a fair manner *accounting fairness*.

One way to address the issue of idle energy accounting is to distribute the accounted idle energy among all queries known to the sensor node. To achieve this, a special RC for this energy class is present in the system. At certain times, this RC is cleared and its content distributed among all existing normal RCs. This has to be done both periodically and on creation/expiration of a query:

- Periodically so that the accounting information remains recent.
- At query instantiation to avoid penalizing this query by accounting sleeping energy spent before its instantiation.
- At query expiration to avoid losing accounted energy.

The fairness of this distribution is subject to discussion and thus should be handled by a project-dependent policy. Policy examples include equal distribution and partitioning according to duty-cycle or used energy.

Concluding, one can picture the RCs in a 3-level hierarchy: the root RC for the node, named RCs for the queries and anonymous RCs to account energy consumed for a (yet) unknown purpose. In this hierarchy, the root RC contains the aggregated accounting data of the named RCs, while the anonymous RCs will eventually be merged with one of the named RCs.

5.4 Shared Data

Caching the acquired sensor data introduces another instance of the accounting-fairness problem. Without additional measures, the first query to sample data bears the cost of acquiring it, subsequent queries can use it at almost zero cost. If the accuracy of timing or accounting can be relaxed, some trade-offs between one of them and accounting fairness can and should be considered.

A trade-off between timing accuracy and accounting fairness can be implemented as a subscriber model for sensor data: The sensor data is sampled either on time-out after the first subscription or when enough parties subscribed to this sensor data. The energy is split among all of the subscribed parties.

A trade-off between accounting accuracy and fairness can be implemented by assigning a value to the sampled sensor data that decays with every access. For example, the initial query bears 3/4 of the costs, the next query 3/4 of the remaining costs, and after a time-out, the rest is distributed across all queries that acquired this data.

5.5 Resource Container Aggregation

The usefulness of Resource Containers becomes apparent when they are shared between all network nodes. With the collected information in these RCs it is possible to account the energy consumption of the whole network individually for each query.

RCs lend themselves quite naturally to sensor nets with dynamically created queries. When receiving a new query, a sensor node allocates an RC for this query, accounts the query's energy costs to that RC and sends the accounted data back together with the responses to this query.

RC contents can easily be aggregated by summation over all RCs with the same query ID. The design of RCs to store all of the energy accounted to it since its creation makes it resilient to occasional packet loss. When accounting information is lost in the network due to occasional packet loss, the aggregated accounting information at the data sink may be incorrect, but it will be correct again after reception of the next packet.

To allow the data sink to compare aggregated RC values from different queries and to detect packet loss, a node should additionally send the number of sensor nodes involved in an aggregate, if this information is not already present in the aggregated sensor data.

6. CONCLUSION

In sensor networks with dynamically created queries, on-line energy accounting is necessary to make the network energy-aware. We presented three parts of an on-line energy accounting solution currently being developed for TinyOS 2. An accounting infrastructure uses an energy model to make each sensor node energy-aware. Resource Containers allow to extend this accounting mechanism to cover the whole network. Together, this is an energy management solution for multi-user dynamic sensor networks.

7. ACKNOWLEDGMENTS

This work is done as part of the BW-FIT project ZeuS.

8. REFERENCES

[1] G. Banga, P. Druschel, and J. Mogul. Resource containers: A new facility for resource management in server systems. In *Proceedings of the Third Symposium on Operating System Design and Implementation (OSDI'99)*, Feb. 1999.

[2] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. In *ASPLOS-IX: Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 93–104, New York, NY, USA, 2000. ACM Press.

[3] M. Krämer and A. Geraldy. Energy measurements for MicaZ node. In *5. GI/ITG KuVS Fachgespräch „Drahtlose Sensornetze“*, number 2006/07, pages 61–68, Universität Stuttgart, Institut für Parallele und Verteilte Systeme, July 2006.

[4] O. Landsiedel, K. Wehrle, and S. Götz. Accurate prediction of power consumption in sensor networks. In *Proceedings of The Second IEEE Workshop on Embedded Networked Sensors (EmNetS-II)*, May 2005.

[5] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 491–502, New York, NY, USA, 2003. ACM Press.

[6] V. Shnayder, M. Hempstead, B. 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*, pages 188–200, New York, NY, USA, Nov. 2004. ACM Press.

[7] B. L. Titzer, D. K. Lee, and J. Palsberg. Avrora: scalable sensor network simulation with precise timing. In *IPSN '05: Proceedings of the 4th International Symposium on Information Processing in Sensor Networks*, page 67, Piscataway, NJ, USA, 2005. IEEE Press.

Analysis of the Authenticated Query Flooding Protocol by Probabilistic Means

Peter H. Schmitt, Frank Werner *
Universität Karlsruhe (TH)
Institut für Theoretische Informatik
{pschmitt,werner}@ira.uka.de

ABSTRACT

Secure multicast communication is a elementary mechanism in the field of wireless sensor networks, addressing a number of security means and algorithms. This paper analyses the energy consumption of the algorithm of authenticated query flooding as proposed by [1] but the applied technique can also be used to a more general probabilistic flooding paradigm. The verification results are obtained by means of the probabilistic model checking tool PRISM. We measure in our analysis the impact if only a portion of all packets stem from an adversary that needs to be hindered from reaching the whole network. What is new when choosing the concept of formal methods and what differs it from the simulation based analysis is that the results are precise and need no confidence level since all paths which lead to an observation are consequently expressed in an expectation value rather than an on-average value with confidence levels.

1. INTRODUCTION

Wireless sensor networks form a very active research topic in Computer Science with a variety of novel applications in a great numbers of diverse areas. For potential applications that transmit valuable or critical data security issues will play an important role. This has triggered investigations into how wireless sensor networks – which are particularly vulnerable to intrusion by an outside attacker – can be protected against malicious or accidental manipulations.
Communication in wireless sensor networks is characterised by the fact that no centralised knowledge about the identity, reachability, location or functionality of the individual nodes is available. Given these constraints *flooding* has become an accepted communication paradigm despite its high energy cost.
We will concentrate in this paper on the scenario that a base station, e.g., a laptop-like device spreads information by flooding authenticated queries to a wireless sensor network. An adversary may post illegitimate or fake queries disrupting or compromising the network. A probabilistic authentication protocol for *authenticated query flooding* has been proposed in [1] to limit the propagation of fake queries. The influence of various protocol parameters on the strictness of this limitation has been investigated by analytical computations and network simulations. In this paper we complement this analysis by estimations on average energy consumption under various parameter settings. This should provide us with valuable information in which situations an implementation of this algorithms is feasible in practice.

We use the PRISM tool – a probabilistic symbolic model checker [2, 5] – for our investigations. This might not be the tool that immediately suggests itself for the job at hand and we do believe that a comparable analysis could have been accomplished by a state-of-the-art simulation tool like ns2 [6]. At this point it is important to highlight, that we do not obtain the presented results by simulation. Rather the values are computed by verification and the inherent exploration of the complete state space, i.e. all possible interleavings of all runs.

What made PRISM attractive from our point of view is its solid foundations on the theory of Markov chains and its comfortable graphical user interface. There is no denying that the size of the Markov chain models accessible to analysis by PRISM is somewhat restrited in the number of module instances that can be carried out simultaneously.

This work is organised as follows. In section 2 we quickly review *authenticated query flooding* from [1], giving some idea of the algorithm and its main parameters. In section 3 we analyse the models with the PRISM tool [5] and explain the parameter influencing the model. Exemplary scenarios are considered and investigated i.e. probabilistic results like *"How much energy is used by reaching N nodes with a faked query?"*. This part also tries to bridge the gap to an practical energy critical example as quested by the ZeuS project where energy and gradual security is considered a central topic. The section is closed by giving an optimal security/energy tradeoff depending on an adversary, and the expected times until the network is flooded. In the end in section 5 the whole matter is wrapped up, giving an outlook for possible improvements and finally concluding with suggestions for further energy related exploration.

2. THE AUTHENTICATED QUERY FLOODING ALGORITHM

The authenticated query flooding (AQF) algorithm proposed in [1] assumes an ID-based key predistribution, see e.g.,[8]. Out of a pool of keys numbered from 1 to ℓ every node receives a ring of k randomly chosen keys. The way this predistribution may be organised is sketched in [1]. When the base station wants to flood a query q it first computes

*This research position is funded by the BW-FIT Project ZeuS (Zuverlssige Informationsbereitstellung in energiebewussten ubiquitren Systemen).

the value $x = h(q)$ of some given hash function h and then uses x as a seed to compute m pseudo random numbers $(kid_1, \ldots, kid_m)$. These numbers are interpreted as the numbers of keys $(k_{kid_1}, \ldots, k_{kid_m})$ from the pool. These keys are used to compute m message authentication codes (MACs). We stick to the design decision from [1] to use 1-bit MACs. Thus, using key k_{kid_i} on $x = h(q)$ the bit m_i is computed. The sequence $(m_1, \ldots, m_k) = macs(q)$ is called the *authenticator* for q. The base station then floods query q together with $macs(q)$ into the sensor network.

Upon receiving a query q and the authenticator $macs(q)$ a sensor node, which is assumed to share the hash function h and the pseudo random number generator with the base station, computes the indices $(kid_1, \ldots, kid_m)$ used to encode $macs(q)$. If for at least one of the keys k_{kid_i} that also belong to its own key ring it detects a mismatch between the computed and the received value m_i, it does not forward the query. In all other cases it sends it to all its neighbours. As in [1] we are only interested in the analysis of the flooding algorithm and ignore the situation when a sensor nodes considers a query as genuine and replies to it. Furthermore, the node memorises processed queries and immediately ignores them when receiving them for a second time. Obviously, a legitimate query q will be received by all reachable nodes in the network.

The adversary model adopted in [1] assumes that an attacker can feed messages plus authenticators into the network in the same way as the base station does. It is furthermore assumed that an adversary may *capture* sensor nodes and get hold of their keys. It thus may start flooding a query q using the correct MAC-bits for the keys it has captured, and randomly guesses the remaining authenticator bits. Assuming that an attacker has captured $\tilde{n}$ nodes using the theory of random sets the average number $\tilde{b}$ of keys known to the adversary and the expected value B of correct MAC-bits in a fake query authenticator can be computed. The probability p_f that a sensor forwards a query with a fake authenticator is according to [1] given by the formula:

$$p_f = \left(\frac{\ell - k}{\ell} + \frac{k}{\ell}\frac{B}{m}\right)^m = \left(\frac{\ell - k}{\ell} + \frac{k}{\ell}\frac{\tilde{b}}{m} + \frac{1}{2\ell}\right)^m$$

It is an essential contribution of [1] to suggest a criterion for choosing plausible values for p_f. Table 1 shows typical parameter values that we investigate with the corresponding probability p_f. In the experiments considered later in this paper we will use topologies with densities varying between 2.3 and 4.1 nodes and forwarding probabilities between 0% and 45% since the algorithm is working very efficient within this region.

Variable	Value range	Description
n	12 - 14	number of nodes in the sensor network
ℓ	1 000-10 000	number of keys in the key pool
k	50-250	number of keys in the key ring of a node
$keylen$	128	length of one key
m	100-500	size of the authenticator
MNKK	-	mean number of keys that a sensor has to validate per query
$data$	8	data bits
d	2-4	network density
$\tilde{n}$	≥ 1	Number of captured nodes
$\tilde{b}$	-	number of captured keys
$E_{\tilde{b}}$	-	number of keys in the authenticator known by the adversary
B	-	number of right bits in the fake authenticator
p_f	-	probability that the message will be forwarded

1: Annotation for the variable meanings and parameters for the AQF algorithm that contribute to the forwarding probability of p_f, and resulting energy. Blank fields depend on the setting and need individual computation.

3. ENERGY CONSUMPTION IN WSN

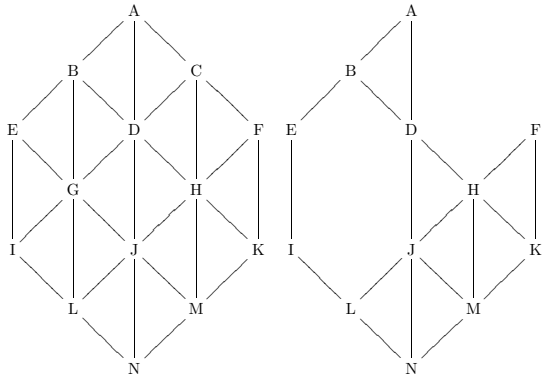
This section aims at providing scenarios and measures that can later on be beneficially used for building an actual wireless sensor network based on authenticated query flooding. Five different topologies of sensor networks (cf. Fig. 3) are introduced that will be analysed under the objective of power consumption until the request terminates. We assume legitimate and fake queries to be injected into the network at node A although any other point is feasible, and could be done in future work. Additionally an adversary might use 2 or more nodes to inject the faked query which is admissible but not covered here and kept as a possible future research topic.

In computing the energy balance all sensors from the network will be taken into account. The networks that we investigate are strongly *interlinked* and consist of up to 18 sensor nodes. The reason for picking symmetric topologies is that we think that results scale to even bigger network. In Figure 1a nodes have at most six neighbours. A variation of this network setup is topology 8 with two missing nodes (cf. Figure 1b). A *check-box-like topology* with 12 sensors is present in Figure 1c, where each sensor node has at most four neighbours. A *hexagon-like structure* presented in Figure 1d with each node having at most 3 proximate nodes. And finally the asymmetric topology from Figure 1e with a more realistic shape and 18 nodes will conclude the study.

By modelling these wireless sensor networks as a Discrete Time Markov Chain (DTMC) we do a reward analysis [4] within the probabilistic model checker PRISM. As such we use a sensor node specific energy consumption function to formulate the energy constraints within the model. The later analysis will reveal how parameters change when dealing with different topologies, and how the individual characteristics under the deployment of a probabilistic flooding manifest and can be compared.

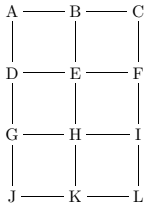
3.1 The Reward Model

The reward model is attaching costs to state transitions in a way that makes them computable for the prism tool. We consider a sensor receiving if it receives a packet by any of the gadgets in its vicinity. After reception it is computing and validating the 1-bit MACs for which it need energy

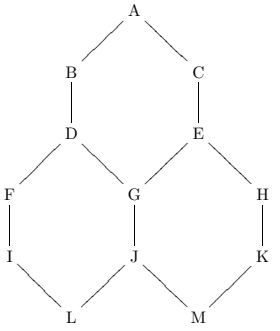


(a) Topology 7

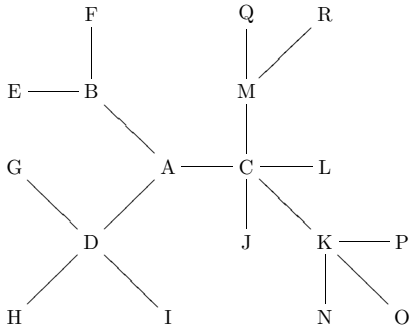
(b) Topology 8



(c) Topology 9



(d) Topology 10



(e) Topology 11

1: problem topologies

E(RC) including the powerrequirements for receiving. After the authenticator is computed the node does either attribute the query as non-fake, distributing it to vicinity nodes in which case a total energy draw of E(RCS) is needed. In case that the authenticator turns out to be faked, the packet is dropped without further transmission.

In the following analysis we use networks of TMote Sky sensors manufactures by Moteiv[7]. The gadgets will be arranged in topologies as illustrated in figure 3. This is rational since we expect to find an overview of how queries are processed using a probabilistic flooding mechanism, and to which extent the choice of the topology influences the security/energy ratio. The approach presented here tries to obtain a theoretical approximation of a real-world scenario by means of Markov chains with all the pros and limitations this involves.

As input parameters for our model (cf. Table 1) we use a 1 byte data packet. The key length is selected as 128 bits which seems to be a sound value considering an available memory of 8kB and the relatively high security level induced hereby. As varying parameters we choose the total number of keys in the keys pool ℓ between 1 000 and 10 000 appropriate for small networks, the size of the authenticator m between 100 and 500 and the number of keys with which the sensor nodes are preloaded, denoted as k to be within the range of 100 and 250. Underlying the model we assume that the adversary only knows the keys deployed on a single sensor node (k valid keys), since we have a small size network with devices more or less out of a burglar's reach. For approximating the average number of hashes a node has to validate, we use the figure *mean number of known keys* by a sensor (MNKK) which depends on the variable input parameters and is computed as:

$$MNKK = \frac{m \cdot k}{\ell}$$

As cryptographic hash function we use the MD2 (Message Digest Algorithm)[3] since it seems to be the best choice when dealing with 8-bit micro processors and a key length of 128 bit. Using these numbers we obtain a varying probability p_f (that a sensor accepts the query with a fake authenticator) which is used as an input parameter for our model. An example for varying the authenticator size m while ℓ, and k, are kept constant is illustrated in Table 3.1 below.

m	MNKK	pf	E(RC)	E(RCS)
-	-	-	0.1723	0.3355
100	1.6767	0.439	0.2574	0.4857
150	2.5	0.291	0.2999	0.5608
200	3.33	0.193	0.3425	0.6359
250	4.17	0.128	0.3850	0.7110
300	5	0.085	0.4276	0.7862
350	5.83	0.056	0.4701	0.8613
400	6.67	0.037	0.5127	0.9364
450	7.5	0.025	0.5552	1.0115
500	8.33	0.016	0.5978	1.0866

2: Power usage for the TMote Sky sensor node in mJ for receiving E(R), computing and comparing the hash values E(RC), and sending E(RCS) for a 8 bit data packet and a 128 bit keylength. The first line is without the AQF algorithm, for the remaining entries the total number of keys is fixed to $\ell = 6000$, the number of keys on each sensor is $k = 100$. Parameter MNKK is representing the *mean number of keys known* by a sensor node and the authenticator parameter m is varying.

For obtaining the expected energy use we finally question our model with the PCTL query as follows:

$$R =? \quad [\quad F \quad (\text{"}deadlock\text{"}) \quad]$$

Although the term "deadlock" might be misleading at this point, it is defining the appropriate state within our model, in which all queries are processed and no further step is possible. Whether is happens due to dropping of the queries

or due to the fact that the whole network is flooded needs no further specification here.

3.2 Energy Use

In the following we briefly sketch the energy requirements influencing the reward model of our sensor network. We expect the processor to run at 8MHz which corresponds to 6 million instructions per second. One byte data packet plus the size of the authenticator is assumed as being the payload.

The energy draw of the radio controller integrated in the board is assumed to be 5.9 mW for receiving, and 5.6 mW for sending which can be drawn out of the data sheets. The micro controller has a power need of 6 mW, resulting in 6 million operations per second (MIPS) total. For moving data from the radio controller to the CPU and vice versa an energy amount of 6.5 mW is needed, since transmission and CPU have to be switched on. All power needs relate to a packet of size 8 bit, a 128-bit key length, and need separate computation when increasing the authenticator size.

When looking at times that we need for computing the operation dependent power, we obtain for a simple reception of a packet of size 108 bit (100 bit authenticator and 8 bit data) the receiver needs 0.035 ms per byte, and in addition to 0.5 ms for initialisation, that sum up to 3.6475 ms per payload. Loading of the data from the radio controller to the CPU requires 0.05 ms per byte plus a constant time of 2 ms. For hashing the data 3 732 block instructions are required using the MD2 hash algorithm that runs in approximately 0.0075 ms. The hashes can be validated by the sensor node in 0.01859 ms. For transmission of the data, times equal to the times for receiving are needed.

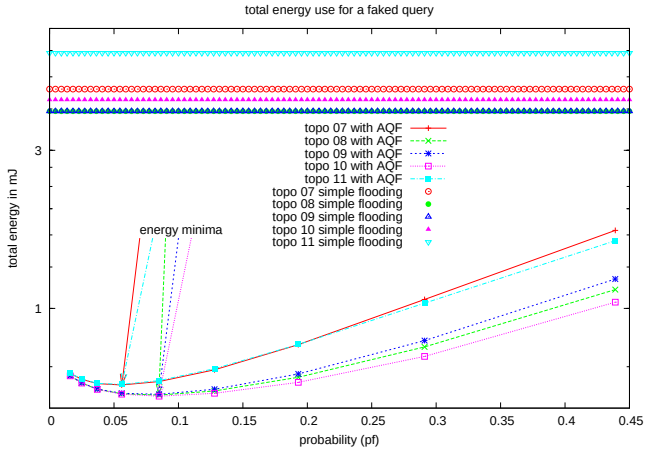
Some emerging energy figures that corelate with the increasing authenticator m are displayed in table 3.1 for the Tmote Sky board. All sensors are either in receive mode during the flooding procedure or try to authenticate the received query. Upon successful authentication, and in case of not being able to authenticate the data packet, they forward the query to neighbouring sensors. Since we want to limit the propagation of fake queries to a small part of the WSN, we consider a forwarding probability of faked packets p_f below 45%. For most topologies. In addition to figure p_f we compute the energy used without the AQF procedure, that is energy required for flooding a sound packet thru the network. During these experiments a query is received and spread to surrounding sensors without any involved computation.

4. ENERGY RESULTS

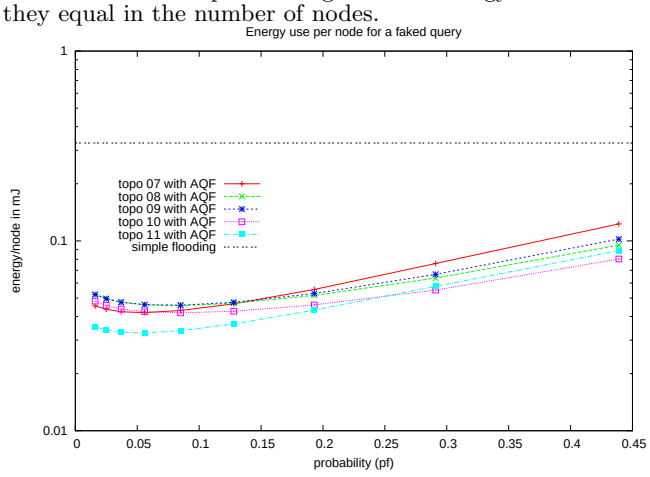
At this point it is necessary to point out, that when doing the reward rate computation the output for different settings are not easily to compare since the topologies vary e.g. in the network density, the total number of nodes in the network topology etc. To account for this, we compute in addition a correlating average rate per node, composed of the respective reward rate energy, and the number of nodes of the network. When doing so, we have in mind that sensor nodes do not power out evenly. Especially nodes close to the base station are expected to spend more power than the others, located further away.

p_f	m	TOP7	TOP8	TOP9	TOP10	TOP11
1	-	4.5924	3.9363	3.9363	4.2643	5.9045
0.439	100	1.7202	1.1398	1.2267	1.0440	1.6013
0.291	150	1.0641	0.7649	0.7997	0.7162	1.0370
0.193	200	0.7773	0.6196	0.6342	0.5981	0.7780
0.128	250	0.6527	0.5633	0.5700	0.5541	0.6575
0.085	300	0.6016	0.5474	0.5506	0.5434	0.6053
0.056	350	0.5871	0.5526	0.5541	0.5508	0.5894
0.037	400	0.5929	0.5701	0.5709	0.5693	0.5941
0.025	450	0.6109	0.5955	0.5958	0.5951	0.6115
0.016	500	0.6366	0.6261	0.6262	0.6259	0.6369

3: Energy in mJ for flooding different network topologies. The first line indicated the power use without AQF algorithm.



(a) Total energy use for a faked packet. Note that topology 8 and 9 have for simple flooding the same energy need since they equal in the number of nodes.



(b) Average energy use per node, which is equal for the simple non-authenticated version of the flooding algorithm.

2: Energy rewards for selected topologies for 100% fake queries.

Figure 4 illustrates different topologies with varying parameter p_f denoted on the x-axis and the involved energy requirements on a logarithmic scaling in mJ. Horizontal lines represent the power need of nodes without the securing mean, i.e. for simple flooding which is independent of the forwarding

probability p_f. The curves for the network topologies with varying energy are leftward curved with an upward slope and a global energy minimum. These minima depend on the topology and vary with the probability of forwarding a fake packet between 5% and 15%. Results are displayed in Table 3 with the pertinent authenticator size m and mean number of keys that a node knows.

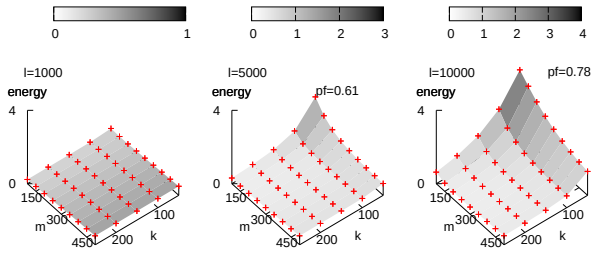
With increasing p_f the costs for security drop while more and more sensor nodes are affected by a faked query, and consequently the amount of energy used increases. Although the choice of topology 7 and 8 are similar and only distinct in 2 nodes, this has an enormous impact on the power needs. Especially the missing node C in topology 8 reduces the connectivity, thus creating a "bottleneck" and flattening the flooding depth. The energy curve of topology 9 almost equals topology 8 since right at the beginning the query is only spread twice. When looking at the energy per node of topology 10 and 11 it is obvious that they have the least energy need, due to the low interconnectivity between the individual nodes. According to this, their slope is comparatively flat, even up to a propagation probability of 40%. It is obvious that although the additional computation effort is made by using the AQF, the energy is far below the line of the unsecured network when assuming that 100% of the packets have a fake authenticator.

Finding the right mix of parameters that determine probability p_f and the way they relate to the energy is the key for successfully applying the AQF algorithm to a practical example. Due to this Figure 3 illustrates their correlation. Note that all three planes do intersect. Using a key pool with $\ell = 1000$ keys, the plane is relatively flat with an energy maximum at $k = 250, m = 500$ of $0.4970mJ$. The value for p_f is at this point 0%, meaning that fake query are dropped with 100% probability at the first sensor node. With increasing parameter ℓ the plane shifts to a new energy maximum at $m = 100, k = 50$ of 2.0382 mJ due to the fact that the parameter p_f here around 61%. Choosing a key pool of size 10 000 the situation becomes even more extreme, attaining power needs of 3.5078 mJ according to the high propagation of faked queries (p_f=78%).

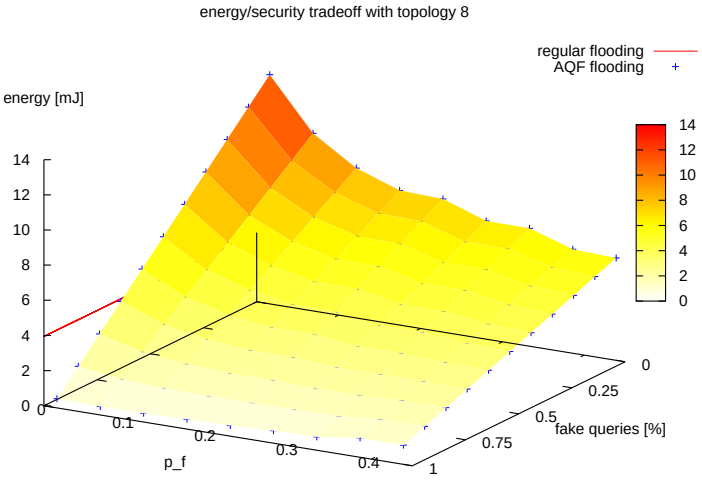
The explanation for this lies in the probability p_f for which we do only indirectly account thru the parameters of m, k, and l. In fact the evident increase in energy as shown in Figure 3 is due to the rapid increase of p_f which grows at many points beyond the admissible range of 50%, e.g. the energy maxima for $\ell = 5000$ and $\ell = 10000$ are $p_f = 0.61$ and $p_f = 0.78$. For this reason it is important to choose parameter ℓ according to the topology since a keypool of 10 000 keys does not achieve the desired effect when having only 15 sensor nodes in a network scenario.

4.1 Energy/Security tradeoff using topology 8

Since the energy wasted is dependent on the severity of the intrusion and the number of faked packets, the figures shown so far do not really account for this. Due to this reason, we now proceed with the problem as motivated earlier in the introduction, namely to find an adequate level of energy that secures the network from outside intrusion. That is in particular the solution of the energy/security tradeoff in relation to the number of packets sent by an adversary. Therefore a



3: The way the input parameters authenticator size m, number of keys per node k, and number of keys in the key pool ℓ correlate with the energy use.



4: Area representing the relation between the number of faked queries in percent, the probability of forwarding a fake packet p_f, and the corresponding energy required.

new variable is included in the model which represents the number of fake queries among the good ones, starting from 0 up to 100 percent. Since such a figure was missing is the section before we are now able to state quantitative predictions about how the energy and security level relate to the severeness of the intrusion.

For fixed parameters ℓ=4000, k=50, and m varying between 100 and 500 bits we compute the probability p_f and the hereby involved power. Denote at this point that there may be more than one parameter configuration that lead exactly to the same forwarding probability, i.e. when doubling ℓ and k the same probabilities p_f are obtained which follows due to obvious reasons. This analysis considers the total amount of energy which is used for a query to be flooded in topology 8, and can also be applied to the other scenarios. The corresponding figure 4 shows the result. The axes labels are: the *percentage of faked queries* on the axis of abscissae, the *security level* as explained by the formula above on the y axis, and the corresponding *power need* in mJ on the z-axis.

Note that 2 different data sources are contained in the illustration. The line on the left side at $p_f = 0$ shows the energy that would be used without the use of any securing

mean. That is packets are received by a node and transmitted again without the computation of hash values (cf. 3.1 variable MNKK), and validation of keys in between. In this case a constant amount of 3.94 mJ is needed for the network to be reached that is completely independent from the probability of forwarding a fake packet p_f.

The second point of interest is the plane showing the relation of faked queries, and p_f to the amount of energy hereby involved. This graph reads as follows: If we assume only sound packets to be send and only little security is in use – that is p_f is high – the AQF algorithm outperforms the regular flooding procedure with respect to energy. By increasing the portion of faked packets, the authenticationed flooding shows effect and the energy need starts to drop. On the other end of the scale ($p_f = 0.016$ and no fake queries) the situation is similar since the securing mean does not show any effect and reached a energy maximum of 13.0392 mJ. As the number of faked queries reaching the network is increasing, more and more packets get filtered out of the network this causing a rapid drop of the energy down to 0.5236 mJ if we assume 100% fake packets to be sent.

5. CONCLUSION

Due to the low-power nature of wireless sensor networks it is hard, choosing the right path between the different constraints such as security, energy, authenticity et cetera. The more it is key that appropriate measures are applied prior to the deployment of a network which shelter against intrusion from outside.
The analysis presented here reveals how the probabilistic model checking tool PRISM can successfully be applied to those challenging problems like the AQF algorithm. We proved, although commonly known problems like state space explosion prevail, there's an enormous wide range of problem instances allowing a deep investigation. The more, what differs them from common simulation tools is the high precision of the obtained results that go without the need of confidence levels.

By the use of reward functions quantitative assertions for a variety of properties can be verified. It turns out that due to the number of nodes in the range of hundreds as proposed in [1], we were not able to validate these results with the formal method approach. The more the here presented analysis should be understood as a complementary approach, that can be used to render simulation input parameters more precisely. As such we proved against our previous anticipation, that the choice of the topology has an tremendous impact on the network security for probabilistic algorithms.

Though our analysis is restricted in the number of nodes for which we give evidence, networks around the size of 15 nodes suffice most of the real-life applications. We further believe that most of the results presented here scale also well for even bigger networks due to symmetry reasons, which needs further proof in future work. So we intent to rerun the presented experiments within an simulation environment to make this work more compareable and strengthen the scalability assumption. Further work could also be spent on the model of the receiving process to include collisions, the initialisation phase, node breakdown failures etc., thus modelling a more realistic network.

6. REFERENCES

[1] Zinaida Benenson, Felix C. Freiling, Ernest Hammerschmidt, Stefan Lucks, and Lexi Pimenidis. Authenticated query flooding in sensor networks. *PerCom Workshops 2006*, 2006.

[2] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM: A tool for automatic verification of probabilistic systems. In H. Hermanns and J. Palsberg, editors, *Proc. 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06)*, volume 3920 of *LNCS*, pages 441–444. Springer, 2006.

[3] B. Kaliski. The md2 message-digest algorithm [rfc1319]. Technical report, RSA Data Security, Inc., April 1992. [RFC1319] Network Working Group Request for Comments.

[4] M. Kwiatkowska, G. Norman, and A. Pacheco. Model checking expected time and expected reward formulae with random time bounds. *Computers & Mathematics with Applications*, 51(2):305–316, 2006.

[5] M. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic symbolic model checker. In P. Kemper, editor, *Proc. Tools Session of Aachen 2001 International Multiconference on Measurement, Modelling and Evaluation of Computer-Communication Systems*, pages 7–12, September 2001. Available as Technical Report 760/2001, University of Dortmund.

[6] The Network Simulator – ns-2. `http://www.isi.edu/nsnam/ns/`.

[7] Datasheet: The TMote Sky Sensor from Moteiv. `http://www.tmotesky.com/products/docs/tmote-sky-datasheet.pdf`.

[8] S. Zhu, S. Xu, S. Setia, and S. Jajodia. Establishing pair-wise keys for secure communication in ad hoc networks: A probabilistic approach, 2003.

Das RTOS Symobi: Erfüllung der Anforderungen in eingebetteten Systemen

Robert Dörfel
Technische Universität München
Fakultät für Informatik
doerfel@in.tum.de

Michael Haunreiter
Miray Software AG
m.haunreiter@miray.de

Uwe Baumgarten
Technische Universität München
Fakultät für Informatik
baumgaru@in.tum.de

ABSTRACT

In eingebetteten Systemen gibt es viele Anforderungen, die das Betriebssystem erfüllen muss. Diese reichen von stark begrenzten Hardwareressourcen, Zuverlässigkeit, Echtzeitfähigkeit bis hin zur Portabilität des Betriebssystems. Die Herausforderung bei der Entwicklung eines solchen Betriebssystems besteht vor allem darin, diese sich teilweise widersprechenden Eigenschaften in einem Gesamtsystem optimal zu vereinen. Das embeddable RTOS Symobi bietet mit seiner Architektur einen modernen Ansatz, mit dem die genannten Anforderungen vereint und gleichzeitig jede einzelne weitgehend erfüllt werden kann.

1. EINLEITUNG

Betriebssysteme für eingebettete Systeme stehen vor dem Konflikt, dass sie einerseits maximale Leistung mit den meist ohnehin knapp bemessenen Hardwareressourcen erzielen müssen, andererseits aber auch möglichst vielseitig einsetzbar sein sollen. Sie sollten daher bei der Erfüllung der an sie gestellten Anforderungen eine möglichst gute Balance zwischen Abstraktion und Spezialisierung bieten. Zu den wichtigsten Anforderungen an ein Betriebssystem für eingebettete Systeme gehört die *Zuverlässigkeit*, weil sie in Umgebungen eingesetzt werden, in denen ein Ausfall der Hardware oder ein Absturz des Betriebssystems in der Regel zu größeren Schäden führen kann als bei einem Desktopsystem. Beispielsweise führt ein Ausfall einer Fertigungssteuerung in der Fabrik zu hohen wirtschaftlichen Schäden. Eingebettete Systeme werden häufig in der Steuerung von technischen Prozessen verwendet. Dort spielt auch die *Echtzeitfähigkeit* eine große Rolle, da das System rechtzeitig auf Sensorwerte reagieren muss, um entsprechend die Aktuatoren zu steuern. Durch die große Anzahl der verschiedenen Hardware-Plattformen sollte das Betriebssystem in eingebetteten Systemen sehr anpassungsfähig sein. So ist es sehr wertvoll, wenn sich das Betriebssystem durch eine schnelle und leichte *Portierbarkeit* auszeichnet, damit es auf den unterschiedlichen in diesem Bereich eingesetzten Prozessoren läuft [3]. Nicht nur die Prozessor-Plattform unterscheidet sich in eingebetteten Systemen sondern auch die Größe des Arbeitsspeichers und die Anzahl und Art der angeschlossenen Geräte. Somit kann ein Betriebssystem mit guter *Skalierbarkeit* leicht an die jeweilige Hardware angepasst werden. Dabei sollte sich das Betriebssystem nicht nur zur Compile- und Installationszeit gut skalieren lassen, sondern auch dynamisch zur Laufzeit. Durch die Ressourcenknappheit, die in eingebetteten Systemen vorherrscht, muss das Betriebssystem in seiner Minimalkonfiguration sehr schlank sein. Trotzdem wird erwartet, dass das Betriebssystem je nach Anforderung und Ausstattung des eingebetteten Systems wachsen kann. Eingebettete Systeme sind zunehmend auch kommunikationsorientiert. Dies fordert vom Betriebssystem möglichst transparente *Kommunikationsmechanismen* für Anwendungen und Dienste. Neben der geforderten Zuverlässigkeit steht in eingebetteten Systemen die *Effizienz* des Betriebssystems im Vordergrund. Da in eingebetteten Systemen die Rechenleistung der Prozessoren häufig niedriger als bei Desktop- oder Serversystemen ist, muss das Betriebssystem möglichst effizient arbeiten, um die für sich selbst benötigte Prozessorlast und den Speicherplatz gering zu halten. Viele der eingebetteten Systeme werden mit Batterie in platzsparenden Gehäusen betrieben. Durch *Energiemanagement* seitens des Betriebssystems kann die Leistung des eingebetteten Systems an momentane Anforderungen angepasst werden. So kann Energie gespart und Wärmeentwicklung vermieden werden, was zu einer längeren Akkulaufzeit führt und unerwünschte oder gar schädliche Wärmeentwicklung im Gehäuse verhindert.

Das RTOS Symobi[1] berücksichtigt bereits in seinem Architekturansatz die genannten Anforderungen. Ziel von Symobi ist es, diese in einem Betriebssystem zu vereinen und gleichzeitig jede einzelne so weitgehend zu erfüllen, dass das Ergebnis mit dem einseitig spezialisierter Betriebssysteme vergleichbar ist. Im nächsten Abschnitt wird zunächst die Architektur von Symobi beschrieben und auf die Besonderheiten des Systems eingegangen. Anschließend wird in Abschnitt 3 vorgestellt, wie Symobi die genannten Anforderungen an ein Betriebssystem für eingebettete Systeme erfüllt.

GI/ITG KuVS Fachgespräch: Systemsoftware und Energiebewusste Systeme, 11. Oktober 2007, Universität Karlsruhe

[1]Das embeddable RTOS Symobi wurde von der Firma Miray Software von Grund auf neu entwickelt. Seit 2002 besteht dabei eine enge Zusammenarbeit mit der Arbeitsgruppe MVS unter Leitung von Prof. Dr. Baumgarten am Lehrstuhl I13 der TU München. Die Zusammenarbeit konzentriert sich besonders auf den Einsatz von Symobi auf mobilen embedded Systemen und hat unter anderem die Unterstützung zum Einsatz von Symobi auf diversen ARM-basierten embedded Systemen (PXA25x, PXA270, IXP425) geführt.

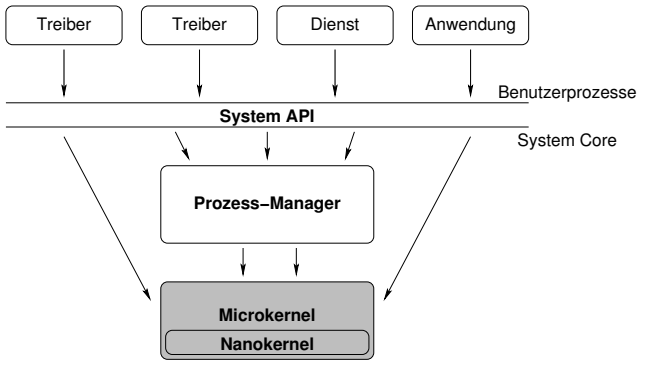


Abbildung 1: Die Microkernel-Architektur von Symobi. Die Pfeile stellen die Funktionsaufrufrichtung dar.

Anhand eines Beispiels zur Benutzerinteraktion in Symobi wird die modulare und skalierbare Architektur von Symobi verdeutlicht. In Abschnitt 3.7 wird beschrieben, wie neben dem im Microkernel vorhandenen allgemeinen auch ein hardwarespezifisches, plattformabhängiges Energiemanagement in Symobi integriert werden kann. Abschließend wird zusammengefasst, wie Symobi die Anforderungen in eingebetteten Systemen mit seinen Konzepten löst.

2. ARCHITEKTUR

Symobi baut auf einer Microkernel-Architektur auf, die in Abbildung 1 dargestellt ist. Basis des Betriebssystems ist der abgeschlossene Microkernel, auf dem die restlichen Komponenten des Betriebssystems aufbauen. Der Prozess-Manager, der zusammen mit dem Microkernel den so genannten *System Core* bildet, sorgt für die Verwaltung der Prozesse. Treiber und Betriebssystemdienste - Aufrufe des System Core werden nicht als Dienste, sondern als Systemaufrufe bezeichnet - sind nicht im Microkernel enthalten und laufen wie Anwendungen als eigenständige Prozesse im nicht-privilegierten Prozessormodus. Über das in C++ implementierte, objektorientierte System API erhalten die Benutzerprozesse Zugriff auf den System Core. Nur der in Abbildung 1 grau hinterlegte Microkernel läuft im privilegierten Prozessormodus. Selbst der Prozess-Manager, der Teil des System Cores ist, wird im nicht-privilegierten Modus ausgeführt.

Da die Treiber, Dienste und Anwendungen in eigenen Prozessen laufen, die getrennte, zugriffsgeschützte Adressräume haben, steht für den Nachrichtenaustausch ein IPC-Mechanismus zur Verfügung, auf den in Abschnitt 2.4 eingegangen wird.

2.1 System Core

Die Basis von Symobi bildet der System Core, der aufgeteilt ist in den privilegierten Microkernel und den nichtprivilegierten Prozess-Manager. Über diese wird eine Trennung zwischen Mechanismus und Strategie realisiert. Zu den Aufgaben des System Core gehören Scheduling, Synchronistionsoperationen, I/O-Management, Speichermanagement und die IPC. Während der Microkernel dem Prozess-Manager und den anderen Benutzerprozessen primitive Operationen zur Verfügung stellt, enthält der Prozess-Manager die nötigen Strategien für komplexere Betriebssystemoperationen und den IPC Mechanismus. Treiber, Dienste und Anwendungen sind in normalen Benutzerprozessen organisiert. Über das System API können diese auf die Funktionen des System Core zugreifen.

Zum Funktionsumfang des Microkernels gehören Scheduling, Thread-Synchronisation, Reservierung und Freigabe von Systemressourcen und Interrupts.

Der Microkernel enthält einen so genannten Nanokernel, der die prozessorspezifischen Low-Level-Operationen implementiert und so dem übrigen Microkernel eine Abstraktion der prozessornahen Hardware zur Verfügung stellt. Die Schnittstelle zwischen dem Nanokernel und dem Rest des Microkernels ist für alle Prozessorplattformen identisch. Bestimmte Prozessoreigenschaften können falls notwendig vom Nanokernel durch Softwareimplementierungen ersetzt werden, damit unter allen Plattformen die einheitliche Schnittstelle vorhanden ist. Nur im Nanokernel sind Codeanteile in Assembler enthalten. Der übrige Microkernel nutzt diese Schnittstelle und ist selbst ausschließlich in C geschrieben.

Der Prozess-Manager enthält hauptsächlich die Strategien für Ressourcenverwaltung und IPC. Bei der Vergabe von Systemressourcen entscheidet der Prozess-Manager wie die Ressourcen verteilt werden und reserviert diese entsprechend über die Primitiven des Microkernels. Dieser Mechanismus wird anhand des Speichermanagements veranschaulicht. Der Microkernel verwaltet den Speicher, der über seine Funktionen reserviert und freigegeben werden kann. Wenn ein Benutzerprozess Speicher reserviert, entscheidet der Prozess-Manager, welcher Speicherbereich verwendet wird und reserviert diesen über die Funktionen des Microkernels.

Der Prozess-Manager enthält überwiegend C++ Programmcode, wobei aus Gründen der Performance auf C++-Sprachmerkmale verzichtet wird, die sich negativ auf das Laufzeitverhalten auswirken können (z. B. Exceptions).

2.2 Treiber und Dienste

Der System Core enthält keine Gerätetreiber oder Betriebssystemdienste, wie z. B. einen Netzwerk- oder Filesystemdienst. Diese sind alle außerhalb des System Core in eigenen Benutzerprozessen realisiert. Zugriff auf die Hardware erhalten die Treiber über das System API.

Bevor der Treiber auf Systemressourcen (z. B. Interrupt, I/O-Port) zugreifen kann, muss er diese über das System API reservieren. Bei einer Reservierung entscheidet der Prozess-Manager, ob der Treiber Zugriff auf die entsprechende Ressource erhält. Falls die Ressource bereits belegt bzw. nicht als *shared* reserviert ist oder der Treiber nicht die dafür nötigen Rechte besitzt, wird der Zugriff vom Prozess-Manager abgelehnt. Sobald dem Treiber die Reservierung genehmigt wurde, kann er ebenfalls über Funktionen des System API darauf zugreifen (z. B. auf Interrupt warten, I/O-Ports lesen/schreiben).

Treiber können ihre Funktionalität auf drei Arten anderen Prozessen zur Verfügung stellen. Üblicherweise ist der Treiber als Bibliothek in einen Dienstprozess eingebunden, auf

den andere Prozesse per IPC zugreifen. Eine weitere Möglichkeit ist die direkte Integration in einen Anwendungsprozess, der diesen Treiber exklusiv verwendet (z. B. Sicherheitsarchitektur, Spezialtreiber). Für kritische Systemumgebungen ist auch die Implementierung eines Treibers als einzelner Prozess vorgesehen, der in diesem Fall seine Funktionen direkt per IPC zur Verfügung stellt. Dienste, die einen solchen Treiber verwenden, bleiben von dessen möglichen Fehlfunktionen unbeeinträchtigt.

2.3 Anwendungen

Anwendungen sind in Symobi Prozesse, die nicht als Dienste fungieren sondern in der Regel als Clients andere Dienste nutzen. Für die Dienste, die Anwendungen zur Verfügung stehen, gibt es unter Symobi verschiedene Möglichkeiten. Durch seinen klaren modularen Aufbau ist Symobi frei erweiterbar. Treiber und Dienste können unabhängig erstellt, verbessert, erweitert und ersetzt werden. Für diese freie Gestaltbarkeit des Betriebssystems wird der Begriff *Open-System-Architecture* eingeführt. Damit können neben den bestehenden auch eigene Treiber und Dienste implementiert werden. Diese sind frei gestaltbar, so dass auch komplette Subsysteme hinzugefügt werden können. Ein Beipiel hierfür ist die im Rahmen einer Diplomarbeit umgesetzte Java-VM für Symobi [2].

Die Anwendungen werden selbst in eingebetteten Systemen immer umfangreicher und komplexer. Um die objektorientierte Entwicklung unter Symobi zu erleichtern, bietet das System API eine auf Reference Counting basierende Garbage Collection an, die echtzeitfähig und weitgehend transparent ist, d. h. einzig bei der Deklaration von Variablen ist eine abweichende Syntax erforderlich.

2.4 Interprozesskommunikation

Da die Prozesse in Symobi getrennte, geschützte Adressräume besitzen, werden Daten zwischen Prozessen per IPC ausgetauscht. Durch die Trennung der Treiber und Dienste in eigene Prozesse ist eine effiziente IPC nötig. Als grundlegender IPC Mechanismus wird das Client/Server-Modell verwendet. Prozesse, die einen oder mehrere Dienste anbieten, werden als Server während Prozesse, die Dienste nutzen, als Clients bezeichnet werden. In einem Server öffnet jeder Dienst mit Hilfe des System API mindestens einen Kanal (Channel). Dieser ist entsprechend dem Client/Server-Modell synchron und bidirektional. Jeder Kanal besitzt eine systemweit eindeutige ID, über die auch Threads aus anderen Prozessen eine Verbindung (Connection) zu diesem Kanal aufbauen können. Der Dienst kann jedoch entscheiden, ob er eine Verbindung mit einem Client eingehen will oder nicht. Die Entscheidung darüber kann dynamisch und situationsabhängig zur Laufzeit erfolgen. Sobald ein Client eine Verbindung zum Server erhalten hat, kann er über Nachrichten Anfragen an den jeweiligen Dienst senden. Während der Dienst die Anfrage bearbeitet, blockiert der Sende-Thread im Client. Sobald der Dienst das Ergebnis als Nachricht über die selbe Verbindung zurückgesendet hat, wird der Sende-Thread wieder geweckt und kann das Ergebnis verarbeiten.

Der IPC-Mechanismus in Symobi lässt es zu, dass ein Prozess gleichzeitig Client und Server ist. Dabei bietet der Prozess über einen oder mehrere Kanäle einen oder mehrere Dienste an und enthält zusätzlich Threads, die als Clients

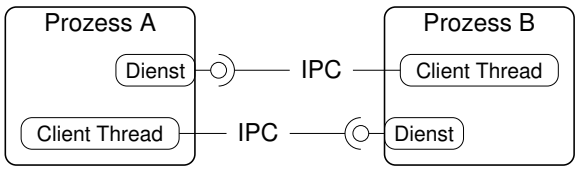


Abbildung 2: Die IPC unter Symobi. Die Kreise an den Prozessen sind Kanäle, an die sich Client Threads per IPC anbinden können.

Verbindungen zu anderen Servern haben. Es ist sogar möglich, dass zwei Prozesse jeweils einen Dienst anbieten, zu dem sie sich gegenseitig verbinden. Diese Konfiguration ist in Abbildung 2 dargestellt, wobei die Kreise die Kanäle der Prozesse sind, an die sich Clients anbinden können. Dabei können mehrere Threads, auch aus unterschiedlichen Prozessen, gleichzeitig mit einem Kanal verbunden sein.

Zusätzlich bietet Symobi *Managed Direct Memory* (MDM) an, das unter anderem auch zusammen mit der IPC für den Datenaustausch zwischen Prozessen genutzt werden kann. Das MDM folgt dem Prinzip eines Zero-Copy-Mechanismus. Allerdings bietet das MDM von Symobi im Gegensatz zu anderen Implementierungen zusätzlich einen umfassenden Zugriffsschutz. Per MDM können insbesondere große Datenmengen zwischen Prozessen noch effizienter ausgetauscht werden als über IPC-Nachrichten. Der IPC-Mechanismus wird trotzdem für die Synchronisation benötigt, so dass MDM kein alternativer Kommunikationsmechanismus ist, sondern nur eine Alternative für den reinen Datenaustausch zwischen Prozessen.

3. ANFORDERUNGEN

Nachdem bis hierher die Architektur von Symobi vorgestellt wurde, erläutert dieses Kapitel, wie Symobi die Anforderungen an ein Betriebssystem für eingebettete Systeme erfüllt. Dabei sind viele der Ziele konträr. So verliert beispielsweise ein zuverlässiger Microkernel an Effizienz [4]. Die Architektur von Symobi ist darauf ausgerichtet, eine optimierte Verbindung aller dieser Ziele zu erreichen.

Durch die im System Core umgesetzten Strategien und spezielle Mechanismen maximiert Symobi die Leistung einzelner Systemeigenschaften ohne andere, teilweise konträre Eigenschaften zu stark zu beeinträchtigen. Dabei steht die Zuverlässigkeit an erster Stelle, noch vor der Effizienz.

Im nachfolgendem werden die Ziele Zuverlässigkeit, Echtzeitfähigkeit, Portierbarkeit und Skalierbarkeit betrachtet. Wie ein transparenter Kommunikationsmechanismus und Benutzerinteraktion in Symobi realisiert werden, ist in den Kapiteln 3.5 und 3.6 beschrieben. Abschließend wird die Möglichkeit diskutiert, wie ein Energiemanagement in Symobi implementiert werden kann.

3.1 Zuverlässigkeit

Einer der wichtigsten Anforderungen an eingebettete Systeme ist deren Zuverlässigkeit und Robustheit, da sie in Umgebungen eingesetzt werden, in denen ein Fehlverhalten

zu körperlichem, materiellem oder wirtschaftlichem Schaden führt [1]. Dafür ist ein zuverlässiges Betriebssystem notwendig. In Symobi ist bereits durch den Einsatz einer Microkernel-Architektur eine höhere Zuverlässigkeit gewährleistet [4]. Die Architektur des Microkernels in Symobi enthält anerkannte Prinzipien, die die Zuverlässigkeit eines Microkernels erhöhen [5]. Durch die Auslagerung des Prozess-Managers aus dem Microkernel ist das Prinzip der Trennung des Mechanismus von der Strategie implementiert. Ebenfalls erhalten die Komponenten in Symobi die jeweils niedrigste Privilegstufe, die sie für die Erfüllung ihrer Aufgaben benötigen, was als ein weiteres Prinzip bekannt ist [5]. Dies ist in Symobi dadurch umgesetzt, dass alle Treiber und Dienste in eigenen Benutzerprozessen laufen und nur der Microkernel im privilegierten Modus des Prozessors arbeitet.

Weil keine Treiber und Dienste im Microkernel enthalten sind, führen fehlerhafte Gerätetreiber nicht zum Absturz des gesamten Systems, sondern nur zum Absturz des jeweiligen Prozesses. Falls ein Treiber abgestürzt ist, kann der jeweilige Prozess aus dem System entfernt und der Treiber in einem neuen Prozess wieder gestartet werden. Da in einem monolithischen Betriebssystem die häufigste Ursache für einen Systemabsturz fehlerhafte Treiber sind [6], wird dadurch in Symobi eine wesentliche Fehlerquelle ausgeschaltet.

In Symobi resultiert die grundlegende Zuverlässigkeit des Betriebssystems auch aus einer geringen absoluten Größe des Microkernels. Mit nur ca. 10-15 Tausend Codezeilen, die im privilegierten Modus des Prozessors ausgeführt werden enthält der Microkernel schon rein statistisch weniger mögliche Fehler als größere Kernel [7]. Außerdem ermöglichen der geringe Codeumfang und die Tatsache, dass der Code im Microkernel unveränderlich ist, prinzipiell auch eine manuelle Codeinspektion.

3.2 Echtzeitfähigkeit

Da eingebettete Systeme häufig zur Steuerung technischer Prozesse eingesetzt werden, muss das dafür verwendete Betriebssystem echtzeitfähig sein. Der Microkernel in Symobi ist ein hart-echtzeitfähiger, reaktiver Microkernel ohne eigene Threads, wofür unter anderem folgende Eigenschaften relevant sind.

Das auf Prioritäten basierende Echtzeit-Scheduling des Microkernel sorgt dafür, dass keine echtzeitkritischen Threads durch andere Threads unterbrochen werden. Dafür werden die 32 Prioritäten in zwei Gruppen eingeteilt. Die niedrigen Prioritäten sind für nicht echtzeitkritische Aufgaben vorgesehen, die nach dem Round Robin Verfahren unter Einbeziehung ihrer Priorität gescheduled werden. Dabei werden sie nach Ablauf ihrer Zeitscheibe unterbrochen, so dass auch Threads mit niedrigerer Priorität aktiv werden können. Die zweite Gruppe mit den höheren Prioritäten sind für echtzeitkritische Threads reserviert. Threads mit diesen Prioritäten können nur von Threads mit einer höheren Priorität unterbrochen werden. Dabei ist der Programmierer selbst dafür zuständig, die Prioritäten in den von ihm erstellten Treibern, Diensten und Anwendungen so zu vergeben, dass die Deadlines zeitkritischer Threads eingehalten werden. Wenn bespielsweise zwei Threads die höchste Priorität erhalten, kann dennoch immer nur einer davon tatsächlich in Echtzeit agieren.

Die Garbage Collection, die das System API anbietet, ist ebenfalls echtzeitfähig. Durch den Mechanismus des Reference Countings wird kein zusätzlicher, asynchron ablaufender Garbage Collection Thread benötigt, dessen Einfluss auf das Echtzeitverhalten nicht vorhersagbar wäre. Die Operationen des Reference-Counting-Mechanismus werden augenblicklich, inline und synchron im jeweiligen Thread ausgeführt, so dass das Echtzeitverhalten unverändert bleibt.

3.3 Portierbarkeit

In Bereich eingebetteter Systeme finden viele verschiedene Prozessorplattformen Verwendung. Ein Betriebssystem für eingebettete Systeme sollte deshalb sehr leicht portierbar sein, damit es auf verschiedenen Plattformen lauffähig ist. Zur Zeit läuft Symobi auf der x86, ARM/XScale und PowerPC Plattform.

Die einfache Protierbarkeit des Microkernels wird durch den enthaltenen Nanokernel erreicht. Da nur der Nanokernel, der die Abstraktion der prozessornahen Hardware bildet, durch seine enthaltenen Assembler-Routinen bei einer Portierung des System Cores angepasst werden muss, ist der System Core leicht portierbar. Der restliche Microkernel und der Prozess-Manager sind in C bzw. C++ geschrieben und bauen auf den Nanokernel bzw. dem Microkernel auf. Damit ist bei einem für die Zielplattform vorhandenen C/C++-Compiler der Portierungsaufwand für den restlichen Microkernel und den Prozess-Manager minimal.

Allerdings muss der Nanokernel auf der neuen Plattform die Schnittstelle zwischen Nanokernel und Microkernel umsetzen. Dennoch ist der Gesamtaufwand für eine Portierung wesentlich niedriger als bei einer Portierung des gesamten Microkernels.

Die Portabilität der Treiber in den Benutzerprozessen ist je nach Programmierung des Treibers unterschiedlich. Wenn sie keinen Assembler Code sondern nur reinen C/C++-Code enthalten, bringt die Portierung auf die neue Hardware keine Schwierigkeiten mit sich, da der Treiber mit einem vorhandenen C/C++-Compiler für die neue Plattform ohne Codeänderungen erzeugt werden kann. Allerdings müssen möglicherweise andere Systemressourcen wie Interrupts oder I/O-Ports reserviert werden. Dafür sind geringfügige Anpassungen im Quelltext notwendig.

Für Dienste, die nur auf anderen Treibern und dem System API aufbauen, sind keine weiteren Anpassungen an eine neue Plattform notwendig. Sie müssen nur mit einem entsprechenden Compiler für die neue Plattform übersetzt werden.

3.4 Skalierbarkeit

Da eingebettete Systeme im Allgemeinen sehr unterschiedliche Größen haben können, sollte ein dafür konzipiertes Betriebssystem sehr gut skalierbar sein, um die entsprechenden Anforderungen zu erfüllen. Die Systeme variieren in der Rechenleistung des Prozessors, in der Größe des Arbeits- und des persistenen Speichers und in der Anzahl und Art der angeschlossenen Geräte.

Symobi ist durch seinen Microkernel, der nur ca. 45 KB umfasst, und dem Prozess-Manager, der eine Größe von

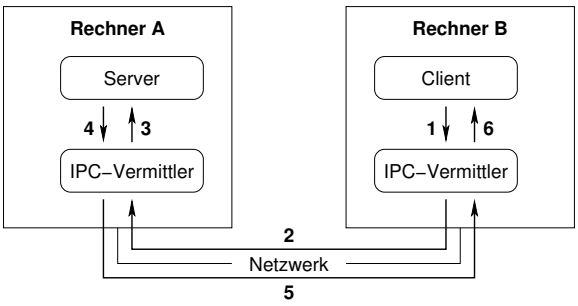


Abbildung 3: Für die Kommunikation über Netzwerk ist der IPC-Vermittler notwendig, der die IPC Nachrichten über das zugrunde liegende Netz schickt.

ca. 55 KB hat, mit einer Gesamtgröße von ca. 100 KB auf der x86 Plattform sehr schlank. So läuft der System Core selbst bereits auf kleinsten eingebetteten Systemen. Da die Treiber und Dienste außerhalb des System Core laufen, können diese je nach Anforderung des Systems hinzugefügt werden. Der gesamte Speicherbedarf von Symobi richtet sich dann nach Größe und Anzahl der gestarteten Treiber, Dienste und Anwendungen.

Darüber hinaus ist Symobi auch zur Laufzeit sehr gut skalierbar. So können Treiber oder Dienste zu jedem Zeitpunkt gestartet oder beendet werden. In kleinen Systemen können somit die knappen Ressourcen je nach Anfrage und Auslastung unterschiedlich reserviert werden. So kann ein Dienst zusammen mit einem möglicherweise notwendigen Treiber erst gestartet werden, wenn dessen Funktionalität gebraucht wird. Wenn dieser nicht mehr benötigt wird, kann er wieder mit dem Treiber beendet werden, um die reservierten Systemressourcen freizugeben.

3.5 Externe Kommunikation

In eingebetteten Systemen spielt die Kommunikation nach außen eine zunehmende Rolle. So findet vor allem in mobilen Endgeräten eine Kommunikation untereinander statt. Für den Informationsaustausch sind diese Endgeräte häufig auch mit einer größeren Infrastruktur oder Basisstation verbunden.

Der IPC-Mechanismus unter Symobi erlaubt prinzipiell auch eine Kommunikation über Rechnergrenzen hinweg. Da sich ein Client mit Hilfe einer ID zu einem Kanal verbindet, kann sich der Server nicht nur auf dem lokalen Knoten, sondern auch auf einem entfernten Knoten im Netzwerk befinden. Dies ist für den Client transparent, da er das System API in beiden Fällen auf die gleiche Weise verwendet. Aus Sicht des Servers erstellen Clients Verbindungen zu dessen Kanälen, über die die Clients ihre Nachrichten schicken. Ob die Verbindung lokal oder über das Netzwerk aufgebaut wird, bleibt dem Server dabei verborgen. Damit können Client und Server mit dem IPC-Mechanismus unter Symobi über ein Netzwerk kommunizieren, ohne dass eine Programmänderung im Client oder Server notwendig ist.

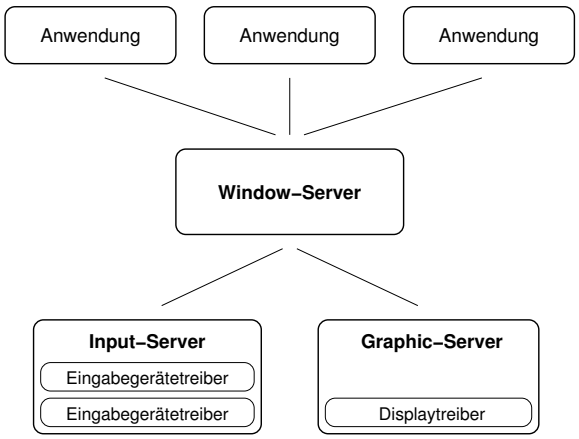


Abbildung 4: Der Input- und Graphic-Server stellt die grundlegenden Operationen für die Benutzerein- und Ausgabe in der Standardkonfiguration bereit. Auf diesen baut der Window Server auf, der den Anwendungen ein graphisches Fenstersystem liefert.

Allerdings muss dafür ein zusätzlicher Dienst, der sogenannte IPC-Vermittler, in das System eingebracht werden, der für das Auffinden der Dienste im Netzwerk und das Versenden der Nachrichten über das Netzwerk zuständig ist [4]. Dieser Zusammenhang ist in Abbildung 3 illustriert. Der IPC-Vermittler ist wie jeder andere Dienst ein Benutzerprozess. Er bietet lokal im Rechner als Server alle Dienste an, die er auf anderen Knoten im Netzwerk findet. Zusätzlich stellt er einen Namensdienst bereit, über den die Clients Kanal-IDs von Diensten im Netzwerk erfragen können. Anschließend verbinden diese sich lokal über IPC mit dem IPC-Vermittler, der die Anfrage über das zugrunde liegende Netz an den IPC-Vermittler des entfernten Knotens schickt. Dieser wandelt die Anfrage wieder in eine lokale IPC-Nachricht um, und übergibt diese dem Server. Die Antwort des Servers wird über den selben Weg zurück an den Client gesandt.

In welchem Netzwerk, die beiden Rechner verbunden sind, ist für den Client- und Serverprozess ebenfalls transparent. Nur der IPC-Vermittler muss für die oben beschriebene Kommunikation wissen, welches Netz verfügbar ist. Wenn mehrere Pfade in unterschiedlichen Netzen für die Verbindung von Client und Server zur Verfügung stehen, ist es auch möglich, das jeweils am besten geeignete zu verwenden. So ist es beispielsweise in mobilen System mit drahtloser Kommunikation häufig der Fall, dass verschiedene Kommunikationsarten mit unterschiedlichen Bandbreiten vorhanden sind. So ist es denkbar, dass in einem mobilen Endgerät die Kommunikation über WLAN, Bluetooth, GSM oder UMTS möglich ist. Auch bei einem Verbindungsabbruch in einem Netz kann der IPC-Vermittler die Kommunikation in einem anderen Netz, in dem sich Server- und Client-Knoten befinden, fortführen.

3.6 Benutzerinteraktion

In eingebetteten Systemen findet meist Interaktion mit dem Benutzer statt, wofür es unterschiedliche Ein- und Ausgabe-

geräte gibt. Um den Anwendungen eine einheitliche Schnittstelle für die Benutzerein- und Ausgabe trotz der Vielfalt der Geräte zu bieten, nutzt Symobi das Konzept der Client/Server-Architektur. So stellt Symobi einen Input-Server für die Benutzereingaben und einen Graphic-Server für eine graphische Oberfläche bereit. Die beiden Server greifen über die entsprechenden Treiber, die als Bibliotheksfunktionen in die Server eingebunden werden, auf die Ein- bzw. Ausgabegeräte zu. Auf den Input- und den Graphic-Server baut der Window-Server auf, der den Anwendungen ein grafisches Fenstersystem in einer einheitlichen Schnittstelle mit Maus- und Tastatureingaben anbietet. Die Abbildung 4 stellt die Realisierung der Benutzerinteraktion in Symobi schematisch dar.

Um die vielen unterschiedlichen Ein- und Ausgabegeräte, die es für eingebettete Systeme gibt, zu unterstützen, bietet die Open-System-Architecture von Symobi eine gute Grundlage. Es müssen lediglich die Treiber für die Ein- und Ausgabegeräte angepasst werden, wobei der Window-Server bei einer Änderung der Ein- oder Ausgabegeräte nicht verändert werden muss, da er seine Informationen aus dem Input- und Graphic-Server bezieht.

3.7 Energiemanagement

Eingebettete Systeme müssen in Umgebungen arbeiten, in denen nicht nur wenig Speicher und wenig Prozessorleistung vorhanden ist, sondern auch Energie nur in begrenzten Maße zur Verfügung steht. Deshalb müssen eingebettete Systeme sehr energiebewusst arbeiten, was von Betriebssystemseite koordiniert werden sollte. In diesem Bereich bietet Symobi durch seine modulare und skalierbare Architektur die Möglichkeit, Komponenten für das Energiemanagement problemlos hinzuzufügen. Das System API hat Funktionen, um die Prozessorauslastung abzufragen. Mit diesen Informationen können Gerätetreiber die Hardware entsprechend steuern. So ist es nicht nur möglich die Hardware für das Energiemanagement anzusteuern, sondern jeder Treiber könnte z. B. die Leistungsaufnahme des von ihm angesteuerten Geräts nach den Informationen über die Auslastung des Systems autonom regeln.

Der Microkernel selbst arbeitet ebenfalls energiebewusst. So hält er den Prozessor mit einem entsprechenden Befehl an, wenn alle Prozesse im System auf Ereignisse warten. Durch das zeitgesteuerte Schedulingverfahren wird der Prozessor mit einem Interrupt wieder geweckt, sobald ein Prozess rechenbereit wird.

4. ZUSAMMENFASSUNG

In eingebetteten Systemen sind unterschiedliche Anforderungen an ein Betriebssystem vorhanden, die zum Teil konträr sind. Das embeddable RTOS Symobi vereinigt und erfüllt mit seinem Architektur-Konzept die Anforderungen hinsichtlich Zuverlässigkeit, Robustheit, Echtzeitfähigkeit, Portabilität und Skalierbarkeit. Auch zunehmend bedeutende Eigenschaften wie transparente Kommunikationsmechanismen und Energiemanagement werden von Symobi berücksichtigt. So wird mit dem Microkernel, der keine Treiber und Dienste enthält, eine zuverlässige Basis geschaffen. Zusammen mit dem Prozess-Manager entsteht daraus ein hochgradig skalierbares RTOS, dessen schlanker System Core den knappen Speicherressourcen in eingebetteten Systemen Rechnung trägt. Durch das Hinzufügen weiterer Treiber und Dienste kann Symobi für spezielle Anwendungen bzw. bestimmte Hardware-Plattformen angepasst werden. Seine Echtzeitfähigkeit erreicht Symobi durch einen reaktiven Microkernel mit einem auf Prioritäten basierende Echtzeitscheduling. Die IPC in Symobi sorgt für eine transparente Kommunikation, bei der Client und Server davon unabhängig sind, ob sich der Kommunikationspartner lokal oder auf einem entfernten Knoten im Netzwerk befindet. Im Bereich des Energiemanagements, dessen Rolle in eingebetteten Systemen in Zukunft noch weiter zunehmen wird, bietet Symobi ebenfalls mit seiner modularen Architektur eine solide Grundlage, die jetzt noch unbekannten Energiefunktionen der zukünftigen Hardware flexible und ohne die Anpassung des Microkernels in das Gesamtsystem zu integrieren.

5. LITERATUR

[1] Calvez, J. P.: *Embedded real-time systems*. John Wiley & Sons, Inc., New York, NY, USA, 1993. Translator-Alan Wyche and Translator-Charles Edmundson.

[2] Dörfel, R.: *Konzeption und Implementierung einer Java VM für μnOS*. Diplomarbeit, Technische Universität München, 2006.

[3] Kopetz, H.: *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[4] Liedtke, J.: *On micro-kernel construction*. SIGOPS Oper. Syst. Rev., 29(5):237–250, 1995.

[5] Shapiro, J. und N. Hardy: *EROS: a principle-driven operating system from the ground up*. Software, IEEE, 19(1):26–33, 2002.

[6] Swift, M. M., B. N. Bershad und H. M. Levy: *Improving the reliability of commodity operating systems*. In: *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, S. 207–222, New York, NY, USA, 2003. ACM Press.

[7] Tanenbaum, A. S., J. N. Herder und H. Bos: *Can We Make Operating Systems Reliable and Secure?*. Computer, 39(5):44 – 51, Mai 2006.

Betriebssystemunterstützung für verteilte Anwendungen in realer Raumzeit

[Extended Abstract]

Matthias Werner
TU Chemnitz
Betriebssysteme
mwerner@informatik.tu-chemnitz.de

Gero Mühl, Helge Parzyjegla
Hans-Ulrich Heiß,
TU Berlin
Kommunikations- und Betriebssysteme
{gmuehl, parzy, heiss}@cs.tu-berlin.de

ZUSAMMENFASSUNG

Mobile Systeme agieren häufig zusammen, um vorgegebene Aufgaben zu lösen. Hierbei sind besonders diejenigen Anwendungen interessant, die von der Kooperation mehrerer Akteure besonders profitieren oder die sich durch Kooperation erst realisieren lassen. Dies trifft vor allem auf Anwendungen zu, deren Ausführung eine gemeinsame Koordinierung in realer Raumzeit (also gleichzeitig in Ort und Zeit) benötigen. Beispiele für adressierte Systeme sind Rechner in Formationen von beweglichen Trägersystemen (z.B. Satelliten, Kleinstflugzeuge, bewegliche Sensor-Netzwerke) sowie Schwärme (z.B. Roboter-Schwärme). Eine exemplarische Anwendung ist die koordinierte Beobachtung eines Naturphänomens (wie z.B. eines Vulkansausbruchs) durch eine Menge von Kleinstsatelliten.

Im Rahmen dieses Papiers wird erläutert, wieso eine neue Art von Betriebssystem für Anwendungen in realer Raumzeit sinnvoll ist und, welche Anforderungen an ein solches Betriebssystem gestellt werden. Anschließend stellen wir unsere Vision eines derartigen Betriebssystems vor, in der in Analogie zu klassischen verteilten Systemen eine Menge von mobilen Systemen als ein einzelnes Gesamtsystem aufgefasst wird. Es werden Konzepte für geeignete Programmier- und Ausführungsmodelle diskutiert, die letztendlich in das an unserem Fachgebiet in Entwicklung befindlichen Betriebsystem Flock-OS einfließen sollen.

1. EINFÜHRUNG

Die Forschung und Entwicklung in der Informationstechnik ist gegenwärtig stark durch den Trend geprägt, dass die absolute Anzahl sowie auch der relative Anteil von mobilen Systemen immer mehr zunehmen. Mobile Systeme agieren jedoch selten völlig autonom, sondern sie interagieren und kooperieren mit anderen Systemen. Dabei wird häufig angestrebt, dass mehrere Systeme Aufgaben *gemeinsam* lösen. Aus dem Blickwinkel der Mobilität kann dies auf unterschiedliche Weisen geschehen: Mobile Systeme können sich bei einer Aufgabe, die an bestimmte Orte gebunden ist, ablösen; Sie können ihre relative oder absolute Position zueinander Nutzen, um geometrieabhängige Aufgaben zu lösen; Auch Kombinationen beider Ansätze sind möglich.

Ein Beispiel für eine solche Kooperation mehrerer Einzelsysteme zur Erfüllung einer gemeinsamen Aufgabe ist die permanente Beobachtung eines lokalen Ereignisses (beispielsweise einer Umweltkatastrophe) durch mehrere Erdsatelliten, die jeder keine geostationäre Umlaufbahn besitzen: Um eine kontinuierliche Beobachtung zu gewährleisten, muss die Aufgabe des Beobachtens zu verschiedenen Zeitpunkten von verschiedenen Satelliten wahrgenommen werden. Bei einem klassischen Vorgehen würde der Entwickler die Aufgabe in eine Menge von Einzelaufgaben zerlegen, die jeweils auf einem einzelnen Satelliten laufen. Eine Einzelaufgabe würde den jeweiligen Satelliten anweisen, innerhalb eines bestimmten Zeitraums bzw. während der Positionierung innerhalb eines bestimmten Bahnabschnittes eine vorgegebene Beobachtung durchzuführen sowie die Beobachtungsergebnisse zu verarbeiten und nach vorgegebenen Kommunikationsmustern anderen Satelliten oder der Bodenstation mitzuteilen. Die manuelle Realisierung und die nachfolgende manuelle Reintegration der Einzelaufgaben zur Gesamtaufgabe sind jedoch komplex und fehleranfällig. Des Weiteren besteht zur Laufzeit nur eine eingeschränkte Flexibilität, weil z.B. nach dem Ausfall eines Satelliten zunächst ein alternatives Vorgehen vorbereitet werden muss.

Abbildung 1: Koordinierte Beobachtung eines lokalen Ereignisses.

In dem vorliegenden Papier wird als Alternative vorgeschlagen, die Menge der Beobachtungssatelliten als *ein einzelnes* System aufzufassen, dessen Komponenten im Raum zu verschiedenen Zeiten unterschiedliche Positionen einnehmen. Potentiell kann dabei sowohl die Anwendung den zeitlichen Verlauf der Position beeinflussen als auch vice versa. Für ein solches System sind dann geeignete Programmier- und Ausführungsmodelle zu entwickeln, die dem Programmierer eine flexible Beschreibung der auszuführenden Aufgabe auf einem adäquaten Abstraktionsniveau erlauben. Zum Beispiel sollte eine geeignet beschriebene Anwendung sowohl mit zehn als auch mit zwanzig Satelliten ausführbar sein und sich automatisch an den Ausfall oder das Hinzukommen einzelner Satelliten anpassen. Wir geben zu möglichen Programmier- und Ausführungsmodellen erste Denkanstöße.

In jedem derartigen System bestehend aus örtlich verteilten mobilen Komponenten sind bestimmte Aufgaben häufig und wiederholt auszuführen, die dafür prädestiniert sind vom *Betriebssystem* übernommen zu werden. Ein Beispiel hierfür ist das Zuordnen von Teilaktivitäten zu bestimmten Komponenten, die auf diesen zu bestimmten Zeitpunkten zur Ausführung gebracht werden. Wir begründen, warum Anwendungen in realer Raumzeit durch das Betriebssystem unterstützt werden sollten und diskutieren die Anforderungen an ein solches Betriebssystem. Des Weiteren wird das Projekt Flock-OS mit dem Ziel vorgestellt, ein entsprechendes Betriebssystem für eine Gruppe von Nanosatelliten zu entwickeln.

Der Rest des vorliegenden Papiers hat den folgenden Aufbau: In Abschnitt 2.1 erläutern wir den Begriff reale Raumzeit näher. Im Anschluss daran begründen wir in Abschnitt 2.2, warum für Anwendungen in realer Raumzeit eine geeignete Betriebssystemunterstützung sinnvoll ist. In Abschnitt 3 legen wir erste Überlegungen zu möglichen Programmier- und Ausführungsmodellen dar. Schließlich wird in Abschnitt 4 das Projekt Flock-OS vorgestellt. Das Papier schließt mit einer Zusammenfassung und einem Ausblick auf zukünftige Arbeiten (Abschnitt 5).

2. GRUNDSÄTZLICHE ÜBERLEGUNGEN

2.1 Reale Raumzeit

In einem Verband von kooperierenden, mobilen Systemen stehen die Ressourcen einzelner Einheiten nur zu bestimmten Zeiten an bestimmten Orten zur Verfügung. So müssen sich, im Beispiel der Forschungssatelliten ohne geostationäre Umlaufbahn, die einzelnen Satelliten rechtzeitig abwechseln, um die kontinuierliche Beobachtung eines festen Gebiets der Erdoberfläche zu gewährleisten. Insofern bildet das Gesamtsystem, bestehend aus der Zusammenfassung aller Einzeleinheiten, ein *Echtzeitsystem* mit klaren zeitlichen Anforderungen und Fristen. Allerdings birgt die Berücksichtigung des Ortes der Ressourcen, also von realem Raum, eine zusätzliche Qualität. In Analogie zu Echtzeitsystemen ließe sich das Gesamtsystem auch als ein *Echtraumsystem* bezeichnen, da ebenfalls klare Anforderungen an die Position seiner Komponenten gestellt werden. Genauer betrachtet sind jedoch Ort und Zeit voneinander abhängig, weshalb es besser und präziser ist, von *realer Raumzeit* zu sprechen und Anforderungen stets unter Beachtung von Raum und Zeit zu formulieren.

Die Abhängigkeiten zwischen Raum und Zeit können verschiedenartig ausgeprägt sein und von Systemen, deren Komponenten bzw. Subsysteme sich auf gegebenen fixen Trajektorien bewegen, bis hin zu Systemen reichen, in denen die Bewegung der Komponenten durch die Applikation ganz oder teilweise gesteuert wird. Zu ersteren zählen Satelliten auf festen Umlaufbahnen, zu letzteren viele *Schwarmsysteme*, beispielsweise [24]. Interessante Sonderfälle bilden Systeme, in denen sich die Bewegung der Komponenten nur äußerst eingeschränkt voraussagen und beeinflussen lässt, wie z.B. Sensornetze für Tiefseeanwendungen [4].

Ebenfalls stellen unterschiedliche Anwendungen verschiedenartige, spezifische Anforderungen an das System. Anwendungsszenarien können sich von der Beobachtung eines festen Gebietes über die Verfolgung eines beweglichen Objektes bis hin zur Erkundung unbekannter Regionen erstrecken und weitere Nebenbedingungen enthalten, wie z.B. die Berücksichtigung der Winkelabhängigkeit der eingesetzten Sensorik oder die Notwendigkeit der Wahrung des Funkkontakts zwischen einzelnen Systemkomponenten sowie zwischen dem Gesamtsystem und einer Bodenstation. Zusammenfassend müssen sowohl für die Formulierung applikationsspezifischer Anforderungen an das System als auch für die Beschreibung des Systemverhaltens selbst stets *beide* Aspekte, also Raum *und* Zeit, berücksichtigt werden. Zeit oder Raum sind – jeweils einzeln für sich genommen – nicht mehr ausreichend.

2.2 Warum ein neuartiges Betriebssystem?

Weder heutige Betriebssysteme, noch aktuelle Middleware-Plattformen bieten eine adäquate Unterstützung für verteilte Anwendungen in realer Raumzeit. Die essentiellen Konzepte gegenwärtiger Betriebssysteme wie Prozesse, Adressräume, Virtualisierung, Kommunikationsobjekte, etc. stammen aus den sechziger Jahren, also aus einer Zeit, zu der es primär zentralisierte, statisch konfigurierte Rechner gab. Das Betriebssystem war Eigentümer aller Ressourcen, die es nach Bedarfs- und Effizienzaspekten vergab. Bisherige Entwicklungen im Bereich der verteilten Systeme haben die grundlegenden Betriebssystemstrukturen nur geringfügig beeinflusst. Vielmehr wird zusätzlich benötigte Funktionalität für knotenübergreifende Aufgaben in der Regel durch eine weitere Schicht, einer Middleware-Schicht zwischen Betriebssystem und Anwendung, realisiert. Das lokale Betriebssystem eines Knotens bleibt weiterhin autonom bezüglich der Verwaltung seiner Ressourcen und geht davon aus, dass diese dauerhaft zur Verfügung stehen. Die Middlewareschicht bietet vereinfachende Programmierabstraktionen, um beispielsweise Aspekte wie Heterogenität, Verteilung oder Mobilität zu verbergen.

Jedoch sind die bisherigen, oben genannten Grundannahmen und -vorstellungen für verteilte, mobile Systeme mit Anwendungen in realer Raumzeit nicht mehr zutreffend. Über viele Ressourcen können einzelne Knoten nicht mehr dauerhaft und autonom verfügen – eine Koordination mit anderen Knoten wird zwingend erforderlich. Beispielsweise mag die Möglichkeit der Kommunikation sowohl von der Position des Knotens abhängen, als auch von der Lage anderer Knoten relativ zu ihm. Ferner ist Mobilität nicht mehr als Aspekt anzusehen, der vor der Anwendung verborgen werden muss, sondern als spezielle zu nutzende Eigenschaft, die

erfasst sowie repräsentiert werden muss und eventuell sogar beeinflussbar oder programmierbar ist.

Anwendungen in realer Raumzeit erfordern neue Konzepte und Denkweisen, deren Unterstützung inhärent im System verankert werden sollte. Eine Umsetzung könnte durch eine (weitere) Middleware-Schicht geschehen, doch sprechen nachfolgende Gründe dafür, die Realisierung möglichst tief und zwar bereits auf Ebene des Betriebssystems anzusiedeln:

- Das Ziel ist, für Anwendungen in realer Raumzeit eine Ausführungsumgebung bereitzustellen; dies entspricht dem grundlegenden Zweck eines Betriebssystems.
- Der Betrieb und die Verwaltung von Ressourcen sind essentielle Aufgaben eines Betriebssystems. In einem System mobiler, kooperierender Einheiten erweisen sich zeitliche und räumliche Einschränkungen in der Verfügbarkeit von Ressourcen und die dadurch erforderliche Notwendigkeit der Koordinierung über die Grenzen einzelner Einheiten hinaus als neue Qualitäten. Diese werden sich beispielsweise in genutzten Schedulingalgorithmen und Zugriffsverfahren widerspiegel müssen.
- Ein System für Anwendungen in realer Raumzeit ist in jedem Fall ein Echtzeitsystem. Deshalb verlangt es – wie jedes Echtzeitsystem – die Möglichkeit eines feingranularen Zugriffs auf einzelne Ressourcen, die in der Regel nur auf Systemsoftwareebene gewährleistet werden kann.
- Anwendungen für verteilte, mobile Systeme in realer Raumzeit verlangen neue Programmierkonzepte bezüglich Aktivitäten und Datenrepräsentation. Beides sind Konzepte, die in einem Betriebssystemkern verankert werden müssen, auch wenn sie auf höherer Ebene anders repräsentiert werden.[1]

3. MODELLE

Ein System für Anwendungen mit realer Raumzeit verlangt spezifische Ansätze zur Modellierung. Dabei ist zwischen zwei Anwendungen von Modellen zu unterscheiden:

- Modelle zur Unterstützung der Argumentation über Korrektheit und Möglichkeiten des Systems;
- Programmiermodelle, die es dem Programmierer ermöglichen, auf einem angemessenen Abstraktionsniveau die Anwendungen zu beschreiben (vgl. Abschnitt 2.2).

Idealerweise sind beide Arten von Modellen aufeinander abbildbar oder sogar identisch. Ein Beispiel für ein solches Modell ist das zyklische Taskmodell, das häufig im Echtzeitrechnen eingesetzt wird und Grundlage vieler Echtzeitschedulingverfahren ist (z.B. Rate Monotonic Scheduling (RMS) und Earliest Deadline First (EDF) [11]). In diesem Modell wird davon ausgegangen, dass eine Task zyklisch aufgerufen wird und, dass für jeweils eine Taskinstanz die Deadline durch das Auftreten der nächsten Instanz gegeben ist. Obwohl es sowohl für die Beschreibung von Echtzeitbedingungen, als auch für die Echtzeitausführung eine Vielzahl von anderen (in der Regel sogar mächtigeren) Möglichkeiten gibt, hat sich dieses Modell durchgesetzt: Die Mehrzahl der Echtzeitbetriebssysteme unterstützt das zyklische Taskmodell und RMS.

[1]Man betrachte in gängigen Betriebssystemen die Abbildung von Nutzer-Prozessen auf Kernelthreads.

Für ein Betriebssystem, das Anwendungen in realer Raumzeit unterstützen soll, ist die Modellierung von Mobilität, Raum und Zeit notwendig. Es gibt in der Informatik verschiedene Ansätze zur Modellierung von Mobilität. Viele von ihnen sind in erster Linie zur Darstellung von Nebenläufigkeit gedacht und erlauben die Modellierung von Mobilität nur implizit, wie z.B. Petri-Netze [16], Algebraic Process Calculus (ACP) [2] oder Communicating Sequential Processes (CSP) [9]). Nur wenige Ansätze, wie das Actor-Modell [8], das π-Kalkül [12, 13] oder das Ambient-Kalkül [3], besitzen eine explizites Konzept von Mobilität. Jedoch sind alle diese Modelle metrik-frei – sie kommen ohne eine explizite Beschreibung von Raum oder von Zeit aus.

Die Einbeziehung von (realer) Zeit gibt es in zahlreichen Modellen. Häufig handelt es sich um Echtzeiterweiterungen bestehender Modellansätze. So existiert beispielsweise ein echtzeiterweitertes CSP (timed CSP) [19], und in [1] wird das π-Kalkül um (diskrete) Zeit erweitert. Jedoch ist den Autoren des vorliegenden Papiers kein Ansatz bekannt, der eine streng-formale Modellierung und Argumentation über Berechnungen in realer Raumzeit ermöglicht. Es ist ein Forschungziel der Autoren, einen entsprechenden Formalismus zu entwickeln.

Ebenso gibt es bisher kaum Ansätze für geeignete Programmiermodelle. Zwar ist das Konzept von lokalitätsbezogenen Daten und Berechnungen nicht neu, aber in der Regel sind diese Ansätze sehr anwendungsspezifisch (vgl. z.B. [15], [5] oder [23]). In ortsgebundenen Diensten (s. z.B. [18]) wird auf realen Raum (und z.T. auch auf reale Zeit) Bezug genommen, jedoch ohne jede Verteiltheitsaspekte. Auch die Programmier-Modelle verteilter und teilweise Mobilität unterstützender Betriebssysteme oder Betriebssystemerweiterungen – wie z.B. Amoeba [21], Plan 9 [17] oder Emerald – [10] abstrahieren von realem Raum. Der für die hier präsentierte Arbeit vermutlich interessanteste Ansatz ist in [6] zu finden. Dort werden virtuelle stationäre Automaten (*virtual stationary automata*, VSA) vorgestellt. Diese sind an (durch GPS bestimmte) örtliche Bereiche gebunden. Halten sich Teilnehmer einer Menge mobiler Knoten innerhalb eines solchen Bereiches auf, übernehmen sie in Vertretung die Ausführung des an den Bereich gebundenen VSAs.

Wir nutzen ein ähnliches Modell: Die Instanz einer Anwendung – *Aktivität* genannt – unterliegt raumzeitlichen Beschränkungen. Typischerweise wird eine solche Beschränkung als Hüllraum um eine Raum-Zeit-Trajektorie angegeben (*spacetime constraints*). Auch ein feststehender Bereich kann hierbei als ein Spezialfall eines solchen Hüllraums beschrieben werden. Je nach Anwendungsfall wird ein solcher Hüllraum dreidimensional (zwei Raumkoordinaten und Zeit) oder vierdimensional (drei Raumkoordinaten und Zeit) angegeben. Die Komponenten eines verteilten mobilen Systems stellen einer Aktivität ihre Ressourcen zur Verfügung. Dabei bewegt sich die Identität einer Ressource mit der Aktivität. Beispielsweise ist der Beobachtungssensor für das Naturereignis aus dem im Abschnitt 1 beschriebenen Anwendungsfall aus Sicht der Aktivität stets identisch – auch wenn er durch die Sensoren verschiedener Satelliten repräsentiert wird. Die Constraints müssen nicht für die gesamte Aktivität angegeben werden, sondern können für einzelne Ressourcen (Daten, I/O-Geräte) einzeln spezifiziert werden. Durch die

Beschreibung von Raumzeit-Bedingungen können implizit Echtzeitbedingungen definiert werden. Die Ausführbarkeit (*feasibility*) dieser Bedingungen ist im allgemeinen Fall nicht trivial zu prüfen – für Spezialfälle (einfache Zyklen in Raum und Zeit) lassen sich jedoch einfache Ausführbarkeitstests angeben.

4. FLOCK-OS

Forschung auf dem Gebiet verteilter Betriebssysteme findet seit den 70er Jahren des vorigen Jahrhunderts statt. Bekannte Vertreter sind Amoeba [21] oder Plan 9 [17]. Diese Systeme gehen jedoch von einer statischen Verteiltheit aus. Eine Komponente existiert an einem bestimmten Ort, von dem weitgehend abstrahiert wird. Lokalitätsmetriken kommen gegebenenfalls in Form von Kommunikationskosten vor und finden beispielsweise bei der Platzierung von Threads in einem Parallelrechnersystem Anwendung. Betriebssysteme, die häufig für Sensornetzwerke eingesetzt werden (z.B. [22]), berücksichtigen den Verteiltheitsaspekt in der Regel überhaupt nicht. Auch SOS [7] oder JaMOS [20], die explizit für Sensornetz- und Schwarmanwendungen entworfen wurden, sind auf Minimalität bezüglich Sensorunterstützung ausgelegt.

Deshalb entwickeln der Lehrstuhl für Betriebssysteme der TU Chemnitz und die Gruppe Kommunikations- und Betriebssysteme der TU Berlin ein verteiltes Betriebssystem, das die diskutierten Aspekte realisieren soll: das Flock-OS (***f**ederation of **l**inked **o**bjects with **c**ommon tas**k**s*). Der folgende Abschnitt beschreibt gegenwärtige Überlegungen zum Entwurf – da Programmiermodell (vgl. Abschnitt 3) und Algorithmen (insbesondere Scheduling) in realer Raumzeit noch Gegenstand der Forschung sind, ist diese Beschreibung als vorläufig zu betrachten.

Ähnlich wie gängige verteilte Betriebssysteme besteht Flock-OS aus Nanokernen auf jedem aktiven Systemknoten. Im Fall von Flock-OS handelt es sich um echtzeitfähige Nanokerne. Im Unterschied zu anderen verteilten Betriebssystemen gibt jeder Knoten jedoch seine Autonomie weitgehend auf und unterstellt sich der Gemeinschaft des Gesamtsystems. Dieses wird durch ein Ensemble von Peers gebildet (in Flock-OS *federation* genannt), die in Kooperation gemeinsame Ziele verfolgen. Das Flock-OS bietet situative dynamische Funktions- und Rollenzuteilung basierend auf Position, Bewegung und Energieverbrauch und verwendet situations- und anwendungsbezogene Kommunikationsmuster. Es unterstützt mehrere verteilte Anwendungen (Multiprogramming) und ihre Einplanung in realer Raumzeit.

Abbildung 2 zeigt die grundsätzliche Architektur von Flock-OS. Die Nanokerne der einzelnen Knoten sind verantwortlich für die lokale Ausführung von (Teil-)Aktivitäten und bieten Zugriff auf lokale Ressourcen und Geräte des Knotens. Die über den Nanokerne angesiedelte Kommunikationsschicht ermöglicht den Datenaustausch zwischen den einzelnen Kernen und ist Grundlage jeder knotenübergreifenden Koordinierung. Mit Hilfe der Kommunikationschicht und gegebenenfalls lokaler Sensorik wird die raumzeitliche Position jedes Knotens bestimmt. Sofern möglich (z.B. bei vorhandenem GPS) geschieht die Bestimmung absolut, sonst wird die relative Position der Knoten zueinander ermittelt. Für jede Aktivität wird ein Positionskonsens hergestellt, der entsprechend der Raumzeitanforderungen und der erwarteten Drift aktualisiert wird. In den Konsens sind nur die Knoten einbezogen, die bis zur nächsten Konsensrunde theoretisch von einer Aktivität betroffen sein könnten. Ist das System eher klein oder besitzen die meisten Aktivitäten eine systemweite Ausdehnung, kann das Flock-OS in einen Modus geschaltet werden, in dem stets ein globaler Positionskonsens durchgeführt wird.[2]

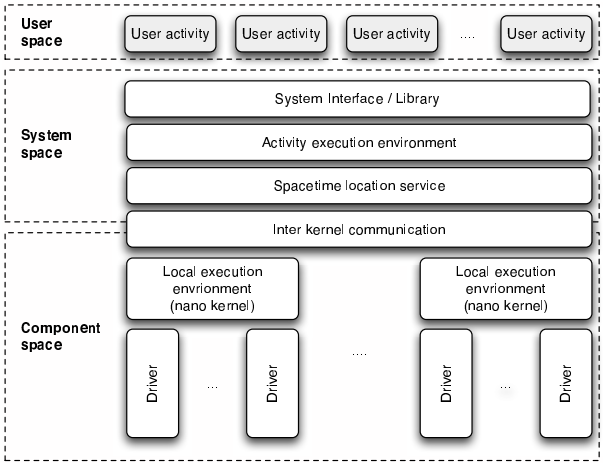


Abbildung 2: Architektur des Flock-OS.

Das *Activity Execution Enviroment* ist als der eigentlich (verteilte) Kern des Flock-OS anzusehen. Hier erfolgt das Einplanen (Scheduling) von Ausführungen und Aktualisierungen von Daten entsprechend den gegebenen Anwendungsbedingungen (*constraints*, siehe Abschnitt 3). Hierfür liegen Programmcode und Daten repliziert auf den einzelnen Knoten vor. Für die Aktualisierung der Daten wird eine spekulative[3] *lazy consistency policy* genutzt. Beim Aktivitätsscheduling wird ein lokaler Kontext auf dem Förderationsmitglied aufgeweckt, der für eine Aktivität eingeplant ist. Gibt dieses Mitglied die Aktivität ab, wird der Kontext blockiert. Bei Speicherknappheit und längerer Nichtbeteiligung an einer Aktivität kann ein lokaler Kontext auch zerstört und gegebenenfalls neu kreiert werden.

Für die Programmierung wird dem Nutzer neben einer Kernelschnittstelle auch eine Bibliothek mit vielen Standardoperationen (z.B. für Trajektorienverfolgung, Formationsbildung) zur Verfügung gestellt. Auch ein direkter Komponentenzugriff mit Hilfe von Bibliotheksroutinen ist möglich.

5. SCHLUSSFOLGERUNGEN

Zunehmende Mobilität und Kommunikationsmöglichkeiten bringen neuartige verteilte Systeme hervor. Viele Anwendungen verlangen die raumzeitliche Kooperation von Komponenten in diesen Systeme – eventuell bewirkt auch erst die

[2]Hierfür wird eine Pseudo-Aktivität gestartet, die keinen eigenen Programmcode besitzt, aber stets über alle Förderationsmitglieder definiert ist.

[3]Bei den bisher betrachteten Anwendungsfällen sind die Raumzeitanforderungen weitgehend *a priori* bekannt, deshalb beschränkt sich die Vorhersage auf die Nutzung der A-priori-Daten.

Anwendung selbst, dass eine Menge von eher separaten Einheiten als *ein* Gesamtsystem aufgefasst wird. Wir haben in dieser Veröffentlichung die Idee eines Betriebssystems vorgestellt, dass ein solches aus mobilen Komponenten bestehendes Gesamtsystem betreibt und Anwendungen unterstützt, die in realer Raumzeit ausgeführt werden.

Wir haben Probleme realer Raumzeitanwendungen diskutiert und ferner analysiert, warum für solche Anwendungen eine Betriebssystemunterstützung vorteilhaft ist. Es wurden Möglichkeiten der Modellierung realer Raumzeitanwendungen vorgestellt, einerseits für die formale Erörterung von Korrektheit sowie andererseits als Abstraktion zur Programmierung. Als einen Ansatz für ein solches Programmiermodell wurde eine Erweiterung der virtuellen stationären Automaten aus [6] besprochen. Schließlich wurde Flock-OS vorgestellt, ein verteiltes, in der Entwicklung befindliches Betriebssystem, das Anwendungen in realer Raumzeit unterstützt.

Die hier dargestellten Ideen und Konzepte befinden sich z.T. noch in den Anfängen, aber es ist bereits jetzt ersichtlich, dass es sich um ein herausforderndes Forschungsgebiet mit vielen Anwendungsmöglichkeiten handelt, das großen Raum für weitere Untersuchungen lässt. Als nächster Schritt soll das Aktivitätsmodell so überarbeitet werden, dass leichtere Schedulingtests für eine größere Menge von Anwendungsfällen möglich werden. Ferner soll ein Formalismus gefunden und sofern praktikabel in das Programmiermodell integriert werden. Dieser soll eine allgemeine Beschreibung und Erörterung von Aktivitäten in Raumzeit zulassen. Parallel wird eine erste konkrete Implementierung von Flock-OS realisiert, wobei als lokaler Echtzeitkernel voraussichtlich eine modifizierte Variante von BOSS [14] eingesetzt wird.

6. LITERATUR

[1] Berger, Martin. „Towards Abstractions for Distributed Systems“. Diss. Imperial College, Deptartment of Computing, 2002.

[2] Bergstra, J.A., und J.W. Klop. „Process algebra for synchronous communication“. In: *Information and Control* 60.1-3 (1984). 109–137.

[3] Cardelli, L., und A. Gordon. „Mobile ambients“. In: *Foundations of Software Science and Computation Structures 1998*. Bd. 1378. Lecture Notes in Computer Science. Springer, 1998. 140–155.

[4] Cui, Jun-Hong, u. a. „Challenges: Building Scalable Mobile Underwater Wireless Sensor Networks for Aquatic Applications“. In: *IEEE Network, Special Issue on Wireless Sensor Networking* 20.3 (2006). 12–18.

[5] Dolev, S., u. a. „Geoquorums: Implementing atomic memory in mobile ad hoc networks“. In: *Proceedings of the 17th International Symposium on Distributed Computing (DISC 2003)*. 2003. URL: `citeseer.ist.psu.edu/article/dolev03geoquorums.html`.

[6] Dolev, Shlomi, u. a. Virtual Stationary Automata for Mobile Networks. MIT-CSAIL-TR-2005-004. Techn. Ber. Massachusetts Institute of Technology, Computer Science, Artificial Intelligence Laboratory, 2005.

[7] Han, Chih-Chieh, u. a. „SOS: A dynamic operating system for sensor networks“. In: *MobiSYS 05: 3rd international conference on Mobile systems, applications, and services*. ACM Press, 2005. 163–176.

[8] Hewitt, Carl, Peter Bishop und Richard Steiger. „A Universal Modular ACTOR Formalism for Artificial Intelligence“. In: *International Joint Conferences on Artificial Intelligence*. 1973. 235–245.

[9] Hoare, Charles Antony Richard. „Communicating sequential processes“. In: *Communications of the ACM* 21.8 (1978). 666–677.

[10] Jul, Eric, u. a. „Fine-grained mobility in the Emerald system“. In: *ACM Transactions on Computer Systems* 6.1 (1988). 109–133.

[11] Liu, C. L., und James W. Layland. „Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment“. In: *Journal of the ACM* 20.1 (Jan. 1973). 46–61.

[12] Milner-Gulland, Robin. *Communicating and Mobile Systems: The Pi Calculus*. Cambridge University Press, 2004.

[13] Milner, Robin, Joachim Parrow und David Walker. „A Calculus of Mobile Processes“. In: *Information and Computation* 100.1 (1992). 1–40 and 41–77. Part I and II.

[14] Montenegro, S., K. Briess und H. Kayal. „Dependable Software (BOSS) for the BEESAT pico satellite“. In: *DAta Systems In Aerospace - DASIA 2006*. 2006. 3–8.

[15] Nath, B., und D. Niculescu. „Routing on a curve“. In: *ACM SIGCOMM Computer Communication Review* 33.1 (2003). 150–160.

[16] Petri, Carl Adam. *Kommunikation mit Automaten*. Bonn: Institut für Instrumentelle Mathematik, Schriften des IIM Nr. 2, 1962.

[17] Pike, R., u. a. „Plan 9 from Bell Labs“. In: *Computing Systems* 8.3 (1995). 221–254.

[18] Schiller, J. H., und A Voisard. *Location-based services*. Morgan Kaufmann Publishers, 2004.

[19] Schneider, S. *Concurrent and Real-time Systems: The CSP Approach*. John Wiley & Sons, 1999.

[20] Szymanski, Marc, und Heinz Wörn. „JaMOS - A MDL2ϵ based Operating System for Swarm Micro Robotics“. In: *IEEE Swarm Intelligence Symposium*. 2007. 324–331.

[21] Tanenbaum, A.S., u. a. „Experiences with the Amoeba Distributed Operating System“. In: *Communications of the ACM* 33 (1990). 46–63.

[22] *TinyOS*. UC Berkeley. 2004. WWW-Seite. URL: `http://www.tinyos.net` (besucht am 14. 09. 2007).

[23] Wegener, Axel, u. a. „Hovering Data Clouds: A Decentralized and Self-Organizing Information System“. In: *Proceedings of the 1st International Workshop on Self-Organizing Systems (IWSOS 2006)*. Bd. 4124. Lecture Notes in Computer Science. Passau, Germany: Springer, 2006. ISBN 978-3-540-37658-3. DOI: `10.1007/11822035_22`. 243–247.

[24] Wörn, Heinz. *Intelligent Small World Autonomous Robots for Micro-Manipulation (I-SWARM)*. Universität Karsruhe. 2004. WWW-Seite. URL: `http://cordis.europa.eu/fetch?CALLER=PROJ_IST&ACTION=D&RCN=71243` (besucht am 14. 09. 2007).

Modeling energy consumption of wireless communications in OMNeT++

Isabel Dietrich, Feng Chen, Reinhard German and Falko Dressler

Computer Networks and Communication Systems
University of Erlangen-Nürnberg, Germany
{isabel.dietrich,feng.chen,german,dressler}@informatik.uni-erlangen.de

ABSTRACT

We present an energy model for the simulation framework OMNeT++ that has been calibrated using several measurements for real sensor nodes. This energy model allows to study sensor network algorithms and applications in a simulation model with high quality energy estimations. The energy model can be adapted for arbitrary sensor nodes based on respective measurement data. We show the applicability of the energy model based on two scenarios: the analysis of the energy performance of IEEE 802.15.4 and the evaluation of the advantages of on-demand sensor network reprogramming.

1. INTRODUCTION

In the last decade, many approaches have been proposed that improve the performance of sensor networks. Some of the most challenging issues that have been studied are the medium access, routing strategies, clustering schemes, and application layer dynamics. All these approaches contribute to the final objective to enable designers to develop and to deploy applications under various environmental conditions. The idea is to provide a broad range of design variants that can be chosen and combined in order to provide the optimal behavior of the sensor network.

All the individual algorithms and techniques have been analyzed regarding their performance, e.g. the speed of adaptation to environmental changes, the end-to-end performance, the produced overhead, and the energy consumption. Studying especially solutions optimizing the energy performance – or the entire network lifetime [3] –, we realized that varying techniques are used for evaluation and analysis of the developed solution.

Basically, we need to distinguish between experimentation and simulation as evaluation techniques. In general, it seems that in most cases, only one of these techniques has been used. This has a number of drawbacks that will become obvious in a short comparison. Simulation allows to study developed methods and techniques without the need of really deploying sensor nodes – that may not yet exist. In many evaluated scenarios, experimentation will be too expensive or not possible at all, e.g. for deployment scenarios in hazardous environments. Also, simulation allows the evaluation of really large networks, which will be infeasible in a lab. On the other hand, experimentation allows to study sensor networks in a real world environment facing typical radio transmission problems and others. Therefore, performance evaluation is usually based on simulation models. Nevertheless, measurements are necessary to calibrate simulation models.

In the last years, a number of research groups started to provide basis measures to be used to improve the quality of sensor network simulation. This includes performance measures of the typical micro controllers, wireless transmission, and energy measures. Examples are the measurements for Mica2 sensor nodes by Landsiedel et al. [6] as depicted in Figure 1 and the analysis of various communication energy measures for normal and encrypted communication in a security-enhanced scenario by Chang et al. [1] as shown in Figure 2.

Device	Current	Device	Current
CPU		**Radio** (900 MHz)	
Active	$7.6\,mA$	Core	$60\,\mu A$
Idle	$3.3\,mA$	Bias	$1.38\,mA$
ADC Noise	$1.0\,mA$	Rx	$9.6\,mA$
Power down	$116\,\mu A$	Tx (-18 dBm)	$8.8\,mA$
Power Save	$124\,\mu A$	Tx (-13 dBm)	$9.8\,mA$
Standby	$237\,\mu A$	Tx (-10 dBm)	$10.4\,mA$
Ext Standby	$243\,\mu A$	Tx (-6 dBm)	$11.3\,mA$
		Tx (-2 dBm)	$15.6\,mA$
LED (each)	$2.2\,mA$	Tx (0 dBm)	$17.0\,mA$
		Tx (+3 dBm)	$20.2\,mA$
Sensor Board	$0.7\,mA$	Tx (+4 dBm)	$22.5\,mA$
		Tx (+5 dBm)	$26.9\,mA$

Figure 1: Energy consumption of Mica2 sensor nodes [6]

In general, energy consumption of sensor networks has been studied manifold. One of the most important observations was that there is a strong contrast between energy consumption for communication and computation. Depending on the source in the literature a factor of 1.000 up to 100.000 needs to be considered. In many current simulation models, only sending activities are counted. Nevertheless, depending on the used duty-cycle and the message rate, message reception and idle listening can be even more expensive than sending. Thus, we need better models for energy consump-

Message Length (Bytes)			8	16	24	32
No Security	CPU		3	4	4	4
	TX		945	1113	1281	2226
CPU and Transmit			948	1117	1285	2230
Hash	CPU	SHA-	154	154	154	154
	TX		2142	2310	2478	3423
Hash and Transmit			2296	2464	2632	3577
Encrypt	CPU	RC5	111	124	137	150
	CPU	DESC	53	79	103	126
	CPU	AES	339	339	339	339
Encrypt and Transmit		RC5	1056	1237	1418	2376
		DESC	998	1192	1384	2352
		AES	1248	1452	1620	2565
Hash, Encrypt & Transmit		RC5	2253	2434	2615	3573
		DESC	2195	2389	2581	3549
		AES	2481	2649	2817	3762

Figure 2: Energy consumption of normal and encrypted communication [1]

tion in our current simulation tools.

In this paper, we contribute to the current research by presenting an energy model for the OMNeT++ simulation framework [7]. We calibrated the model using reference measures provided by other groups as described below. We also show two application scenarios, in which we used the model to analyze the performance of the IEEE 802.15.4 model and the advantages of on-demand sensor network reprogramming, respectively.

2. OUR ENERGY MODEL

In this section, we describe the functionalities and implementation details of the energy model developed for OMNeT++. This simulation framework provides a discrete event simulation environment with support for many network protocols such as WLAN (IEEE 802.11), TCP/IP, and many others. Additionally, mobility models and traffic models are available. Recently a number of models for ad hoc and sensor network protocols have been integrated including the ad hoc routing protocol DYMO (dynamic MANET on demand) and IEEE 802.15.4, the MAC protocol for ZigBee.

Our energy model is developed as a protocol-independent module and serves as a plug-in to various wireless protocol models in OMNeT++. It adopts the initial battery energy, the radio power in different working state, and the CPU power as its input parameters. Based on these configurations, the model continuously performs the calculation of the energy consumption both on radio and CPU in real-time. If needed, it displays the remaining energy level in animations during the simulation running. Depending on the purpose of the study, the energy model can be configured to execute one of the following two actions upon exhaustion of battery power:

- The simulation is terminated when the first node exhausts its battery power.
- The simulation keeps running until a specified node (e.g. the central node) dies or all nodes in the network die. The dead nodes have to be cut from the network communication, which is maintained by all other active nodes. In our model, we implement this by dynamically disconnecting the *radioIn* gate of the dead node from the channel module and reconnecting it to an empty gate.

In our energy model, energy consumption on both radio and CPU is considered. Since the energy consumption of the wireless communications is differentiated depending on the current radio state, a proper radio model defining various working states is necessary. Our energy model supports a usual four-state radio interface that are known to lead to a different energy consumption of the node for most hardware platforms:

- Idle
- Sleeping
- Transmitting
- Receiving

To calculate the energy consumed by the radio in real-time, the energy model tracks every state switch in the PHY module using the OMNeT++ notification board. This board allows to centrally observe distributed events. Upon receiving state switch event from the notification board, the energy model updates the accumulated time for each radio state and recalculates the current energy consumption. If no battery power is found left after the recalculation, one of the above mentioned two actions will be executed.

Estimating and modeling energy consumption on the CPU is much more difficult than doing this on the radio, because the activity of the CPU is complex and depends on a couple of factors. For instance, CPU will be busy while processing a packet just received by the MAC or executing some encryption or decryption algorithms. It can also be idle while the radio is busy transmitting or sleeping. Therefore, we can only consider a rough approximation. In our model, we define two CPU states, active and inactive. It is assumed that the CPU will follow the same sleeping schedule of the radio interface, which means that CPU is inactive only during the radio sleeping period.

Finally, the model needs to be calibrated for specific systems (hardware modules). This is done based on measurements as presented before. Here, we need to mention that the degree of details strongly depends on requirements because the energy model consumes processing time in the simulation. Further details of the energy model are discussed in the following section that outlines two application examples.

3. APPLICATION EXAMPLES

3.1 Energy performance of IEEE 802.15.4

Based on a new simulation model of the ZigBee MAC protocol IEEE 802.15.4, we analyzed the performance of this protocol. Some results from these measurements are presented in the following. The simulation model itself is described in [2].

IEEE 802.15.4 defines MAC and physical layers for low-rate wireless personal area networks (LR-WPANs) [5] and the upper layers to form a complete network stack built are specified by ZigBee. The objective of IEEE 802.15.4 is to enable low-cost communication between devices. In particular, the physical layer allows data rates up to 250 kBit/s. The MAC layer provides collision avoidance with CSMA/CA as

well as real-time support by reservation of guaranteed time slots. Beaconing is used for synchronization between devices.

At the MAC layer, a superframe structure may be defined by the PAN coordinator that controls an entire network. The structure of a superframe is depicted in Figure 3.

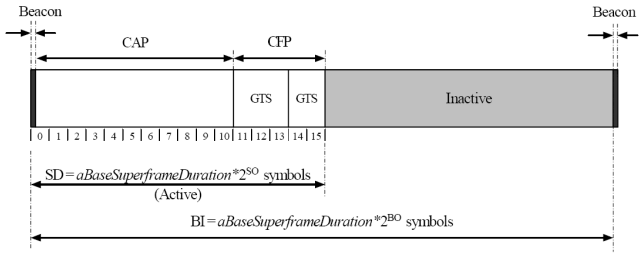


Figure 3: Superframe structure of the MAC layer of IEEE 802.15.4

The implementation in OMNeT++ is outlined in Figure 4. It consists of a PHY and a MAC model plus several supporting models including an interface queue (IFQ) and our energy model.

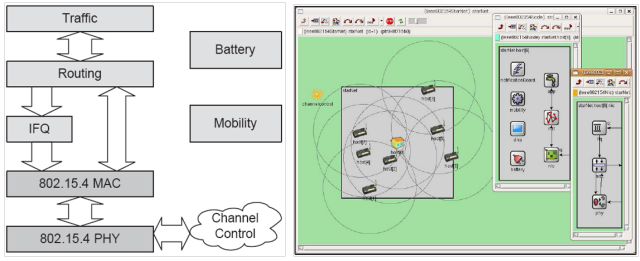


Figure 4: Scheme (left) and implemented models (right) of the IEEE 802.15.4 model

As an example, we analyzed a three node network topology. We connected two devices to a PAN coordinator. The first device is generating packets and sending them to the second device via the PAN coordinator. We analyzed different combinations of BO/SO. The results are shown in Figure 5. Please note that the duty cycle remains constant at 50% for the given BO/SO combinations. Shown is the energy consumption per successfully transmitted byte for different traffic rates ($0.01 \ldots 10\text{s}^{-1}$) on a log scale. Obviously, the energy consumption is higher, the lower the traffic rate is. The reasons lies in the long active periods in which no data is transmitted. Thus, this example shows that modeling the CPU energy consumption is essential for evaluating network protocols.

3.2 Network lifetime

In another experiment, we studied the lifetime of dynamically reprogrammed sensor networks. The complete setup including a description of the concept and the main ideas are presented in [4]. In short, we prepared a sensor network with three types of sensors. All the sensor nodes gather data and forward them to a central base station (using WLAN and the ad hoc routing AODV). Mobile robot systems are used for on-demand sensor node reprogramming. In particular, the robots continuously check the application requirements

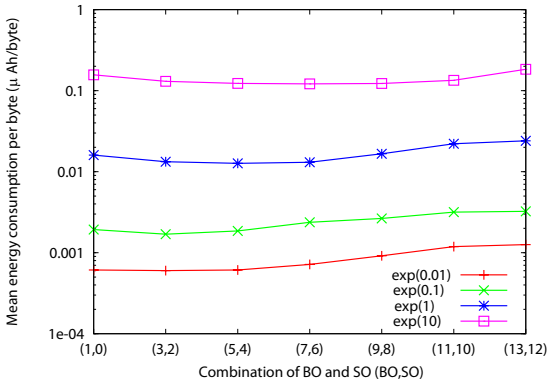


Figure 5: Energy consumption of IEEE 802.15.4 for different traffic rates and different (BO,SO) combinations

and, if necessary, they identify a spare sensor node that can be reprogrammes. We analyzed the sensor network lifetime according to a set of different setups. The programmed application requirement is that each of the four sectors of the network needs at least two of each sensor type to guarantee a sufficient degree of coverage. The scenario is shown in Figure 6.

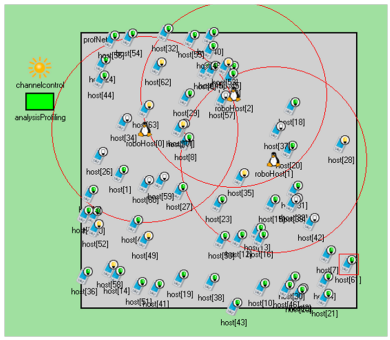


Figure 6: Simulation setup for the lifetime measurements

Some selected measurement results are shown in Figure 7. In order to demonstrate the features of the energy model, we selected three different node programs that lead to a different energy consumption of the node per time:

- P0: simple sensor, measurement cycle is 60 s
- P1: simple sensor, measurement cycle is 10 s
- P2: complex sensor (additional energy consumption for each measurement), measurement cycle is 10 s

Two reprogramming strategies were used by the mobile robot systems: random selection of a nearby node and selection of the node with the most remaining energy. Also, we changed the number of robots (0, 1, 3) and the initial programming of the network was randomly chosen. We analyzed the percentage of network operable time compared to the complete simulation time (the simulation terminated if the application demands cannot be fulfilled any further).

	% never operable	Min	Mean	Max
no robot, Energy	65	0	31.34	35.28
no robot, Random	65	0	31.34	35.28
1 robot, Energy	0	49.5	84.26	86.14
1 robot, Random	5	0	82.85	82.85
3 robots, Energy	0	81.83	88.1	89.88
3 robots, Random	0	88.65	88.85	91.98

Figure 7: Lifetime of the sensor network

4. CONCLUSION

We described a new energy model for use within the OMNeT++ simulation framework. This model allows to evaluate the energy performance and, thus, the network lifetime for arbitrary sensor network applications. In particular, we presented two application examples that we analyzed in related work using the described model. Currently, we calibrated the energy model for sensor nodes of type Mica2 according to measurement results available in the literature. Further types of hardware can be supported according to adequate measurement results. Currently, the model supports four different radio states and we also support the modeling of CPU intensive operations. Future work include fine granular CPU state modeling and support for a wider range of sensor systems.

5. REFERENCES

[1] C.-C. Chang, D. J. Nagel, and S. Muftic. Assessment of Energy Consumption in Wireless Sensor Networks: A Case Study for Security Algorithms. In *4th IEEE International Conference on Mobile Ad Hoc and Sensor Systems (IEEE MASS 2007): 3rd IEEE International Workshop on Wireless and Sensor Networks Security (WSNS'07)*, Pisa, Italy, October 2007.

[2] F. Chen and F. Dressler. A Simulation Model of IEEE 802.15.4 in OMNeT++. In *6. GI/ITG KuVS Fachgesprch Drahtlose Sensornetze, Poster Session*, pages 35–38, Aachen, Germany, July 2007.

[3] I. Dietrich and F. Dressler. On the Lifetime of Wireless Sensor Networks. Technical Report 04/06, University of Erlangen, Dept. of Computer Science 7, December 2006.

[4] F. Dressler and I. Dietrich. Lifetime Analysis in Heterogeneous Sensor Networks. In *9th EUROMICRO Conference on Digital System Design - Architectures, Methods and Tools (DSD 2006)*, pages 606–613, Dubrovnik, Croatia, August 2006.

[5] IEEE. Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low Rate Wireless Personal Area Networks (WPANs). IEEE Sandard 802.15.4-2006, 2006.

[6] O. Landsiedel, K. Wehrle, and S. Gtz. Accurate Prediction of Power Consumption in Sensor Networks. In *Second IEEE Workshop on Embedded Networked Sensors (EmNetS-II)*, Sydney, Australia, May 2005.

[7] A. Varga. The OMNeT++ Discrete Event Simulation System. In *European Simulation Multiconference (ESM 2001)*, Prague, Czech Republic, 2001.

Exact Localization on Resource Limited Sensor Nodes - Making it Feasible -

Frank Reichenbach, Dirk Timmermann
Institute of Applied Microelectronics and Computer Engineering,
University of Rostock, Warnemuende, Germany
E-mail: {frank.reichenbach; dirk.timmermann}@uni-rostock.de

ABSTRACT

The random deployment process and the unpredictable movement of sensor nodes lead to a high demand for an exact and reliable self localization process. Existent methods are mostly not feasible on strongly resource limited sensor nodes with an absolute minimum of energy. Thus, this paper describes the "Distributed Least Squares (DLS)"-algorithm that reduces the needed amount of energy to a minimum by distributing the complex localization process. DLS is a resource-aware localization method that achieves 47% computation savings and 86% energy savings compared to the reference method, a "Fully Distributed Multilateration" on every sensor node. In the end, DLS significantly extends the overall network lifetime.

Categories and Subject Descriptors

C.2.4 [**Distributed Systems**]: Distributed applications

General Terms

Algorithms, Design, Theory

Keywords

wireless sensor networks, localization

1. INTRODUCTION

Wireless sensor networks (WSN) are composed of hundreds of tiny electronic devices, able to sense the environment, compute simple tasks and communicate with each other. Gathered information (e.g. temperature, humidity etc.) are transmitted in a multi hop fashion over direct neighbors to a data sink, where the data is interpreted [1]. With methods such as self configuration and self organization the network reacts to node failures.

Due to the desired node size of only a few cubic millimeters, the dimensions of the communication module and the battery are critical. Consequently, the scarcest resource within a network is the available energy [2]. Therefore, achieving a long lifetime of the sensor network requires low power hardware as well as optimized algorithms.

After deploying the sensor network over an area of interest, initially the sensor nodes carry no position information. Sensor information are only useful if combined with their geographical position. Possible positioning technologies are the Global Positioning System (GPS), the Global System for Mobile Communication (GSM) or soon the European System Galileo [3],[4],[5]. However, these systems are unsuitable for miniaturized sensor nodes and could only be used for a small number of nodes, due to the size of the hardware, the high prices and the high energy requirements. Thus, it is a common technique to integrate an existing localization system on some more powerful nodes, further called beacons. Then, all remaining nodes estimate their own position with measurements such as distances to these beacons autonomously. A node's position is very important, because (i) sensed data without a location where they were gathered are generally useless, (ii) full covered sensor networks enable energy aware geographic routing, (iii) self configuration and self organization are key mechanisms for robustness and can be easily realized with position information, and (iv) in many applications the position itself is the information of interest.

In this paper we present a new approach to energy-saving determination of unknown coordinates with a higher precision compared to approximate positioning methods [6]. Using the "Distributed Least Squares-(DLS)"-algorithm, all calculations are split between the resource-limited sensor nodes and the high-performance base station.

This paper is structured as follows: In Section 2 we give a basic overview of the methods for localization in wireless sensor networks. In Section 3 we describe the position estimation based on relationships to known points. Next, we present in Section 4 our new DLS-algorithm to split the least squares method with the aim to minimize the load on the sensor nodes. Furthermore, the complexity of DLS is analyzed in Section 5, simulated in Section 6 and discussed in Section 7. We finally conclude the paper with Section 8.

2. RELATED WORK

Considering energy constraints in sensor networks, the group of approximate algorithms consumes less power, but estimates a position with a higher localization error (see Fig. 1). Different approximate (also called coarse-grained) localization approaches exist in the literature [7, 6, 8, 9, 10].

Exact localization of a sensor node features high precision and is based on solving a linear system of equations with coordinates of the beacons and distances to them. With at least three beacons, required in 2-dimensions, sensor nodes estimate their positions via trilateration. More beacons than required result in an over determined system of equations that must be solved with e.g. a least-squares method (multilateration). The multilateration produces accurate results, however it is complex and resource-intensive and therefore not feasible on resource-limited sensor nodes. Nevertheless, different authors deals with reducing the complexity of these methods [11, 12, 13, 14, 15].

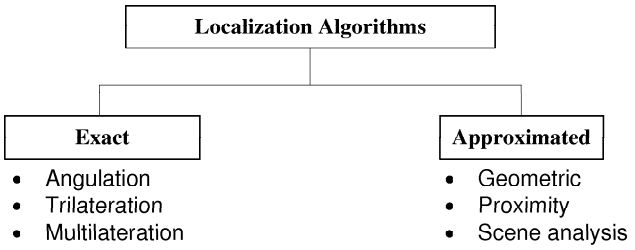


Figure 1: Classification of common localization techniques in sensor networks.

We demand exact localization methods that work on tiny sensor nodes with highly limited energy resources. To achieve this, we transfer the complex calculations such as matrix multiplication or matrix inversion to the base station. Consequently, only simple calculations have to be executed on the sensor nodes. Additionally, we reduce the communication and memory requirements through optimizations of the proposed algorithm.

3. BACKGROUND: MULTILATERATION

Estimating the position of an unknown point $P(x, y)$ requires at least three known points in two-dimensions. With m known coordinates $B(x_i, y_i)$ and its distances r_i to them we obtain:

$$(x - x_i)^2 + (y - y_i)^2 = r_i^2 \quad (i = 1, 2, \ldots, m) \tag{1}$$

This system of equations must be linearized by using the j'th equation of (1) as the linearization tool. By adding and subtracting x_j and y_j to all other equations this leads to:

$$\begin{array}{l}(x - x_j + x_j - x_i)^2 + (y - y_j + y_j - y_i)^2 = r_i^2 \\ (i = 1, 2, \ldots, j-1, j+1, \ldots, m)\end{array} \tag{2}$$

With the distance r_j (r_i), the distance between the unknown point and the j'th (i'th) beacon, and the distance d_{ij}, the distance between beacon B_i and B_j, this leads, after resolving and simplifying, to:

$$\begin{array}{l}(x - x_j)(x_i - x_j) + (y - y_j)(y_i - y_j) \\ = \frac{1}{2}\left[r_j^2 - r_i^2 + d_{ij}^2\right] = b_{ij}\end{array} \tag{3}$$

Because it is not important which equation we use as a linearization tool, $j = 1$ is sufficient. This is equal to choosing the first beacon and if $i = 2, 3, \ldots, m$ this leads to a linear system of equations with $m-1$ equations and $n = 2$ unknowns.

$$\begin{array}{rcl}(x - x_1)(x_2 - x_1) + (y - y_1)(y_2 - y_1) & = & b_{21} \\ (x - x_1)(x_3 - x_1) + (y - y_1)(y_3 - y_1) & = & b_{31} \\ & \vdots & \\ (x - x_1)(x_m - x_1) + (y - y_1)(y_m - y_1) & = & b_{m1}\end{array} \tag{4}$$

This system of equations can be written in the matrix form $A\mathbf{x} = \mathbf{b}$ with:

$$A = \begin{pmatrix} x_2 - x_1 & y_2 - y_1 \\ x_3 - x_1 & y_3 - y_1 \\ \vdots & \vdots \\ x_m - x_1 & y_m - y_1 \end{pmatrix},$$

$$\mathbf{x} = \begin{pmatrix} x - x_1 \\ y - y_1 \end{pmatrix}, \mathbf{b} = \begin{pmatrix} \frac{1}{2}\left[r_1^2 - r_2^2 + d_{21}^2\right] \\ \frac{1}{2}\left[r_1^2 - r_3^2 + d_{31}^2\right] \\ \vdots \\ \frac{1}{2}\left[r_1^2 - r_m^2 + d_{m1}^2\right] \end{pmatrix} \tag{5}$$

Now this basic form has to be solved using the linear least squares method. Due to the fact that overdetermined systems of equations with $m >> n$ have no exact solution for $A\mathbf{x} = \mathbf{b}$, we have to apply the L2-norm. This is also called the Euclidean norm, which minimizes the sum of the squares of the residuals:

$$\underset{x \in \Re^n}{Minimize} \quad ||A\mathbf{x} - \mathbf{b}||_2. \tag{6}$$

A trivial solution of the least squares problem is to reconvert after $\mathbf{x}$. In this case, the unique solution of $A\mathbf{x} \approx \mathbf{b}$ is given by:

$$||A\mathbf{x} - \mathbf{b}||_2 \rightarrow A^T A\mathbf{x} = A^T\mathbf{b}. \tag{7}$$

Solving normal equations is a good choice if the linear system has many more equations than unknowns,

i.e. $m >> n$, because after the multiplication $A^T A$ the result is only a quadratic $[n \times n]$-matrix. This simplifies the following computation and makes it easier to be implemented in software.

4. ALGORITHM DESCRIPTION

DLS builds on the mathematical formulations introduced in the background section. By using the linearization tool the matrices in Equ. (5) have two important benefits. First, all elements in the coefficient matrix A are generated by beacon positions $B_1(x, y) \dots B_m(x, y)$ only. We assume in the first instance that all sensor nodes can establish communication links between all beacons, then matrix A is the same on every sensor node. Second, vector $\mathbf{b}$ contains distances between sensor nodes and beacons $r_1 \dots r_m$, which have to be estimated on every sensor node independently. The result is that the normal equations can be split into two parts - a more complex part, the *precalculation*: $A_p = \left(A^T \cdot A\right)^{-1} \cdot A^T$ and a simple part: $A_p \cdot \mathbf{b}$, further called the *postcalculation*. Here, the precalculation is executed on one high performance node, which additionally avoids high redundancy, because normally this precalculation has to be executed on all sensor nodes separately. It is very important to emphasize that the precalculation is identical on each sensor node. Thus, it is calculated only once, conserving expensive energy resources. The simple postcalculation is then executed on each sensor node with its individual distance measurements to all beacons. This approach complies with two important design strategies for algorithms in large sensor networks - a **resource-aware** and **distributed** localization procedure. Finally, this can be achieved with less communication overhead required for other exact algorithms.

At this point we briefly describe the algorithm process. DLS is divided into three phases, which are shown in Fig. 2.

- **Phase 1: Initialization**
  - All beacons send their position $B(x, y)$ to the base station.

- **Phase 2: Complex Precalculation (central)**
  - Base station builds matrix A and vector $\mathbf{d}_p$.
  - Starting the complex precalculation of matrix A_p.

- **Phase 3: Simple Postcalculation (distributed)**
  - Base station sends matrix A_p and vector $\mathbf{d}_p$ to all sensor nodes.
  - Sensor nodes determine the distance to every beacon $r_1..r_m$.
  - Sensor nodes receive matrix A_p and vector $\mathbf{d}_p$, built vector $\mathbf{b}$ and estimate their own position $P_{est}(x, y)$ autonomously.

5. THEORETICAL ANALYSIS

5.1 Computation Complexity

At this point it is important to compare the complexity of the normal equations with the postcalculation. In order to define the complexity mathematically, we count the number of floating point operations (flops), which is a commonly used method in literature. It is described in [16] that the total complexity for the least squares method with m beacons and $n = 2$ unknowns is $15m - 5$ flops. That means, with 100 beacons and the summation of x_1 and y_1 we need 1497 floating point operations. Now it can be determined what we save on the sensor nodes without complex precalculations regarding only the remaining postcalculation. On the basis of (7), the base station precalculates $A_p = (A^T A)^{-1} A^T$ and $\mathbf{d}_p = \mathbf{d}^2$. The matrix A_p and vector $\mathbf{d}_p$ are sent to all sensor nodes. Together with the distances $\mathbf{r}$ to all beacons, which every sensor node must determine itself, the postcalculation requires $8m - 11$ flops, which leads with 100 beacons and the summation of x_1 and y_1 to 791 flops. It results that DLS saves 47.16% on the sensor nodes compared to the full calculation.

5.2 Communication Effort

Due to the fact that communication consumes most of the energy, data transfer between sensor nodes must be minimized. Particularly, sending data over long distances stresses the energy capacity of sensor nodes. Communication between base station and beacons is less critical and must be preferred if possible. Therefore, we classify communication in two phases. An uncritical phase, where all beacons send their positions to the base station. This causes no energy loss on the sensor nodes. Additionally, in a critical phase, where the base station sends precalculated information to the sensor nodes that have, in theory, to receive only. Due to errors in the transmission channel and protocols that require acknowledge packets etc., transmitting/sending is never lossless in practice. Furthermore, the base station cannot reach every sensor node in one hop, which demands multi-hopping over some nodes.

Here, we focus on a theoretical examination of the algorithm that is, for the moment, independent of protocol definitions and media access operations. Hence, every sensor node must receive the precalculated matrix A_p and vector $\mathbf{d}_p$ with $[n \cdot (m-1) + (m-1)]$ elements. This results in receiving $(3m - 3)$ elements, which are 1188 bytes with 100 beacons and floating point representation of every element.[1]

[1]On common microcontrollers, that are presently integrated

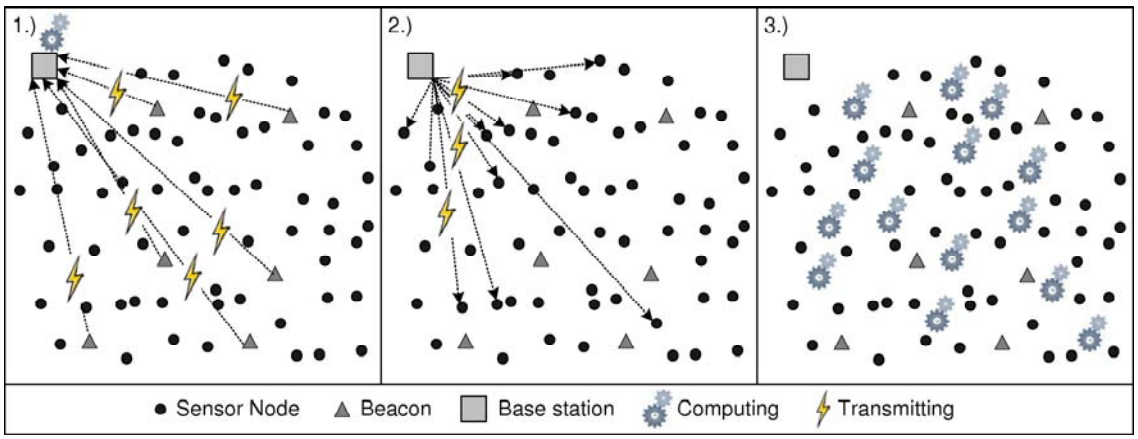


Figure 2: Procedure of the DLS-algorithm, which is divided into three phases.

5.3 Memory Requirement

The reduced calculations must be feasible on sensor nodes with a very small memory, mostly not more than a few kilobytes. In our case, the memory consuming operation is always $A_p \cdot \frac{1}{2} \cdot (r_1^2 - \mathbf{r}^2 + \mathbf{d}_p)$. In the worst case A_p and $\mathbf{r}$ plus $\mathbf{d}_p$ must be stored temporarily in memory before the execution on the sensor node can start. In more detail $[2 \cdot (m-1)] + (m-1) + (m-1) = (4m-4)$ elements must be stored.

6. SIMULATION

In the following, we show simulation results of a packet simulator. Here, DLS was compared to a direct competitor, the "Fully Distributed Multilateration" (FDM), which was among others used by Savvides et al. in [11] in a similar way. FDM is fully distributed, because every sensor node receives beacon positions directly from beacons and executes both the pre- and postcalculation completely independent.

The simulation was performed in J-Sim, a sensor network simulation framework by Tyan et al. [17], in which we added a more complex and therefore more realistic energy model. Our energy model considers the following parameter of energy consumption [2]:

- Power-mode dependent energy consumption with sleep and active mode
- Switching energy from sleep to active mode
- Distance dependent transmission of packets
- Computation complexity of the pre- and postcalculation
- Distance estimations with RSS measurements
- Position estimation via GPS on beacons

The specific energy parameters are based on the MICA2-mote, which is currently the most popular sensor node platform. Beacons have batteries with 21600 Joule, sensor nodes 6650 Joule and base stations are not limited due to access to an infrastructure. The network consists of one base station at position $P(x, y) = 1, 1$ and 315 randomly uniformly deployed nodes, including 15 beacons (with transmission range $60m$) and 300 sensor nodes (with transmission range $20m$) in a sensor field with the dimension $100m \times 100m$. We decided to use no special routing protocols, but restricted flooding to achieve a fair comparison between both algorithms. Restricted means that no more packets are forwarded by a node if all data required for computation have been received.

7. DISCUSSION

Fig. 3 shows the accumulated energy consumption of all nodes in the field. The first position packet for FDM was received at 66,82 seconds simulation time. After $16,20s$ every node received all beacon positions and started the computation process to estimate its position. In this communication phase 224 position packets were sent by the beacons and 52429 packets were received by all nodes ($95,42\%$ by sensor nodes), which is highly energy-intensive.

DLS started phase 1 at $83,32s$ by sending the first position packet. After that, phase 2 began and ended at $90,78s$. At this time the last sensor node received a packet with the precalculated matrix A_p and was therefore able to estimate its position. The whole process required the sending of 240 packets and the receiving of 2320 packets, where $12,93\%$ were received by the sensor nodes only. Summarized, DLS consumed $46,64J$

on sensor node platforms, every element is stored in floating point representation as a 4 byte number.

[2]Details of our energy model or the source code can be obtained from frank.reichenbach@uni-rostock.de.

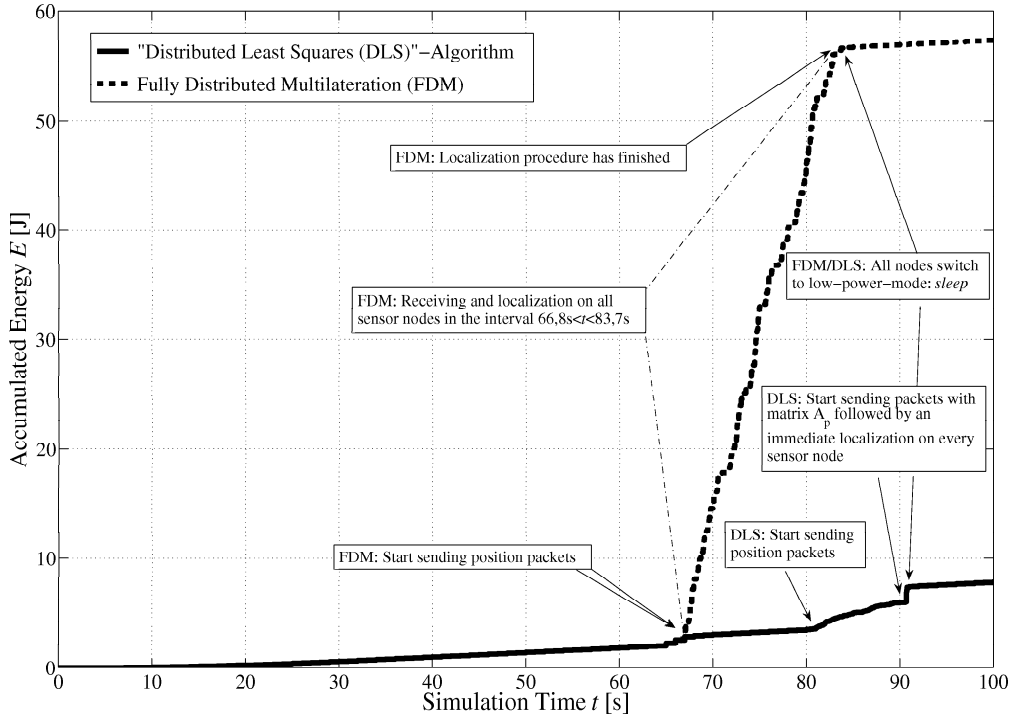


Figure 3: Accumulated energy usage of all nodes in the sensor network.

less energy than FDM, which is equal to 86,48% savings. This is mainly due to three reasons. First, every sensor node must receive only one packet. After receiving this packet the node switched its transceiver to the very low energy consuming *sleep* state. Second, compared to FDM, a very simple calculation is executed on every sensor node only. And third, instead of computing the matrix A_p on all sensor nodes, which is performed once at the base station, high redundant computations are avoided. Both algorithms estimated the positions with a localization error under 10^{-14}%, reflecting the exact distances used.

8. CONCLUSION

We presented the "Distributed Least Squares"-algorithm (DLS), which allows exact position estimation with minimal energy consumption. This algorithm is based on the least squares method, which is, for many beacons, unfeasible on resource constrained sensor nodes. However, we decreased communication overhead and computation complexity while keeping its high precision. This can be achieved by splitting the linear least squares method into a complex part, precalculated on a high-performance base station, and a very simple post-calculation on every sensor node. Thus, we eliminated redundancy, because normally every sensor node has to process the precalculation. With this approach and based on a network containing 100 beacons we achieved 47.16% savings in computation on every sensor node in comparison to the "Fully Distributed Multilateration" (FDM), as a direct competitor. Moreover, DLS needs only a few kilobytes of memory on the sensor nodes. Finally, we showed in the packet-simulator J-Sim that DLS consumes less energy than FDM - more than 86% savings in total. One next step will be to implement DLS on a sensor network platform and test it in a real world environment.

Acknowledgment

This work was supported by the German Research Foundation under grant number TI254/15-1 and BI467/17-1 (keyword: Geosens).

9. REFERENCES

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: A survey. *Computer Networks*, 38(3):393–422, 2002.

[2] R. Min, M. Bhardwaj, S. Cho, A. Sinha, E. Shih, A. Wang, and A. Chandrakasan. Low-power wireless sensor networks. In *International Conference on VLSI Design*, 2001.

[3] R. Bill, C. Cap, M. Kohfahl, and T. Mund. Indoor and outdoor positioning in mobile environments a review and some investigations on wlan positioning. *Geographic Information Sciences*, 10(2):91–98, 2004.

[4] J. Gibson. *The mobile communications handbook*. CRC Press, 1996.

[5] M. Alouini. *Global Positioning System: An Overview*. California Institute of Technology, 1996.

[6] N. Bulusu. Gps-less low cost outdoor localization for very small devices. *IEEE Personal Communications Magazine*, 7(5):28–34, 2000.

[7] H. Tian, H. Chengdu, B. M. Brian, S. A. John, and A. Tarek. Range-free localization schemes for large scale sensor networks. In *9th annual international conference on Mobile computing and networking*, 2003.

[8] J. Blumenthal, F. Reichenbach, and Dirk Timmermann. Precise positioning with a low complexity algorithm in ad hoc wireless sensor networks. *PIK - Praxis der Informationsverarbeitung und Kommunikation*, 28(2):80–85, 2005.

[9] L. Doherty, K.S.J. Pister, and L. Ghaoui. Convex position estimation in wireless sensor networks. In *20th Annual Joint Conference of the IEEE Computer and Communications*, pages 1655–1663, 2001. Anchorage, AK, USA.

[10] S. Capkun, M. Hamdi, and J.-P. Hubaux. Gps-free positioning in mobile ad hoc networks. *Cluster Computing*, 5(2):157–167, 2002.

[11] A. Savvides, C.-C. Han, and M. B. Srivastava. Dynamic fine grained localization in ad-hoc networks of sensors. In *7th ACM MobiCom (Rome, Italy,2001)*, pages 166–179, 2001.

[12] Y. Kwon, K. Mechitov, S. Sundresh, W. Kim, and G. Agha. Resilient localization for sensor networks in outdoor environments. In *25th IEEE International Conference on Distributed Computing Systems (ICDCS'05)*, pages 643–652, 2005.

[13] A. A. Ahmed, H. Shi, and Y. Shang. Sharp: A new approach to relative localization in wireless sensor networks. In *Second International Workshop on Wireless Ad Hoc Networking (WWAN) ICDCSW'05)*, pages 892–898, 2005.

[14] T. C. Karalar, S. Yamashita, M. Sheets, and J. Rabaey. An integrated, low power localization system for sensor networks. In *First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous'04)*, pages 24–30, 2004.

[15] K. Langendoen and N. Reijers. Distributed localization in wireless sensor networks: A quantitative comparison. *Computer Networks (Elsevier), special issue on Wireless Sensor Networks*, 43(4):499–518, 2003.

[16] F. Reichenbach, A. Born, D. Timmermann, and R. Bill. A distributed linear least squares method for precise localization with low complexity in wireless sensor networks. In *2nd IEEE International Conference on Distributed Computing in Sensor Systems*, pages 514–528. Springer Verlag, Juni 2006. San Francisco, USA.

[17] J-sim: A component-based, compositional simulation environment, http://www.j-sim.org/, 2007.

"Testbeds or bad tests? Konzept einer Experimentierumgebung für energieeffiziente WSNs

Mario Pink
Forschungsgruppe Dezentrale Systeme und Netzdienste
Institute für Telematik
Universität Karlsruhe
76131 Karlsruhe
mario.pink@kit.edu

ABSTRACT

Wir präsentieren TeZeuS einen Entwurf für skalierbare, flexible Testbed-Architekturen zur Unterstützung der Entwicklung von drahtlosen Sensor-Netzwerken (WSN). Der Entwurf von TeZeuS basiert auf der Analyse von Experimentnotwendigkeiten für eine zuverlässige Informationsbereitstellung in energiebewussten ubiquitären Systemen. Das Ziel des Entwurfs ist es die Unterstützung von WSN-Experimenten für ubiquitäre Systeme, sowohl für ressourcenstarke als auch ressourcenschwache Sensor-Knoten zu verbessern, Optimierungspotentiale über detailierte Energiemessungen aufzuzeigen sowie mit der Simulation von verschiedenen Energiezuständen der Sensor-Knoten, WSN-Anwendungen auf ihre Robustheit zu testen. Die Möglichkeit der Analyse des Funkmediums und die Simulation von verschiedenen Sensor-Knoten-Distanzen ermöglicht zudem eine genaue Betrachtung des Einflusses von Hindernissen auf das Kommunikations- und das Energieverhalten. Zusätzlich können mit Hilfe von kontrollierten Simulationen von Umgebungseinflüssen Experimente mehrfach unter gleichen Bedingungen reproduziert und somit bessere Optimierungsstrategien identifiziert werden. Selbstorganisationsmechanismen, der Einsatz von Hardware mit standardisierten Schnittstellen, ein drahtloses Management-Backbone-Netzwerk und Open-Source Software sollen den TeZeuS-Entwurf hoch skalierbar, kostengünstig und einfach realisierbar machen.

General Terms

Design, Experimentation, Measurement

Keywords

Wireless sensor networks, Testbeds, Energie measurement, Frequency measurement

1. EINFÜHRUNG

Drahtlose Sensor-Netzwerke oder Wireless Sensor-Networks (WSN) und deren Anwendungen besitzen ein hohes Potential, die Art der Informationsgewinnung zu revolutionieren. Sie ermöglichen die Überwachung schwer zugängiger und gefährlicher Gebiete [17][33][11] mit einer adaptierbaren Informationsqualität, sowie die Entwicklung neuer Geschäftsprozesse, z.B. Warenverfolgung in der Logistik. Aufgrund der inhärenten Flexibilität in ihrer Anwendung und der unberechenbaren Einsatzumgebung dieser Systeme verfügen drahtlose Sensor-Netzwerke über zum Teil völlig neue Systemanforderungen, die bei ihrer Entwicklung berücksichtigt werden müssen, zum Beispiel:

- *Wartungs-, Unterbrechungsfreier Betrieb* über mehrere Jahre,
- *sehr hohe Skalierbarkeit*, von einigen 10 bis mehreren 10000 Sensor-Knoten, verbunden mit
- *hoher Fehlerrobustheit und Zuverlässigkeit* sowie
- *effiziente Verwendung von Sensor-Knoten-Ressourcen*, z.B. CPU-Power,
- *minimaler Verbrauch der beschränkten Batterieenergie* zur Maximierung der Lebenszeit

Diese Anforderungen unterscheiden sich signifikant gegenüber traditionellen Verteilten-Systemen. Im Unterschied zu diesen Systemen ist in Sensor-Netzwerken die beschränkte Verfügbarkeit von Energie der primäre Faktor, der die Funktionsfähigkeit des Netzwerkes bestimmt. Dieser Umstand erfordert daher zum einen die Entwicklung neuer Algorithmen, z.B. Datenaggregationsalgorithmen, als auch die Adaption existierender Algorithmen und Kommunikationsmechanismen auf einen geringen Energieverbrauch.

Zur Entwicklung und Optimierung werden gegenwärtig überwiegend Simulationen eingesetzt. Simulationen aber besitzen den Nachteil, dass ihre zugrundeliegenden Modelle das Verhalten von Sensor-Netzwerken meist nur beschränkt widerspiegeln. So werden sehr oft unzureichende Schätzungen, z.B. für das Datenübertragungsvolumen, Fehlermuster und Topologien, in einer Simulation verwendet. Entwicklungsschritte wie Implementierung, Evaluation und Optimierung der Anwendungsperformanz sowie Fehlerrobustheit und andere nicht funktionale Eigenschaften können in Simulationen

nur schwer analysiert werden. Aufgrund dieser mangelhaften Abbildung der Realität sind Simulationen allein unzureichend. Wirklichkeitsgetreue Experimente sind daher unumgänglich. Die integrale Komplexität solcher Experimente aber erfordert entsprechende Experimentierumgebungen, die das Experimentieren in Sensor-Netzwerken vereinfachen. Diese Experimentierumgebungen (*"Testbeds"*) unterstützen alle Phasen eines Experimentes:

- *von der Spezifikation*,
- *über die Ausbringung der Anwendung auf den Sensor-Knoten*,
- *der darauffolgenden Konfiguration der Sensor-Knoten*,
- *bis hin zur Ausführung und*
- *automatisierten Überwachung des Versuchsablaufs der sich anschliessenden Analyse.*

Im folgenden beschreiben wir einen skalierbaren, flexiblen *Testbed-Entwurf für die zuverlässige Informationsbereitstellung in energiebewussten ubiquitären Systemen* (TeZeuS). Der Rest des Dokuments ist wie folgt aufgebaut: Abschnitt 1.1 begründet den Entwurf von TeZeuS, über eine Analyse der Unterstützung von Experimenten, in existierenden Testbeds. Abschnitt 2 beinhaltet eine Anforderungsanalyse für Sensor-Netzwerk-Testbeds und Abschnitt 3 beschreibt darauffolgend die Realisierung dieser Anforderungen in Form des TeZeuS-Architektur-Entwurfs.

1.1 Stand der Technik

Gegenwärtig existiert bereits eine gewisse Anzahl von WSN-Testbeds, die in der Regel die Überwachung von Experimenten, z.B. Energie-, Radiomessungen, nur in einem beschränkten Maße unterstützen. Diese Testbeds, wie z.B. Omega [2], sMote [3], Kansei [19], WASAL [16], EmStar [27][24], mobile Emulab [4][30][31] und MistLab [8], sind in der Regel primär zur Entwicklung von WSN-Anwendungen und verteilten Kommunikations- oder Systemanalyse-Mechanismen gedacht. Für die Entwicklung und Optimierung von energieeffizienten Kommunikationsmechanismen und WSN-Anwendungen sind diese daher nicht optimal geeignet.

Einige Testbeds ermöglichen zwar die Simulation von verschiedenen Energiezuständen der Sensor-Knoten, indem diese entweder mit Hilfe konstanter Energie oder einer Batterie versorgt werden bzw. die Energieversorgung vollständig abgeschaltet wird, z.B. TWIST [28]. Eine detaillierte Beobachtung des Energieverhaltens mit Hilfe von Energiemessungen ist aufgrund des fehlenden Messequipments aber nur sehr schwierig realisierbar und in der Regel nicht ihr primärer Einsatzzweck. Andere Testbeds, wie z.B. MoteLab[26][14], gewährleisten zwar eine bessere Unterstützung für Energiemessungen, beschränken diese aber in der Regel auf eine begrenzte Anzahl von Sensor-Knoten. Neuere Testbed-Entwicklungen versuchen diesen Mangel zu beheben, indem Energiemessungen für jeden Sensor-Knoten realisiert werden können. Einige von ihnen beschränken aber aufgrund der geringen Datenübertragungsrate im Manage ment-Netzwerk die Genauigkeit der Messungen, z.B. JAWS [29]. Wie Energiemessungen werden auch Radiomessungen gegenwärtig nur unzureichend von einigen wenigen Testbeds unterstützt, z.B. Kansei [19][25]. So kann, z.B. die Signalstärke, in vielen Testbeds ausschließlich unter Verwendung der Sensor-Knoten ermittelt werden, wodurch die Lebenszeit des Netzwerkes beeinträchtigt wird, z.B. JAWS. Ungeachtet der beschränkten Messmöglichkeiten wird eine Validierung der Experimente, also die Sicherstellung des korrekten Ablaufs von Experimenten, die Wiederholung von Messungen unter gleichen Bedingungen und die Möglichkeit Umwelteinflüsse zu simulieren, gegenwärtig nur unzureichend zur Verfügung gestellt. Einige Testbeds eröffnen aber die Möglichkeit die realen experimentellen Resultate mit Hilfe von Simulationen näherungsweise zu Überprüfen.

Das EmStar-Testbed [27][24] bietet diese Möglichkeit, indem es Anwendungen sowohl auf realen als auch auf simulierten Sensor-Knoten ausführen kann, ohne dafür weitere Anpassungen der Anwendung vornehmen zu müssen. Die Genauigkeit dieser Simulation ermöglicht aufgrund der modellgenerierten Sensordaten bestenfalls eine sehr grobe näherungsweise Abschätzung der korrekten Experimentausführung. Kansei [19][25] erweitert diesen Ansatz, indem es zusätzlich gespeicherte Sensordaten in der Simulation verwenden kann. Insgesamt betrachtet werden die Vorteile von realen Experimenten im Vergleich zu Simulationen gegenwärtig nur unzureichend ausgeschöpft. Diese Testbeds sind daher für die Entwicklung und Optimierung von energiebewussten ubiquitären Systemen nur sehr begrenzt einsetzbar. Wir benötigen daher ein Testbed, dass eine gezieltere Unterstützung von WSN-Experimenten gewährleistet.

2. ANFORDERUNG AN WSN TESTBEDS

Damit ein Testbed optimale Hilfestellung beim Experimentieren gewährleisten kann, müssen eine Vielzahl an Anforderungen erfüllt sein. Dabei gilt: je besser die Unterstützung des Experimentierens, um so besser die Ergebnisse der Experimente. Die Anforderungen an ein Testbed lassen sich daher in:

- *Wissenschaftliche Anforderungen*,
- *Management Anforderungen*
- *Ökonomische Anforderungen*

differenzieren.

2.1 Wissenschaftliche Anforderungen

Für ein wissenschaftliches Experiment lassen sich drei Schlüssel-Anforderungen definieren, die von einem Testbed mit geeigneten Mechanismen unterstützt werden müssen [32]:

- Reproduzierbarkeit des Experimentes
- Verständnissförderung des Experimentes
- Korrekte Ausführung des Experimentes

2.1.1 Reproduzierbarkeit von Experimenten

Die Reproduzierbarkeit ermöglicht die Wiederholung von Experimenten, unter gleichen Bedingungen, mit identischem Ablauf und bildet somit die Basis der Entwicklung und Optimierung von Sensor-Netzwerken. Reproduzierbarkeit stellt hohe Anforderungen an die Überwachung und Protokollierung des experimentellen Aufbaus und der Durchführung des Versuchs. Folgende Anforderungen sind dabei von zentraler Bedeutung:

Unterstützung der Experimentspezifikation.
Um einen mehrfachen identischen Versuchsaufbau auf dem Testbed realisieren zu können, muss eine detailierte Beschreibung des Versuchsaufbaus, z.B. Knotenanzahl, Knotentyp, des Experimentablaufs, der Laufzeit des Experimentes und der Ereignisse, die während des Experimentes simuliert werden sollen in standardisierter Art realisiert werden.

Automatisierte Testbed-Konfiguration.
Damit unvorhergesehene Zustände des Testbeds während des Versuchsaufbaus, z.B. Konfiguration von Sensor-Knoten, Parametern, Diensten und Datenbanken etc., vermieden werden können, muss die Konfiguration des Testbeds automatisiert und überwacht werden, z.B. über Zustandsautomaten.

Archivierung der Experimente.
Damit ein Experiment wiederholt und auf seine korrekte Ausführung überprüft werden kann, z.B. durch Simulation, müssen alle Experimentdaten archiviert werden. Dazu gehören neben der Experimentspezifikation alle während des Experimentes gewonnenen Informationen, einschließlich der Umwelteinflüsse und des Testbedzustands, vor und nach dem Experiment.

Rekonstruktion von Umwelteinflüssen.
Umwelteinflüsse, wie z.B. Licht- oder akustische Verhältnisse, sind in der Regel temporär und einzigartig. Ergebnisse des Experimentes können daher bei jeder Wiederholung differieren und somit eine Analyse der experimentellen Ergebnisse erschweren. Umwelteinflüsse müssen somit quantifiziert, archiviert und entsprechend rekonstruiert werden können.

2.1.2 Verständnis der Experimente

Um das Verständnis von Experimenten zu fördern und Optimierungsansätze, im Hinblick auf den Energieverbrauch, aufzuzeigen, sind überwachte Experimente unter unterschiedlichen Bedingungen, z.B. Sensor-Knoten-Ausstattung, Kommunikationsbeziehungen, Umgebungseinflüssen, Skalierbarkeit, erforderlich. Testbeds müssen daher Mechanismen zur Verfügung stellen, die solche unterschiedliche Bedingungen realisieren und sinnvolle Informationen über das Verhalten des Systems während des Experimentes ermitteln können.

Messungen.
Messungen ermöglichen die Überwachung des Experimentes und die Quantifizierung des Experimentstatus in Zahlen. Für Sensor-Netzwerke sind besonders folgende Messungen von zentraler Bedeutung:

- *Messung des Energieverbrauchs* aller Sensor-Knoten.
- *Messung des Funkverhaltens* aller Sensor-Knoten z.B. zur Analyse von Frequenz-Sprung-Verfahren, Beeinflussung der Kommunikationsreichweite durch Hindernisse (z.B. Bäume).

Neben diesen eher praktischen Messungen sollten weitere Messungen über konkrete Analysen des Sensor-Netzwerkes zur Verfügung gestellt werden. Diese Messungen lassen sich wie folgt klassifizieren:

- Vermessung des Sensor-Knoten-Verhaltens, z.B. CPU-Auslastung, Speicherauslastung.
- Vermessung des Verhaltens von Sensor-Knoten-Gruppen, z.B. Latenz, Datendurchsatz, Aggregationsverhalten.
- Vermessung der gruppenübergreifenden Kommunikation und des Verhaltens des gesamten Sensor-Netzwerkes, z.B. Latenz, Datenaufkommen, Datendurchsatz.

Heterogenität der Umgebung.
Das Verhalten von Sensor-Netzwerken unterscheidet sich entsprechend seiner Einsatzumgebung. Sensor-Netzwerke in Laborumgebungen sind in der Regel Umgebungseinflüssen, wie z.B. Wind, Sonnenlicht, Hitze, Eis, nicht in dem Maße ausgesetzt wie in der freien Natur. Testbeds müssen daher die Vielfältigkeit der unterschiedlichen Umgebungseinflüsse sowohl aus Laborumgebungen (Indoor) als auch der freien Natur (Outdoor) abbilden können.

Mobilität der Sensor-Knoten.
Sensor-Netzwerke bzw. deren Sensor-Knoten unterliegen einer unterschiedlichen Dynamik. Diese Dynamik kann zum einen Bestandteil ihrer Anwendung sein, z.B. Hochseebojen zur Erforschung von Strömungsverhältnissen, oder aber sich unvorhergesehen sporadisch ändern, z.B. herabfallen der Sensor-Knoten von Bäumen[11]. Zur Analyse solcher dynamischen Zustandsänderungen in Experimenten sollten Testbeds daher sowohl statische als auch mobile Sensor-Knoten zur Verfügung stellen und deren Dynamik zur Reproduktion quantifizierbar machen.

Vielfalt der Ausstattung.
Abhängig vom verwendeten Sensor-Knoten-Typ unterscheidet sich möglicherweise das Verhalten des Sensor-Netzwerkes bei gleicher Anwendung. Messungen auf einer homogenen Sensor-Knoten-Plattform können diese Verhaltensunterschiede nur unzureichend erfassen. Testbeds sollten daher Sensor-Knoten mit geringen Ressourcen ("reduzierte Funktion") als auch mit hohen Ressourcen ("vollständiger Funktion") und möglichst unterschiedlichen Hardware- und Software-Architekturen zur Verfügung stellen.

Vielfalt der Architekturen.
Damit komplexe Experimente und deren integrale Anwendungen sukzessive aufgebaut und verbessert werden können, um dadurch verteilte Zusammenhänge besser zu verstehen. Sollte ein Testbed die Anwendungsentwicklung mit Hilfe verschiedener Abstraktionstufen der Zielnetzwerk-Architektur vereinfachen. Man unterscheidet drei Abstraktionstufen:

- Flache Architekturen: bestehen aus homogenen Sensor-Knoten mit identischen Anwendungen und Protokollen.
- Segmentierte Architekturen: bestehen aus mehreren, über Gateways gekoppelten, flachen Architekturen.
- Schichten-, hierarchische Architekturen: bestehen aus Hierarchien von flachen Architekturen.

Vielfalt der Topologien.
Abhängig von der verwendeten Netzwerk-Topologie kann es in Folge von Sensor-Knoten-Ausfällen aufgrund der Selbstorganisationsmechanismen von Sensor-Netzwerken zu spo-

radischen Topologieänderung kommen. Der Ressourcenverbrauch erhöht sich dabei in der Regel unkalkulierbar. Die Fehleranfälligkeit und der Energieverbrauch von Sensor-Netzwerken sind somit entscheidend von der verwendeten Netzwerk-Topologie abhängig. Ein Testbed sollte daher Experimente mit verschiedenen Sensor-Netzwerk-Topologien ermöglichen.

Skalierbarkeit.
Aufgrund der hohen Anzahl von Sensor-Knoten in Sensor-Netzwerken sollte ein Testbed über eine möglichst hohe Anzahl an Sensor-Knoten verfügen. Um diese Anzahl zusätzlich zu erhöhen, sollten neben physischen auch simulierte und emulierte Sensor-Knoten für Experimente zur Verfügung stehen.

Simulation von Umwelteinflüssen.
Umwelteinflüsse und Energiezustände der Sensor-Knoten bilden eine Vielzahl variabler Parameter, die in die Entwicklung und Optimierung von Sensor-Netzwerken mit einbezogen werden müssen. Abhängig von der Art des Experimentes verringert sich dadurch das Verständnisspotential der experimentellen Resultate. Ein Testbed sollte daher eine zielgerichtete kontrollierte Simulation der Umweltbedingungen, z.B. Verringerung, Erhöhung der Lichtintensität, sowie die Simulation von verschiedenen Sensor-Knoten-Zuständen, z.B. Batterie-, Konstante-Energieversorgung, ermöglichen, um diese Variabilität temporär zu kompensieren.

Auswertung der Experiment-Daten.
Neben den Möglichkeiten das Verständniss der Zusammenhänge und Ergebnisse von Experimenten in Sensor-Netzwerken mit den erwähnten Mechanismen zu unterstützen. Müssen geeignete Werkzeuge zur automatisierten Analyse der ermittelten Experimentdaten zur Verfügung stehen. Ein Testbed sollte daher entsprechende Analyse-Werkzeuge, z.B. Benchmarks, und Schnittstellen zur Verfügung stellen, um zusätzliche Werkzeuge, z.B. von Simulatoren, verwenden zu können.

2.1.3 Korrekte Ausführung von Experimenten

Das beste Verständnis eines Experimentes ist unzureichend, wenn das Experiment inkorrekt realisiert wurde. Um falsche Schlussfolgerungen aus Experimenten zu vermeiden, muss daher von jedem Testbed die korrekte Ausführung von Experimenten sichergestellt werden.

Diagnose des Testbed-Equipments.
Das Testbed-Equipment sollte entweder im Vorfeld von Experimenten oder in periodischen Zyklen unter kontrollierten Bedingungen auf seine korrekte Funktionsweise überprüft werden. Das Testbed sollte folgende Diagnosetests zur Verfügung stellen:

- Sensor-Knoten-Tests, z.B. Sensor-Test, CPU-Test, Memory-Test, Batterie-Test
- Sensor-Netzwerk Kommunikationstests, z.B. Erreichbarkeit von Sensor-Knoten etc.
- Management-Netzwerk Kommunikations-Tests
- Messinstrumente-Tests

Validierung durch Simulation.
Zur Validierung des korrekten Experimentablaufs sollte das gesamte Experiment ebenfalls in einer parallelen Simulation ausgeführt werden. Die daraus resultierenden Simulationswerte ermöglichen somit eine wahrscheinlichkeitsbehaftete Validierung der Experimentergebnisse. Ein Testbed sollte daher eine Experimentspezifikation verwenden die ebenfalls von einem Simulator interpretiert werden kann und die parallele Simulation unterstützen.

2.2 Ökonomische Anforderungen

Ökonomische Aspekte die beim Entwurf und der Entwicklung eines Testbeds primär beachtet werden müssen, sind vor allem die finanziellen Kosten.

2.2.1 Systemkosten

Sensor-Knoten variieren von der Größe (Formfaktor) einer Schuhkiste bis zur mikrokopischen Größe eines Partikels. Dementsprechend variieren auch die Kosten von einigen bis zu mehreren hundert Euro. Die hohe Skalierbarkeit und nicht die Ressourcenstärke der Sensor-Knoten macht Sensor-Netzwerke einzigartig. Es sollte daher eine möglichst hohe Anzahl an Sensor-Knoten angestrebt werden.

2.2.2 Wartungskosten

Insbesondere bei sehr großen Testbeds, kann der Wartungsaufwand beträchtlich sein, z.B. zur Auswechslung von Batterien . Die Wartung des gesamten Testbeds sollte daher mit geeigneten Mechanismen unterstützt werden, die eine Validierung der Funktionsfähigkeit des Testbed-Equipment ermöglichen und einen regelmäßigen Austausch der Sensor-Knoten-Batterien vermeiden.

3. TEZEUS

Der Testbed-Entwurf TeZeuS basiert auf einer hybriden Testbed-Architektur, die sich aus verschiedenen Testbed-Abschnitten zusammensetzt. TeZeuS unterscheidet einen statischen, einen portablen und einen Testbed-Abschnitt mit mobilen Sensor-Knoten. Jeder Testbed-Abschnitt ist für einen bestimmten Einsatzzweck sowie Einsatzort optimiert und kann aus mehreren Sensor-Knoten-Instanzen bestehen. Diese Sensor-Knoten-Instanzen werden mit Hilfe von emulierten, simulierten oder physischen Sensor-Knoten realisiert. Ein auf dem TeZeuS-Entwurf basierendes Testbed kann somit für die individuellen Nutzeranforderungen maßgeschneidert werden.
Wie Abbildung 1 zeigt, verwendet TeZeuS drei Softwareeinheiten: Emulations-, Simulations- und Managementeinheiten. Um die Komplexität der TeZeuS-Testbeds zu vereinfachen, können sowohl Emulations- als auch Simulationseinheiten mehrfach in einem Testbed-Abschnitt vorhanden sein.

- *Emulierte Sensor-Knoten* imitieren das Verhalten eines Sensor-Knoten in einem Sensor-Netzwerk. Sie verfügen über einen von der Netzwerk bis zur Anwendungsschicht anwendungsspezifisch implementierten Protokollstack. Ausschließlich die physikalische und die Netzwerkschicht werden simuliert oder in hybriden Emulationen auf einem realen Sensor-Knoten ausgeführt. TeZeuS verwendet sowohl simulierte als auch hybride Emulationen.

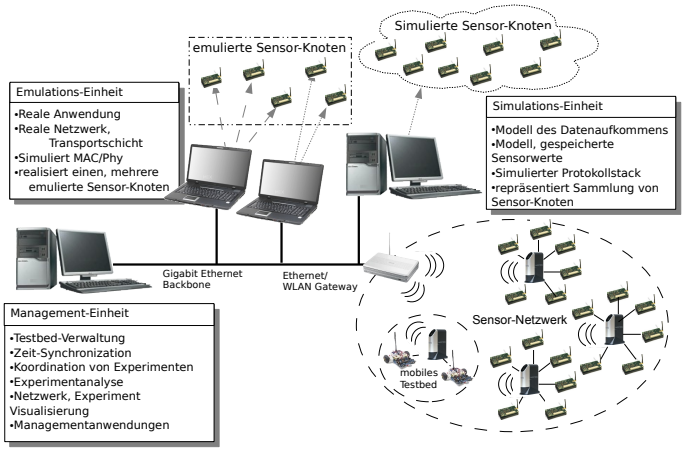


Abbildung 1: TeZeuS-Architektur

- *Simulierte Sensor-Knoten* modellieren den gesamten Protokollstack eines Sensor-Knoten. TeZeuS verwendet ebenfalls simulierte Sensor-Knoten und unterscheidet zwei verschiedene Arten der Sensordatenerzeugung: die Erzeugung von Sensordaten über Modelle und die Verwendung von gespeicherten Sensordaten.

- Die *Testbed-Abschnitt Management-Einheit* verwaltet den jeweiligen Testbed-Abschnitt und die darin enthaltenen emulierten, simulierten als auch physischen Sensor-Knoten. Sie steuert die Ausführung der Experimente und überwacht das Verhalten des Sensor-Netzwerk im Testbed-Abschnitt. Jede Testbed-Abschnitt Managementeinheit wird zudem über die hierarchische Managementtopologie in TeZeuS von einer globalen Managementeinheit verwaltet.

- Die *globale Managementeinheit* koordiniert den Experimentablauf im gesamten Testbed und stellt grafische Nutzerschnittstellen für die Visualisierung der Netzwerkdynamik sowie Veränderung von Parametern zur Verfügung. Sowohl globale als auch Testbed-Abschnitt Managementeinheiten werden in der Regel auf einem System ausgeführt. Ausschließlich bei der Verwendung des portablen Testbed-Abschnitts besteht die Möglichkeit, dessen Managementeinheit auf einem anderen System auszuführen.

Die Verwendung einer hybriden Testbed-Architektur besitzt viele Vorteile.

- Erstens, die Untergliederung der Testbed-Architektur in mehrere äquivalente Teilbereiche ermöglicht die nebenläufige Ausführung von Experimenten und die Erfassung von vielfältigen Umwelteinflüssen in Laborumgebungen sowie unterschiedlichen natürlichen Regionen.

- Zweitens, die Verwendung von emulierten, simulierten und physischen Sensor-Knoten verbessert die Skalierbarkeit des gesamten Testbeds.

- Drittens, die Verwendung von archivierten und modellbasierten Sensordaten ermöglicht realitätsgetreue Simulationen und Emulationen.

3.1 Die TeZeuS-Architektur

Die TeZeuS-Architektur basiert auf einem leicht modifizierten Deployment-Support-Network (DSN) [23] [21] und besteht aus zwei verschiedenen Arten von Knoten: DSN-Knoten und Sensor-Knoten. Abbildung 2 beschreibt die Hardware-Architektur von TeZeuS. Jeder DSN-Knoten verwal-

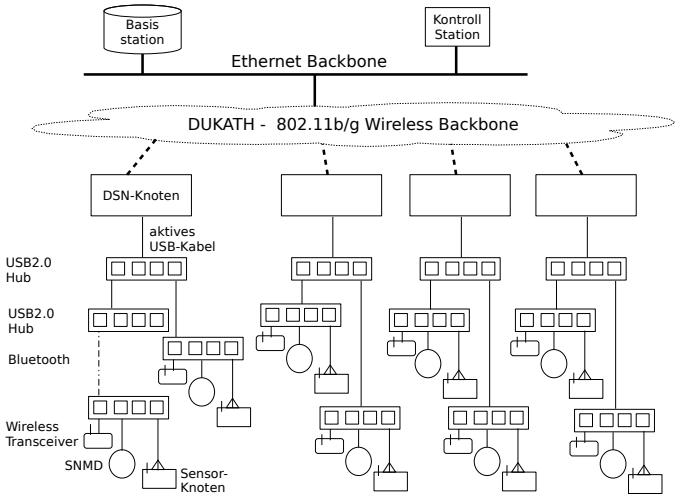


Abbildung 2: Hardware-Architektur des TeZeuS-Testbeds

tet eine bestimmte Anzahl von Sensor-Knoten, in Form eines Clusters und kommuniziert drahtlos über ein Funknetzwerk mit einer Basisstation. Diese clusterorientierte, drahtlose Testbedstruktur ermöglicht den einfachen Aufbau von stationären und portablen Testbed-Einheiten. DSN-Knoten und deren assoziierte Sensor-Knoten können dadurch einfach aus der stationären Testbed-Einheit entnommen und als portable Testbed-Einheit verwendet werden.

3.2 Sensor-Knoten

Um den vielfältigen Experimenten in WSN gerecht zu werden und der Entwicklung von Sensor-Netzwerken zu begegnen, verwendet TeZeuS sowohl ressourcenschwache als auch ressourcenstarke Sensor-Knoten. Als Vertreter für die ressourcenschwachen Sensor-Knoten ist besonders der MICAz Sensor-Knoten für einen Einsatz im TeZeuS prädestiniert, da er gegenwärtig als informeller Standard für ressourcenschwache Sensor-Knoten angesehen werden kann und den Standard IEEE 802.15.4 unterstützt. Die Architektur des MICAz ist mit der Architektur vieler Sensor-Knoten weitestgehend identisch, z.B. BTnode [20],Tmote-Sky [15], Iris [5], MICAz [7], TelosB [13] etc., und unterscheidet sich nur geringfügig in der Sensor- und CPU-Ausstattung. Im Gegensatz zu den ressourcenschwachen Sensor-Knoten existieren gegenwärtig nur einige wenige ressourcenstarke Sensor-Knoten ,z.B. Sun-SPOT, die für einen Einsatz im TeZeuS geeignet sind. Der TeZeuS verwendet daher Sun-SPOTS als Vertreter für ressourcenstarke Sensor-Knoten.

3.2.1 MICAz

Der MICAz [7] Sensor-Knoten besteht aus einem *Prozessor*- und einem *Sensorboard*, die über eine *Batterie* mit Energie versorgt werden. Das Prozessorboard besteht aus einem Atmel Atmega 128 8-Bit Microcontroller mit 64kByte internem RAM, 4 KByte EEPROM und 64 KByte Flash-Speicher sowie 64 KByte externem Flash-Speicher. Insgesamt steht somit ein 128KByte und 512 KByte seriell beschreibbarer Flash-Speicher zur Verfügung. Das Prozessorboard bietet zusätzlich zwei Universal-Asynchronous-Receiver-Transmitter (UART), einen 10-Bit Analog/Digital Converter mit acht Kanälen, mehrere Digitale Ein-,Ausgänge, Inter-Integrated-Circuit- (I^2C) und Serial-Peripheral-Interface (SPI)-Busanschlüsse und 3 LEDs. Die Funkschnittstelle besteht aus einem IEEE 802.15.4 kompatiblen Transceiver, der eine Kommunikation mit maximal 250 kBps im ISM Band auf 2.4 GHz ermöglicht. Die Reichweite beträgt in Gebäuden ca. 20 Meter und auf freiem Feld ca. 100 Meter. Jeder MICAz im TeZeuS ist anstatt seinen AA-Batterien mit einem Lithium-Ionen (LiION) Akku von 750mAh Kapazität ausgestattet. Bei einer minimalen Leistungsaufnahme (CPU-Sleep-Modus) beträgt die theoretische Betriebsdauer somit.

$$\frac{750mAh}{(24h * 15uA)} = 2083Tage(\approx 6Jahre) \quad (1)$$

Zusätzlich ist jeder MICAz mit einer Signalleitung zur Aktivierung von Energiemessungen und einem modifizierten USB-Gatewayboard [6] ausgestattet. Dieses USB-Gatewayboard ermöglicht somit auch bei Batteriebetrieb die Überwachung des MICAz über den USB-Bus. Dieser Sensor-Knoten kann somit sehr einfach im TeZeuS integriert werden. Als Betriebssystem wird das frei verfügbare TinyOS verwendet.

3.2.2 Sun-Spot

Sun-SPOTs (Small Programmable Object Technology) [12] bestehen wie MICAz aus drei Hauptkomponenten. Einem Prozessor-Board mit Funkschnittstelle, einem austauschbaren Sensor-Board und einem LiION Akku mit 750 mAh Kapazität. Das Prozessor-Board besteht aus einem 180 MHz getakteten 32-Bit ARM920T Prozessor, 512 kByte RAM und 4MByte Flash-Speicher sowie einem IEEE 802.15.4 kompatiblen Radiochip mit integrierter Antenne, der eine maximale Datenübertragungsrate von 250 kbps über eine maximale Reichweite von bis zu 80 Meter ermöglicht. Das Sensorboard besteht aus mehreren Dreifarb-LEDs, einem 3-Achsen Beschleunigungssensor (2G oder 6G-Skala), einem Temperatur- und Lichtsensor, sechs analogen Eingängen, fünf beliebig verwendbaren Ein-, Ausgängen sowie vier Ausgangspins für Hochstrom. Zur Interaktion mit einem Host-System ist das Prozessorboard zusätzlich mit einer USB-Schnittstelle ausgestattet. Der integrierte LiIon-Akku ermöglicht bei einer minimalen Leistungsaufnahme einen theoretischen Betrieb von

$$\frac{750mAh}{(24h * 48uA)} = 651Tage(\approx 2Jahre). \quad (2)$$

Als Betriebsystem wird eine spezielle Java Virtual Machine (Squawk) verwendet. Diese VM ermöglicht es Anwendungen für Sensor-Netzwerke in der Programmiersprache Java (Spezifikation CLDC 1.1) zu entwickeln.

3.3 Mobile Sensor-Knoten

Mobile Sensor-Knoten im TeZeuS werden über frei programmierbare ASURO-Roboter [1] realisiert, die mit einem MICAz oder Sun-SPOT Sensor-Knoten ausgestattet sind. Der ASURO-Roboter ist ein multisensorieller Roboter bestehend aus einem Atmel Microcontroller und zwei separat steuerbaren Antriebsrädern. Zusätzlich stehen sechs Kollisionstaster, eine optische Linienverfolgungseinheit, zwei Odometer und einige Anzeigeelemente zur Verfügung. Für die Integration des Roboter in das Testbed und zur Gewährleistung der Kommunikation des Roboter mit dem Testbed ist dieser mit einem Infrarot-Bluetooth-Transceiver ausgestattet. Der Roboter kann somit über Bluetooth gesteuert und bei Bedarf sehr einfach durch einen anderen, ebenfalls über eine Infrarotschnittstelle steuerbaren Roboter ersetzt werden. Der Roboter übermittelt seine Position über einen zusätzlichen GPS-Empfänger an seinen DNS-Knoten und ist über das Testbed mit einer konstanten Energieversorgung verbunden. Diese Energieversorgung besitzt die Form eines Oberleitungssystems, ähnlich dem von Autoscootern, und ermöglicht dadurch die freie Bewegung des Roboter im mobilen Testbed-Abschnitt.

3.4 Sensor Node Management Device

Die optimale Analyse des Energieverhaltens der Sensor-Knoten und somit des Sensor-Netzwerkes erfordert die Vermessung des Energieverbrauches jedes einzelnen Sensor-Knoten. Energiemessgeräte mit der notwendigen Genauigkeit und der Möglichkeit, Spannung und Stromstärke simultan zu vermessen sind gegenwärtig mit sehr hohen Kosten verbunden (ca. 500 Eur). Diese Energiemessgeräte ermöglichen zwar die simultane Vermessung von mehreren Energiequellen und könnten somit theoretisch für die Vermessung mehrerer Sensor-Knoten verwendet werden. Die dafür benötigten Kabel und deren integraler Kabelwiderstand verursachen aber fehlerhafte Messungen. TeZeuS verwendet daher ein speziell für Testbeds enworfenes Sensor-Knoten-Management-Gerät (SNMD), das hoch genaue Energiemessungen für ein Testbed zur Verfügung stellt und ca.80 Euro kostet. Die Funktionen des SNMD lassen sich wie folgt klassifizieren:

- Mehrfach redundante Energiemessungen mit einer Genauigkeit von 10-, 15- und 16-Bit/5Volt
- Simultane Messung von Spannung und Stromstärke
- Realisierung verschiedener Energieversorgungzustände des Sensor-Knoten (Ausfall, Konstante-, Akkuenergie)
- Automatisiertes Aufladen der Sensor-Knoten-Akkus
- kontrolliertes entladen des Akku zur Rekonstruktion von Akkuzuständen
- Diagnose, Funktionsüberprüfung des Sensor-Equipments
- Simulation der Umwelt für Experiment-Wiederholungen

Mit Hilfe dieses Gerätes ist es möglich, das fehleranfällige Verhalten von Sensor-Knoten zu simulieren und reale Sensor-Knotenausfälle durch Experiment-Migration zu kompensieren. Die mehrfach redundante Möglichkeit der Energiemessung gewährleistet zudem die Korrektheit der Energiemessungen. Notwendige Wartungen des Testbeds werden

auf ein Minimum reduziert und erfordern nunmehr ausschließlich einen manuellen Eingriff bei defekten Sensor-Knoten oder Akkus.

3.5 2.4 GHz Wireless Transceiver

Sensor-Netzwerke besitzen im Gegensatz zu kabelgebundenen Netzwerken eine ungeplante Struktur. Die Sendereichweiten der einzelnen Sensor-Knoten können daher beim Entwurf dieser Systeme nicht exakt definiert werden. Abhängig von der Distanz zwischen den Sensor-Knoten steigt bzw. sinkt der für die Kommunikation benötigte Energieverbrauch. Die Kommunikation hat somit einen direkten Einfluss auf das Energieverhalten und somit die Lebenszeit des gesamten Sensor-Netzwerkes. Testbeds aber besitzen eine vordefinierte Struktur und verfügen somit über das inheränte Problem, dass sie diese Realität nicht genau reflektieren können. Jeder Sensor-Knoten im Testbed besitzt eine bestimmte nicht veränderbare Position mit exakt ermittelbarem Abstand zu den übrigen Sensor-Knoten. Der Energieaufwand für die Kommunikation ist somit in der Regel konstant. Der TeZeuS verwendet daher einen speziellen 2.4 GHz Wireless Transceiver[18], der für den TeZeuS entsprechend adaptiert wird und die Möglichkeit eröffnet verschiedene Distanzen zwischen den Sensor-Knoten zu simulieren sowie die vom Sensor-Knoten abgestrahlte, aufgenommene Energie zu ermitteln.

3.6 Sensor-Knoten und USB-Verkabelung

Jeder "vollausgestattete" Sensor-Knoten verfügt über einen USB-Hub inklusive eines Sensor-Knoten (MICAz, Sun-SPOT), eines SNMD und eines 2.4 GHz Wireless Transceiver. Dieser USB-Hub ist bei statischen Sensor-Knoten über aktive USB-Kabel und bei mobilen Sensor-Knoten über Bluetooth mit einem DSN-Knoten verbunden (siehe Abbildung 2). Die Verwendung von aktiven USB-Kabeln besitzt den inheränten Vorteil, dass die Distanz zwischen Sensor-Knoten und DSN-Knoten nicht wie bei passiven USB-Kabeln auf eine maximale Länge von 5 Meter beschränkt ist. Der Abstand zwischen Sensor-Knoten und DSN-Knoten kann somit auf mehr als 5 Meter vergrößert werden. Aktive USB-Kabel werden in TeZeuS über USB-Hubs realisiert. Neben der Kommunikationsmöglichkeit erhält jeder Sensor-Knoten über seinen DSN-Knoten und seine USB-Bus Geräte-ID einen eindeutigen Identifikator innerhalb von TeZeuS und kann somit direkt adressiert werden.

3.7 Der DSN-Knoten

Jeder DSN-Knoten des Management-Backbone wird über einen Linksys Network Storage Link für USB 2.0 (NSLU2) [9] realisiert, der mit einer konstanten Energieversorgung, einer USB-WLAN Karte (802.11 b/g) sowie 1 GB USB-Speicher ausgestattet ist. Die NSLU2 basiert auf einem 266 MHz getakteten Intel 32-bit ARM-Prozessor mit 32 MB RAM, 8 MByte Flash-Speicher und realisiert mit Hilfe des freien Betriebssystems Linux Debian (Etch Release) die Form eines drahtlosen Mikroserver. Neben einem Ethernet RJ-45 Anschluss verfügt dieses Gerät über zwei USB 2.0 Anschlüsse, über die maximal 127 USB-Geräte angeschlossen werden können. Jeder DSN-Knoten kann somit 125 zusätzliche USB-Geräte oder 23 vollausgestattete Sensor-Knoten über USB-Hubs verwalten. Zusätzlich wurde dieses Gerät mit einer serielle Schnittstelle [10] (RS232) ausgestattet, um dessen Verwaltung über eine serielle Konsole zu vereinfachen.

3.8 Das Testbed Management-Backbone

Das TeZeuS Testbed-Management-Backbone besteht aus zwei unterschiedlichen Teilnetzwerken, einem drahtlosen (WLAN) und einem kabelgebundenen Ethernet-Netzwerk. Das IEEE 802.11b/g konforme drahtlose Netzwerk, z.B. DUKATH-Netz, verbindet die DSN-Knoten über verschiedene WLAN/Ethernet Gateways, z.B. Access-Points, WLAN-Router, mit dem kabelgebundenen Netzwerk. Jedem DSN-Knoten wird dabei über einen DHCP-Server seine IP-Adresse zugewiesen, mit der er über das drahtgebundene Ethernet der Basisstation kommuniziert. Neben der Basisstation und den DSN-Knoten werden zusätzlich verschiedene Simulations- und Emulationssysteme sowie eine Kontrolleinheit verwendet.

3.9 Basisstation

Um Experimente spezifizieren, analysieren zu können und deren Ausführung sowohl zeitlich als auch räumlich koordinieren zu können, verwendet TeZeuS eine Basis- oder Managementstation. Diese Einheit bildet das Herz des Testbeds und kontrolliert die Ausführung der Experimente sowie die Funktion des Testbeds. Das Kernstück der Basisstation bildet der DSN-Server des JAWS-Testbeds[22], der für den TeZeuS mit entsprechenden Anpassungen versehen wird. Dieser DSN-Server führt die Testbed-Datenbank aus und ermöglicht eine persistente Speicherung der Experimentdaten, die von den DSN-Knoten an die Basisstation übertragen werden. Diese Experimentdaten können daraufhin von verschiedenen Managementanwendungen über eine XML-RPC Schnittstelle verwendet werden. Hardware und Software-Ressourcen der Basisstation sind dementsprechend dimensioniert, um die Hochverfügbarkeit des Testbeds zu garantieren. Als Betriebsystem wird daher ebenfalls Linux, eine Standard Ubuntu Installation, verwendet.

3.10 Kontrolleinheit

Um eine Überlastung der Basisstation durch Ausführung von Management-Anwendungen zu vermeiden, ist der TeZeuS mit einer zusätzlichen Kontrolleinheit ausgestattet. Diese Kontrolleinheit erhält über die XML-RPC Schnittstelle der Basisstation direkten Zugriff auf deren Dienste und die Datenbasis der Experimente, um damit verschiedene Management-Anwendungen auszuführen und bei Bedarf die Basisstation entsprechend zu steuern.

3.11 Systeme für Simulation, Emulation

Um die Skalierbarkeit im TeZeuS zu erhöhen, können neben physischen Sensor-Knoten zusätzlich simulierte und emulierte Sensor-Knoten verwendet werden. Der TeZeuS verfügt dafür über entsprechend ausgestattete IT-Systeme, die sowohl simulierte als auch emulierte Sensor-Knoten ausführen können. Diese IT-Systeme sind nicht primär für Simulationen, Emulationen von Sensor-Knoten im TeZeuS gedacht, sondern können bei Bedarf anderweitig verwendet werden. Der TeZeuS verwendet dafür spezielle virtuelle DSN-Knoten, die dynamisch entsprechend den verfügbaren Ressourcen die Anzahl der simulierten, emulierten Sensor-Knoten variieren und ausführen können. Wie die Basisstation verwenden auch die Simulations- und Emulations IT-Systeme eine Standard Ubuntu Linux Installation.

4. ZUKÜNFTIGE ENTWICKLUNGEN

Die Entwicklung des TeZeuS befindet sich gegenwärtig in der prototypischen Phase zur Erprobung des vorgestellten Testbed-Ansatze. Zukünftige Arbeiten im TeZeuS Kontext befassen sich mit dem Aufbau des statischen und des portablen Testbed-Abschnitts sowie der Realisierung des Testbed-Managements mit Hilfe des DSN-Server.

5. REFERENZEN

[1] Asuro roboter website. `http://www.asurowiki.de/`.

[2] Berkeley omega testbed website. `http://omega.cs.berkeley.edu/`.

[3] Berkeley smote testbed website. `http://www.millennium.berkeley.edu/sensornets/`.

[4] Emulab - mobile wireless networking website. `http://www.emulab.net/tutorial/mobilewireless.php3`.

[5] Iris sensor-node website. `http://www.xbow.com/Products/productdetails.aspx?sid=264`.

[6] Mib520-usb gateway website. `http://www.xbow.com/Products/productdetails.aspx?sid=227`.

[7] Micaz website. `http://www.xbow.com/Products/productdetails.aspx?sid=156`.

[8] Mit mistlab testbed website. `http://mistlab.csail.mit.edu/`.

[9] Network storage link unit for usb 2.0 website. `http://www.linksys.com/`.

[10] Serial interface for nslu2 website. `http://www.nslu2-linux.org/wiki/HowTo/AddASerialPort`.

[11] Sonoma redwood forrest. `http://www.berkeley.edu/news/media/releases/2003/07/28_redwood.shtml`.

[12] Sun-spot sensor-node website. `http://www.sunspotworld.com`.

[13] The telows b sensor-node website. `http://www.xbow.com/Products/productdetails.aspx?sid=252`.

[14] the motelab testbed website. `http://motelab.eecs.harvard.edu/`.

[15] Tmote-sky website. `http://www.moteiv.com/`.

[16] The wireless ad-hoc sensor and actuator nets lab website. `http://wasal.epfl.ch/`.

[17] Environmental studies with the sensor web: Principles and practice. pages 103–117. Sensors, vol. 5, 2005.

[18] S. Armitage. Low-cost 2.4-ghz spectrum analyzer, April 2006.

[19] A. Arora, E. Ertin, R. Ramnath, M. Nesterenko, and W. Leal. Kansei: A high-fidelity sensing testbed. volume 10, pages 35–47, Piscataway, NJ, USA, 2006. IEEE Educational Activities Department.

[20] J. Beutel. Fast-prototyping using the btnode platform. `http://btnode.ethz.ch`, March 2006.

[21] J. Beutel, M. Dyer, L. Meier, M. Ringwald, and L. Thiele. Next-generation deployment support for sensor networks. In *TIK-Report No: 207*, Computer Engineering and Networks Lab Swiss Federal Institute of Technology (ETH) Zurich 8092 Zurich, Switzerland, Jul.

[22] J. Beutel, C. Zimmermann, and L. Thiele. Online sensor-network monitoring. Jul 2005.

[23] 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 *Proceedings of the 4th European Conference on Wireless Sensor Networks*, pages 195–211. Springer, Berlin, Jan. 2007.

[24] J. Elson, S. Bien, N. Busek, V. Bychkovskiy, A. Cerpa, D. Ganesan, L. Girod, B. Greenstein, T. Schoellhammer, T. Stathopoulos, and D. Estrin. EmStar: An Environment for Developing Wireless Embedded Systems Software. Technical Report CENS Technical Report 0009, Center for Embedded Networked Sensing, University of California, Los Angeles, March 2003.

[25] E. Ertin, A. Arora, R. Ramnath, V. Naik, S. Bapat, V. Kulathumani, M. Sridharan, H. Zhang, H. Cao, and M. Nesterenko. Kansei: a testbed for sensing at scale. In *IPSN '06: Proceedings of the fifth international conference on Information processing in sensor networks*, pages 399–406, New York, NY, USA, 2006. ACM Press.

[26] W. A. Geoff, P. Swieskowski, and M. Welsh. Motelab: A wireless sensor network testbed. Technical report, Harvard University, April 2005.

[27] L. Girod, J. Elson, A. Cerpa, T. Stathopoulos, N. Ramanathan, and D. Estrin. Emstar: a software environment for developing and deploying wireless sensor networks. In *Proceedings of the 2004 USENIX Technical Conference*, Boston, MA, 2004. USENIX.

[28] H. Handziski, A. Köpke, A. Willig, and A. Wolisz. Twist: A scalable and reconfigurable wireless sensor network testbed for indoor deployments. Technical Report TKN-05-008, Telecommunication Networks Group, Technische Universität Berlin, Nov. 2005.

[29] D. Hobi, L. Winterhalter, and L. Thiele. Large-scale bluetooth sensor-network demonstrator. october 2005.

[30] D. Johnson, T. Stack, R. Fish, D. Flickinger, R. Ricci, and J. Lepreau. Truemobile: A mobile robotic wireless and sensor network testbed. In *Flux Technical Note FTN-2005-02, University of Utah*, April 2005.

[31] D. Johnson, T. Stack, R. Fish, D. Flickinger, L. Stoller, R. Ricci, and J. Lepreau. Mobile emulab: A robotic wireless and sensor network testbed. In *Proceedings of INFOCOM*. IEEE, Apr. 2006.

[32] W. Kiess, S. Zalewski, A. Tarp, and M. Mauve. Thoughts on mobile ad-hoc network testbeds. In *Proceedings of IEEE ICPS Workshop on Multi-hop Ad hoc Networks: from theory to reality*, pages 93–100, Santorini, Greece, Jul 2005.

[33] N. Xu, S. Rangwala, K. K. Chintalapudi, D. Ganesan, A. Broad, R. Govindan, and E. Estrin. A wireless sensor network for structural monitoring. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 13–24, New York, NY, USA, 2004. ACM Press.