TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY
Julkaisu 740 • Publication 740

Mauri Kuorilehto

System Level Design Issues in Low-Power Wireless Sensor Networks

Tampere 2008

Tampereen teknillinen yliopisto. Julkaisu 740
Tampere University of Technology. Publication 740

Mauri Kuorilehto

System Level Design Issues in Low-Power Wireless Sensor Networks

Thesis for the degree of Doctor of Technology to be presented with due permission for public examination and criticism in Tietotalo Building, Auditorium TB104, at Tampere University of Technology, on the 6th of June 2008, at 12 noon.

Tampereen teknillinen yliopisto - Tampere University of Technology
Tampere 2008

ISBN 978-952-15-1985-7 (printed)
ISBN 978-952-15-2006-8 (PDF)
ISSN 1459-2045

Mauri Kuorilehto

System Level Design Issues in Low-Power Wireless Sensor Networks

Final manuscript
April 29, 2008

Contact Information:

Mauri Kuorilehto

Mail: Tampere University of Technology
Institute of Digital and Computer Systems
P.O. Box 553 (Korkeakoulunkatu 1)
FI-33101 Tampere
Finland

Tel.: +358–50–486 9915 (mobile)
Fax: +358–3–3115 4561
E-mail: mauri.kuorilehto@nokia.com

ABSTRACT

Wireless Sensor Networks (WSN) are an emerging technology that is considered to have a high potential for realizing the vision of ambient intelligence. Tiny WSN nodes are unobtrusively embedded to environment for performing sensing, data processing, and actuating tasks. Capabilities of a single node are limited, but the feasibility of WSNs lies on the collaboration of nodes. WSNs are envisioned for a wide variety of applications ranging from home automation to military surveillance. Supporting the diversity of applications within the resource constraints is commonly considered to necessitate application-specific tailoring of nodes, communication protocols, and application algorithms.

This Thesis presents a design methodology for facilitating the development of application-specific WSNs from an abstract design phase to the prototype implementation. The functionality of a WSN is first designed in Wireless Sensor Network Simulator (WISENES) design environment with abstract deployment models. The novel features in WISENES are the graphical design of the models combined with the full-scale design time simulations for accurate evaluation of WSN performance, and the back-annotation of the performance results from prototypes for further improving simulator accuracy.

A runtime environment in the methodology preserves the design time abstractions during a prototype implementation on node platforms. The runtime environment for resource limited nodes is realized by a preemptive multithreading Operating System (OS), SensorOS, and a middleware that supports runtime distribution of application processing. A distinctive feature in SensorOS is its accurate time concept that enables the implementation of tightly synchronized WSN protocols and time sensitive applications. A lightweight allocation algorithm of WSN node middleware assigns application tasks and network maintenance roles with an objective to maximize network lifetime.

The application-specific tailoring of WSNs and the feasibility of the presented design methodology are illustrated by two case studies. An energy efficient Tampere University of Technology WSN (TUTWSN) targeted to low data rate monitoring ap-

plications is adapted to relay TCP/IP data through WSN and configured for delay sensitive operation in an indoor surveillance application. The results show that presented methods and tools facilitate and hasten the design, configuration, and implementation of WSN protocols and algorithms for different applications.

PREFACE

The research work for this Thesis was carried out in the Institute of Digital and Computer Systems at Tampere University of Technology during the years 2001–2007.

I would like to express my gratitude to my supervisor Prof. *Marko Hännikäinen* for his guidance and motivation during the research. I am also grateful to Prof. *Timo D. Hämäläinen* for his guidance and for the opportunity to carry out the research in DACI research group. In addition, thanks to both for giving me challenging and rewarding responsibilities in DACI research group. Sincere acknowledgements go also to Prof. Shuvra S. Bhattacharyya and Prof. Jari Porras for reviewing and giving valuable comments on the manuscript of the Thesis.

Many thanks to my colleagues in DACI research group for the inspiring working atmosphere and enjoyable discussions mostly in the coffee room. Especially, Dr. Tero Kangas, Dr. Panu Hämäläinen, Dr. Jari Heikkinen, Mr. Timo Alho, M.Sc, and Mr. Erno Salminen, M.Sc, deserve thanks for their insights and support on research related and not so much research related issues. Also thanks to Mr. Mikko Kohvakka, M.Sc, Mr. Jukka Suhonen, M.Sc, and to the other members of TUTWSN team for their valuable work in the development of technology that made this Thesis possible.

This Thesis was financially supported by Tampere Graduate School in Information Science and Engineering (TISE), Finnish Funding Agency for Technology and Innovation (TEKES), Academy of Finland, Nokia Foundation, Tekniikan edistämissäätiö (TES), Heikki ja Hilma Honkasen säätiö, KAUTE-säätiö, Ulla Tuomisen säätiö, and HPY:n tutkimussäätiö.

Finally, thanks to my family. Most of all, I thank my beloved wife Miia. Your love and understanding carried me through these years, and you were there for me even then, when I may not have deserved it. You and our lovely sons Emil and Alex keep showing me what is really important in this life.

Tampere, May 2008

Mauri Kuorilehto

TABLE OF CONTENTS

LIST OF PUBLICATIONS

This Thesis consists of an introductory part and the following publications that have been previously published. In the introductory part the publications are referred to as [P1], [P2], ..., [P6].

[P1] M. Kuorilehto, M. Hännikäinen, T. D. Hämäläinen, "Rapid Design and Evaluation Framework for Wireless Sensor Networks," *Ad Hoc Networks*, accepted, DOI: 10.1016/j.adhoc.2007.08.003.

[P2] M. Kuorilehto, M. Hännikäinen, T. D. Hämäläinen, "A Survey of Application Distribution in Wireless Sensor Networks," *EURASIP Journal on Wireless Communications and Networking, Special Issue on Ad Hoc Networks: Cross-Layer Issues*, vol. 2005, no. 5, pp. 774–788, December, 2005.

[P3] M. Kuorilehto, T. Alho, M. Hännikäinen, T. D. Hämäläinen, "SensorOS: a New Operating System for Time Critical WSN Applications," in *Embedded Computer Systems: Architectures, Modeling, and Simulation*, Lecture Notes in Computer Science, vol. 4599, S. Vassiliadis, M. Bereković, T. D. Hämäläinen (Eds.), Springer-Verlag, Heidelberg, Germany, 2007, pp. 431–442.

[P4] M. Kuorilehto, M. Hännikäinen, T. D. Hämäläinen, "A Middleware for Task Allocation in Wireless Sensor Networks," in *Proceedings of the 16th Annual IEEE International Symposium on Personal Indoor and Mobile Communications (PIMRC 2005)*, Berlin, Germany, Sep. 11–14, 2005, pp. 821–826.

[P5] M. Kuorilehto, J. Suhonen, M. Kohvakka, M. Hännikäinen, T. D. Hämäläinen, "Experimenting TCP/IP Performance for Low-Power Wireless Sensor Networks," in *Proceedings of the 17th Annual IEEE International Symposium on Personal Indoor and Mobile Communications (PIMRC 2006)*, Helsinki, Finland, Sep. 11–14, 2006, 6 pages.

[P6] M. Kuorilehto, J. Suhonen, M. Hännikäinen, T. D. Hämäläinen, "Tool-Aided Design and Implementation of Indoor Surveillance Wireless Sensor Net-

work," in *Embedded Computer Systems: Architectures, Modeling, and Simulation*, Lecture Notes in Computer Science, vol. 4599, S. Vassiliadis, M. Bereković, T. D. Hämäläinen (Eds.), Springer-Verlag, Heidelberg, Germany, 2007, pp. 396–407.

Supplementary Publications

Supplementary publications are not included into the Thesis, but due to their close relation to the Thesis they are separated as an own list. These are referred to as [S1], [S2], ..., [S6] in the introductory part.

[S1] M. Kuorilehto, M. Kohvakka, M. Hännikäinen, T. D. Hämäläinen, "High Level Design and Implementation Framework for Wireless Sensor Networks," in *Embedded Computer Systems: Architectures, Modeling, and Simulation*, Lecture Notes in Computer Science, vol. 3553, T. D. Hämäläinen, A. D. Pimentel, Jarmo Takala, S. Vassiliadis (Eds.), Springer-Verlag, Heidelberg, Germany, 2005, pp. 384-393.

[S2] P. Hämäläinen, M. Kuorilehto, T. Alho, M. Hännikäinen, T. D. Hämäläinen, "Security in Wireless Sensor Networks: Considerations and Experiments," in *Embedded Computer Systems: Architectures, Modeling, and Simulation*, Lecture Notes in Computer Science, vol. 4017, S. Vassiliadis, M. Berekovic, T. D. Hämäläinen (Eds.), Springer-Verlag, Heidelberg, Germany, 2006, pp. 167-177.

[S3] J. K. Juntunen, M. Kuorilehto, M. Kohvakka, V. A. Kaseva, M. Hännikäinen, T. D. Hämäläinen, "WSN API: Application Programming Interface For Wireless Sensor Networks," in *Proceedings of the 17th Annual IEEE International Symposium on Personal Indoor and Mobile Communications (PIMRC 2006)*, Helsinki, Finland, Sep. 11–14, 2006, 5 pages.

[S4] J. Suhonen, M. Kuorilehto, M. Hännikäinen, T. D. Hämäläinen, "Cost-Aware Dynamic Routing Protocol for Wireless Sensor Networks - Design and Prototype Experiments", in *Proceedings of the 17th Annual IEEE International Symposium on Personal Indoor and Mobile Communications (PIMRC 2006)*, Helsinki, Finland, Sep. 11–14, 2006, 5 pages.

[S5] M. Kohvakka, M. Kuorilehto, M. Hännikäinen, T. D. Hämäläinen, "Performance Analysis of IEEE 802.15.4 and ZigBee for Large-Scale Wireless Sen-

sor Network Applications", in *Proceedings of 3rd ACM International Workshop on Performance Evaluation of Wireless Ad Hoc, Sensor, and Ubiquitous Networks (PE-WASUN'06)*, Torremolinos, Spain, Oct. 6, 2006, pp. 48-57.

[S6] J. Suhonen, M. Kohvakka, M. Kuorilehto, M. Hännikäinen, T. D. Hämäläinen, "Cost-Aware Capacity Optimization in Dynamic Multi-Hop WSNs", in *Proceedings of Design, Automation and Test in Europe (DATE'07)*, Nice, France, Apr. 16-20, 2007, pp. 666-671.

LIST OF ABBREVIATIONS

ADC	Analog-to-Digital Converter
API	Application Programming Interface
CAD	Computer Aided Design
CORBA	Common Object Request Broker Architecture
CPU	Central Processing Unit
DCOM	Distributed Component Object Model
DVS	Dynamic Voltage Scaling
EFSM	Extended Finite State Machine
ETSI	European Telecommunications Standards Institute
FIFO	First In First Out
FSM	Finite State Machine
GDI	Great Duck Island
GUI	Graphical User Interface
HAL	Hardware Abstraction Layer
HIPERMAN	High Performance Radio Metropolitan Area Network
HIPERLAN	High Performance Radio Local Area Network
HVAC	Heating, Ventilation, & Air Conditioning
HW	Hardware
I2C	Inter-Integrated Circuit

I/O	Input/Output
ID	Identifier
IEEE	Institute of Electrical & Electronics Engineers
IP	Internet Protocol
IPC	Inter-Process Communication
JMS	Java Message Service
LR-WPAN	Low-Rate Wireless Personal Area Network
MAC	Medium Access Control
MIPS	Million Instructions Per Second
MCU	Micro-Controller Unit
MoC	Model of Computation
MOM	Message Oriented Middleware
NWK	Network
OS	Operating System
OSI	Open Systems Interconnection
PC	Personal Computer
PDA	Personal Digital Assistant
PIR	Passive Infra-Red
POSIX	Portable Operating System Interface
QoS	Quality of Service
RFID	Radio Frequency Identification
RMI	Remote Method Invocation
RPC	Remote Procedure Call
RTOS	Realtime Operating System

RTT	Round-Trip Time
SDL	Specification and Description Language
SLP	Service Location Protocol
SQL	Structured Query Language
SQTL	Sensor Query and Tasking Language
SW	Software
TCL	Tool Command Language
TCP	Transmission Control Protocol
TDMA	Time Division Multiple Access
TDOA	Time Difference of Arrival
TUT	Tampere University of Technology
TUTWSN	Tampere University of Technology Wireless Sensor Network
UART	Universal Asynchronous Receiver Transmitter
UI	User Interface
VM	Virtual Machine
WISENES	Wireless Sensor Network Simulator
WLAN	Wireless Local Area Network
WMAN	Wireless Metropolitan Area Network
WPAN	Wireless Personal Area Network
WSN	Wireless Sensor Network
WWAN	Wireless Wide Area Network
XML	Extensible Markup Language

1. INTRODUCTION

The number of embedded computing systems used in the everyday life is growing rapidly. The development is gearing towards the vision of ambient intelligence, where these small-sized, low-cost, embedded devices will become unobtrusive, or disappearing technology [67, 68, 124, 179, 180]. One of the key building blocks for the envisioned technology are *Wireless Sensor Networks (WSNs)* [85, 104]. WSN does not refer only to a single network implementation but a new emerging technology [13, 26, 40, 178].

A WSN consists of large number of tiny sensor nodes gathering information from the surroundings, processing it collaboratively, and communicating it wirelessly to locations, where data can be exploited [6]. The envisioned possibilities of the technology have resulted in constantly growing interest in research community [5, 6, 26, 31], and more recently also in industry [104].

1.1 WSN Technology Overview

The concepts related to the WSNs are introduced with an example scenario depicted in Fig. 1. A large number of nodes is randomly deployed in the vicinity or inside an inspected phenomenon [6]. The nodes self-organize and collaboratively coordinate the sensing process depending on the phenomenon [131]. Instead of sending raw data, each node refines its measurement results. The results are further aggregated or fused to obtain more accurate and complete application results while forwarding them towards a data gathering point, a sink node [40, 47, 153]. The sink node typically acts as a gateway to other networks and user devices [85]. The backbone infrastructure may implement a part of the complex data processing functionality of the WSN, and contain components for data storing, visualization, and network control [157].

In WSNs, networking is controlled by a layered protocol stack, an example of which is shown in Fig. 1. The stack does not strictly follow the layered Open Systems Interconnection (OSI) reference model due to the cross-layer design needed for the

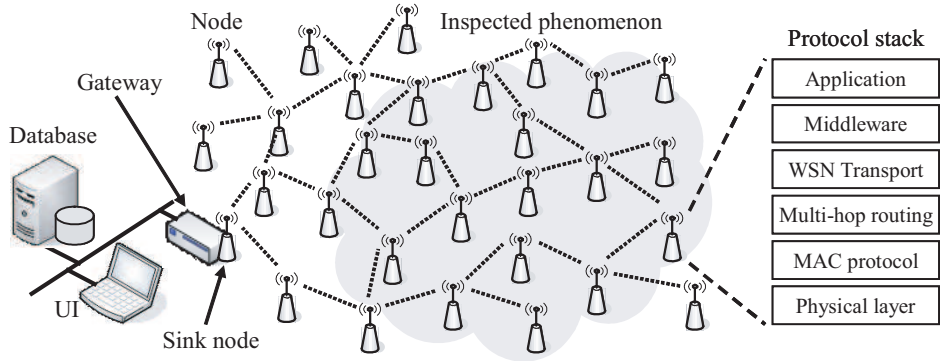


Fig. 1: An example WSN application scenario.

optimization of networking performance and energy efficiency [81, 149]. A Medium Access Control (MAC) protocol at the data link layer controls channel access, and maintains network connectivity and topology. A routing protocol at the network layer creates multi-hop paths between endpoint nodes, while a transport protocol implements end-to-end flow control, if necessary. At the presentation layer, a middleware sits between the application, the rest of the protocol stack, and an Operating System (OS). The main tasks of a WSN middleware are the abstraction of underlying communication and hardware, the formulation and coordination of application task execution within the network, and the combining of data gathered from different sources, i.e. data fusion [131].

The features of WSNs enable both monitoring and control functionality, and a variety of potential applications in environmental, home, health, and military domains [5, 6, 26, 85, 156, 178]. Thus, WSNs may cover scenarios from information-intensive video monitoring systems [4, 103] to low duty cycle, indoor and outdoor monitoring networks [88, 157]. Similarly, the number of nodes in a network may vary from a few to hundreds or thousands.

The diversity of applications and their contradictory requirements make a single, fixed solution suitable for all WSN applications impossible [50, 85, 132]. Existing standards for wireless communications, such as cellular telephone networks, WiMAX [186], Institute of Electrical & Electronics Engineers (IEEE) 802.11 Wireless Local Area Network (WLAN) [74], or Bluetooth [20] are not suitable for most WSN scenarios because of the large number of nodes and required energy efficiency. While IEEE 802.15.4 Low-Rate Wireless Personal Area Network (LR-WPAN) [77] and ZigBee [223] are promising standard technologies also for WSNs [13, 57], they

cannot be adapted to all potential applications [113].

Mainly due to the immaturity of the technology and the lack of standards, the commercialization of WSNs is still taking the first steps. Most of the available products are node platforms with little or no further functionality [161,193–198,202,204,206–208], but complete monitoring and control systems are also emerging [190–192,196, 199–201,203,205,206,209]. Still, most of the implementations rely on proprietary, mainly research initiated solutions [6,62,132].

Early WSN research merely focused on protocols and algorithms for low-power and scalable MAC [5, 64, 120, 136, 148, 172, 216] and multi-hop routing [2, 7, 13, 80, 93, 134]. Subsequently, more attention has been paid on higher layer issues such as middleware [59, 65, 66, 129, 218], in-network processing and collaboration [34,47,212,215,222], as well as on the definition of suitable abstractions for WSN programming and data access [54,56,92,101,105,115,142,149].

In order to meet the contradictory objectives resulting from the diversity of application domains, the protocols, algorithms, and nodes need to be tailored according to the application-specific requirements [21,104,132,168]. This requires design choices at different levels of abstraction, starting from the initial selection of components and ending to the fine tuning of compile and runtime parameters [50]. Consequently, system level design methodologies are needed for the management and configuration of the potential networking implementations so that the tightening performance and reliability requirements of the applications are satisfied [135].

1.2 Objective and Scope of Research

Chong and Kumar state that "the development of sensor networks requires technologies from three different research areas: sensing, communication, and computing (including hardware, software, and algorithms)" [26]. This Thesis discusses the design and implementation of software for wireless communications and application algorithms. Sensing and hardware issues are not discussed in detail.

This Thesis focuses on low-power WSNs consisting of resource constrained nodes. To define the resources more explicitly, the order of magnitude for the available resources is outlined by the capabilities of an example node. The average power consumption of a node should be from hundreds of microwatts to few milliwatts. The node is battery powered and equipped with a Micro-Controller Unit (MCU) having a couple of MIPS processing capacity, around 128 KB of code, and less than 10 KB of

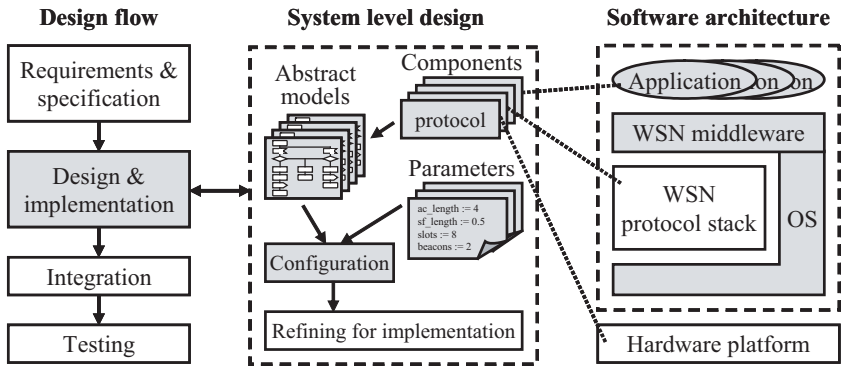


Fig. 2: Design flow, concepts in system level design, and a single node software architecture used in this Thesis. Design phases and components covered in this Thesis are shaded.

data memory. The communication bandwidth provided by the duty-cycled protocols varies from few bits per second to around a hundred kbps.

In this Thesis, WSNs are viewed as *embedded systems* that consist of hardware and software components [11, 41]. In general, an embedded system can be loosely defined as a special-purpose system that includes one or more software programmable parts [135,187]. This definition can be intuitively applied to both a single sensor node and the WSN as a whole. The analogy continues with design challenges: how much hardware is needed, is it possible to reuse existing components, how to minimize power consumption, does it work and fulfill the requirements, etc. [187]. These challenges are even more critical in WSNs, since the testing and evaluation of networks are difficult [50]. While a wide variety of design methodologies and tools has been proposed for the development embedded systems, these design issues have gained far less attention in WSNs [11,21, P6].

1.2.1 Research Topics

The main contributions of the Thesis are outlined in Fig. 2. The focus is on the abstractions, methods, and tools for *system level design* of WSNs, and on the architecture supporting *software implementation* for WSNs. The sequence of steps followed during the WSN design, i.e. *design flow* [187], is adopted from embedded systems. It is an iterative process that consists of requirements and specification, design and implementation, integration, and testing (verification) phases [84,133,187].

In system level design, a WSN is composed of a set of *components* that comprise node platforms, communication protocols, and applications. Initially, these components are described by abstract *models* defining the system functionality. During the design, the models are refined to detailed hardware and software implementations. The components can be either selected from a library and configured according to the application requirements or implemented from scratch [85]. *Configuration* means the composition of utilized algorithms and the tuning of parameters so that operational requirements are met. The requirements are set by the *deployment* case. The deployment in this context means the placing of the fully functional WSN and its applications to the final operation environment [132]. The functionality of a deployed WSN can also be changed through runtime reprogramming or reconfiguration [37, 126, 177].

The architecture for software implementation is provided by *systems software* that facilitates the prototyping and final implementation on physical node platforms. The systems software manages node resources and provides application independent services. In this Thesis, systems software consists of an OS and a middleware layer. Application scenarios illustrate the design and configuration of application-specific WSNs.

1.2.2 Research Methods and Results

The system level design methodology and the tools presented in this Thesis have been developed during the research. The research outcomes and the main topics they address are clarified in Table 1. The definition of abstract *deployment models*, their early refinement, and evaluation with large scale and long-term simulations are done with the Wireless Sensor Network Simulator (WISENES) tool. WISENES allows the graphical design of deployment models at *higher abstraction level*. Thus, the core functionality of the system is described without the consideration of low level implementation details, such as specific implementations of algorithms, composition of packets, or order of bit fields, etc. The design time evaluation in WISENES makes it possible to assess the suitability of a WSN configuration prior to the actual low level implementation.

The software architecture for prototyping is implemented by SensorOS. It supports time critical WSN applications with sophisticated resource and power management functionality. Further, SensorOS system services support the low level implementation of the deployment models with the same Model of Computation (MoC) as used in WISENES. On top of SensorOS, the developed middleware supports runtime coor-

Table 1: Main research results of this Thesis and the addressed topics.

Research outcome	Covered research topics
WISENES	- Abstract WSN models as communicating state machines - WSN design flow (design and implementation phase) - Component configuration and configuration evaluation
SensorOS	- Systems software for WSN applications - Single node software architecture
WSN node middleware	- Systems software for WSN applications - Runtime coordination and configuration
WSN deployment cases	- Evaluation and testing of presented tools and methods

dination and management of application tasks. Finally, these tools and environments are combined to a single design flow that aids design, configuration, and implementation of WSNs.

The challenges in the design and implementation of application-specific WSNs are highlighted by two example application scenarios. External data relay and indoor surveillance applications are implemented using Tampere University of Technology Wireless Sensor Network (TUTWSN) protocols and prototypes [88,157]. TUTWSN is a proprietary WSN technology developed in the Institute of Digital and Computer Systems at Tampere University of Technology (TUT). It is targeted to low data rate monitoring applications, in which the main objectives are scalability and energy efficiency [88]. Prototype node platforms, applications, and a backbone support infrastructure have been implemented for the evaluation of TUTWSN protocols and for exploring the possibilities of WSNs [157].

The technological details and implementations of node platforms, algorithms for MAC and routing, as well as detailed approaches for cross-layer design are outside the scope of this Thesis. Similarly, solutions for automated design space exploration [84] or code generation from WISENES to SensorOS on hardware platforms are not given. The limited resources of WSN node platforms still favor manual implementation [S1], even though a tight integration of the design environment and OS diminishes the resource consumption considerably [176].

To summarize, systematic methodologies and tools supporting different phases from application requirements to the WSN deployment are needed in order to manage the continuously expanding WSN design space [132]. The claim of this thesis is that the *design methods, tools, and software architecture presented in this Thesis facilitate the design and configuration of application-specific WSNs.*

1.3 Main Contributions

As a summary, the main contributions of the Thesis are:

- A survey of existing tools and frameworks for the design and simulation of WSNs, and a review of WSN OSs and middleware.
- Abstract deployment models for describing WSN functionality. The models divide the system to manageable, interactive entities.
- A high abstraction level design and evaluation framework, WISENES, for WSN protocol and application development.
- Systems software for WSN nodes, consisting of SensorOS and WSN node middleware. The software architecture aids the implementation of WSN applications and protocols on a very resource constrained WSN nodes.
- Two deployment cases that express the challenges related to design and implementation of real WSN deployments, and prove the feasibility of developed methods and tools.

1.4 Thesis Outline

This Thesis consists of an introductory part and six publications [P1-P6]. The introductory part presents WSN technology, gives technical background and motivates the work. The main results are presented in the publications. The rest of the introductory part is organized as follows:

Chapter 2 introduces WSNs and highlights their special characteristics and requirements. WSN applications and application domains are also discussed. The chapter includes also a short overview of existing and emerging standards related to WSNs.

Chapter 3 focuses on the systems software for WSNs. The chapter concentrates on the general concepts and related work regarding OSs and WSN middleware.

Chapter 4 discusses the WSN design challenges and motivates the need for a systematic design flow. An overview of existing methodologies and tools is given.

Chapter 5 composes the research results described in detail in publications [P1-P6]. The main focus is on the tools and methodologies related to WSN design and prototyping with WISENES and SensorOS.

Chapter 6 summarizes the publications included in the Thesis.

Chapter 7 concludes the Thesis.

2. WSN CHARACTERISTICS AND DESIGN FACTORS

This chapter outlines the design space of WSNs based on the current and envisioned application domains and experimental implementations. Standards related to WSN communication and software architecture are presented as a baseline for WSN design. From the outlined design space, a set of common characteristics is derived to represent the unique properties of WSNs. While these characterize most of the WSN scenarios, they do not describe individual deployments. Therefore, the design factors that formalize more detailed requirements for each case are given.

2.1 WSN Applications

The tasks for WSN applications depend mainly on the sensors and actuators available in nodes, since the in-network processing and communication capabilities allow a rich set of application functionality. Currently, the actuators are limited mostly to servo drives and different types of switches, while the physical quantities that can be measured with already existing technologies are diverse. These include for example temperature, humidity, pressure, acceleration (vibration), sound, light (luminance, image), magnetic fields (compass), location, chemical compositions, and mechanical stress [6,43,85].

2.1.1 Envisioned Application Domains

The measured physical quantities and their fusion produce input information for the applications. The typical WSN application domains envisioned in the literature are:

Home automation: WSNs are envisioned as one of the key building blocks for the smart homes and intelligent buildings. Example usage scenarios include Heating, Ventilation, & Air Conditioning (HVAC) control, and local and remote management of home appliances [6, 85].

Environmental monitoring: Environmental applications cover scenarios from condition monitoring to wildlife tracking. WSNs can be used for instrumenting both agriculture and wild nature. Another application area is catastrophe (e.g. wildfire, earthquake, tsunami) prevention and disaster relief [6, 26, 31, 62, 85, 165].

Industrial monitoring and control: The replacement of traditional cabling in machine surveillance and maintenance systems is the main application for WSNs. Further, WSNs can be benefitted for managing logistics [26, 85].

Military: Military applications were the driver for the first WSN research initiatives [26, 132]. While the scope has expanded rapidly, military is still one of the main application areas. Usage scenarios cover for example intelligence, surveillance, reconnaissance, and targeting [6, 26].

Personal security and asset management: WSNs are envisioned to replace or extend existing wired alarm systems in home, office, and other public environments such as airports and factories. Compared to wired systems, WSNs allow faster deployment, more flexibility, and larger area coverage [6, 26].

Traffic control: WSNs make it possible to extend the traffic monitoring and control systems farther away from the most critical points. Temporary situations such as roadworks and accidents can be covered in place [26, 85]. A far reaching vision embeds a WSN node to every vehicle. These nodes share the traffic information, generate warnings of accidents and jams, and guide drivers in route selections [26, 128].

Health care: Biomedical sensor networks can be used to gather physiological data directly from patients [6, 85, 139]. Other health care related application areas are drug administration, and tracking of doctors and patients in hospital premises [6, 85].

2.1.2 Application Tasks

In most of the WSN applications, the nodes are either data sources or data sinks. Source nodes perform sensing and disseminate the data to sink nodes. Typically, sink nodes relay the information to an external client, e.g. a human user [5, 85]. Applications, in which the gathered data is processed within the network and used for controlling actuators are still rare [3].

Even though the application domains are diverse, the applications share some basic characteristics. In general, four typical tasks that are independent of the application domain can be identified [6, 85, 110]:

Monitoring: Determine the value of a parameter in a given location or at the coverage area of the network. Typically, the task is completed using periodic measurements.

Event detection: Detect the occurrence of events of interest and their parameters. The detection can be performed either by a single node or by a group of nodes depending on the complexity of the event.

Object or event classification: Identify an object or an event. This requires the combination of data from several sources and collaborative processing to conclude the result.

Object tracking: Trace the movements and position of a mobile object within the coverage area of the network.

2.1.3 Experimental Applications

A set of experimental application deployments is listed in Table 2 in order to illustrate the application types in practice. The selected applications are based on research proposals, since commercial networks are still quite rare and detailed information about them is not available. The columns of the table give an overview of the characteristics of the current deployments. The scale defines the number of nodes, and the lifetime the length of the deployment. The data interval illustrates the activity of the network by giving the frequency of data communication.

Most of the presented WSNs are targeted to environmental monitoring. The scale of the deployments is still quite limited and the lifetimes do not reach a year. Further, the activity is quite infrequent in most of the applications.

Weather conditions in forest environments are monitored in the heathlands of Northern Germany [171], in Kangasala [157], and by NIMS [15], Macroscope [170] and PicoRadio project [125]. CORIE [154] measures the temperature and pressure at stations located in the Columbia River, while GlacsWeb [106] measures similar aspects in glaciers. An agricultural WSN deployment to a vineyard setting is discussed in [16]. Wildlife is observed by Great Duck Island (GDI) [164] that monitors the

Table 2: Examples of research-based experimental WSN deployments.

Deployment	Scale	Lifetime	Data interval
Environmental monitoring			
CORIE [154]	18	N/A	1-15 min
GDI [164]	150	4 months	20 min
GlacsWeb [106]	8	few months	1 day
Heathland [171]	24	16 days	1 hour
Kangasala [157]	19	>4 months	1 minute
Macroscope [170]	33	44 days	30 seconds
NIMS [15]	7	N/A	3-20 seconds
PicoRadio [125]	25	1-2 months	5 seconds
Vineyard [16]	65	6 months	5 minutes
ZebraNet [221]	7	12 months	8 minutes
Shellfish [30]	4	<1 day	5 minutes
Building monitoring			
SensorScope [137]	20	2 months	5 minutes
Wisden [213]	10	<1 day	<1 second
Object tracking			
Multi-target tracking [114]	144	<1 day	few seconds
PEG [141]	100	<1 day	0.5 seconds
PinPtr [145]	56	<1 day	<1 second
Vehicle tracking [130]	6	N/A	<1 second
VigilNet [63]	70	<1 day	<1 second

occupancy of seabird burrows, by ZebraNet [221] that tracks animal movements, and by WSN for a short period monitoring of shellfish catches [30] .

In buildings, the monitored aspects are either structures or conditions for HVAC control. Wisden [213] tracks vibrations in building structures, while SensorScope [137] measures light, sound, and temperature in an indoor environment.

Due to the relative complexity of object tracking, the number of proposals is quite limited and the lifetimes of the deployments are quite short. In most of the cases, the given lifetime is not restricted by the battery capacity but the fact that the deployment is merely an experiment performed only for a short period of time. Vehicle tracking is implemented using ultra-sound in PEG [141], with infrared light in [130], and with multiple different types of sensors in VigilNet [63]. Multiple targets are tracked simultaneously in [114] using Passive Infra-Red (PIR) sensors. PinPtr [145] locates a shooter based on the Time Difference of Arrival (TDOA).

2.2 Communication and Systems Software Standards Related to WSNs

Standards and de facto technologies in wireless communications and systems software are diverse, but only few are suitable for low-power WSNs. Even these cannot be applied to all WSN applications. Instead, the standards set a common reference platform, to which application-specific tailoring of WSNs can be based on [113]. Furthermore, especially in industrial applications the inter-operability and continuity guaranteed by standards are important design drivers [23, 185].

In this Thesis, the standards set a starting point and reference for the research. The discussed standards are limited to wireless communications, OSs, and middleware. Design and programming languages are not discussed, since they do not address system level design issues.

2.2.1 Wireless Communication Standards

Currently, the wireless communication technologies are aiming towards both higher data rates, and lower cost and power consumption [111]. From the standard technologies, Wireless Wide Area Networks (WWANs), such as cellular telephone networks, and Wireless Metropolitan Area Networks (WMANs), e.g. harmonized IEEE 802.16 [76] and ETSI High Performance Radio Metropolitan Area Network (HIPERMAN) [46] standards promoted and certified by the WiMAX Forum [186], target to large area coverage and broadband data communication.

WLANs are designed for replacing or extending wired Ethernet. Currently, IEEE 802.11 WLAN [74] is the prevailing technology, which is expanding rapidly to businesses, homes, and most recently to public hot spots [108, 183]. Ad-hoc networking support is being standardized for IEEE 802.11 WLAN by the ESS Mesh Networking Task Group [79]. Other WLAN technologies, such as High Performance Radio Local Area Network (HIPERLAN)/2 [45], have not been able to penetrate to the WLAN market.

The separation of WLANs and Wireless Personal Area Networks (WPANs) is not distinct [71]. In general, WPANs target to low-cost and low-complexity connection of personal devices, like laptops, Personal Digital Assistants (PDAs), and mobile phones [144]. Bluetooth [20] and IEEE 802.15.4 LR-WPAN [77] together with ZigBee [223] are currently the most well-known technologies. Several industry initiatives have proposed improvements or alternatives for Bluetooth and ZigBee. These include Wibree [184], Z-wawe [220], MiWi [48], and ANT [8] that target to lower power

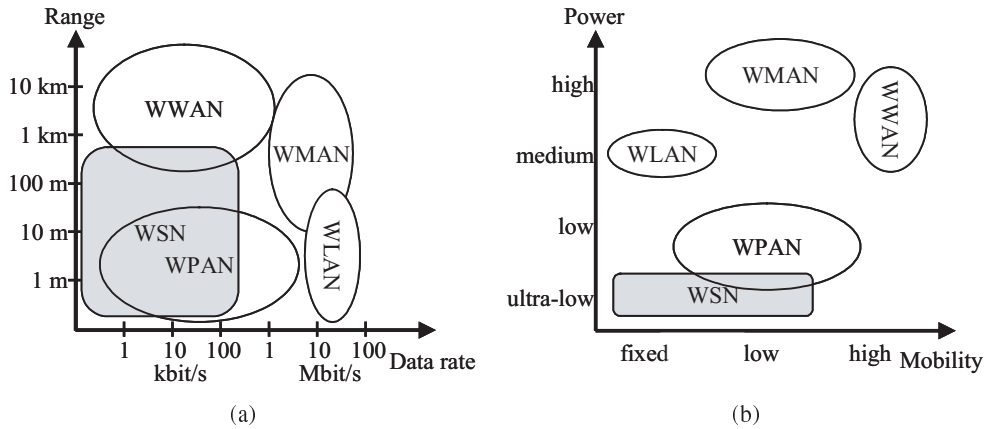


Fig. 3: Comparison of WSNs to other wireless technologies in terms of (a) range and data rate, and (b) power consumption and mobility.

consumptions or cost effectiveness compared to the standard technologies.

IEEE 1451.5 working group targets to the inter-operability of sensors by standardizing the interfaces and use of wireless communications in smart transducer systems [78,163]. The wireless interfaces used by IEEE 1451.5 include standard WPAN and WLAN technologies [163]. The main focus of IEEE 1451.5 is on the interfaces and it does not define any new protocols for communication.

The suitability of the wireless technologies for low-power WSNs is assessed in Fig. 3. An IEEE originated classification categorizes technologies according to their range, data rate, and power consumption [70,71,144]. In addition to these, the figure considers also the mobility of nodes [23, 108]. Scalability is left out from the comparison, since none of the technologies reaches the scales of tens of thousands of nodes envisioned for WSNs. The power consumption caused by high data rates and long range communications in WWAN, WMAN, and typically also in WLAN make them unsuitable for low-power WSNs. WPANs are closest to meet the unique requirements of WSNs.

2.2.2 Systems Software Standards

Several standards or de facto standards defining OSs and especially middleware architectures have been widely adopted in computer systems. While standard technologies for OSs and Realtime Operating Systems (RTOSs) are quite rare, several middleware standards exist for implementing distributed processing in computer networks and clusters.

Widely used OSs, such as Microsoft Windows and Linux, have not been standardized as is. Yet, they as well as most of the other common OSs and a part of RTOSs conform Portable Operating System Interface (POSIX) standard [75]. POSIX specifies a set of OS services required for application implementation and a common Application Programming Interface (API) for accessing these. More technology specific variations of the POSIX standard have been defined by Single Unix Specification [116] and Linux Standard Base [99].

The middleware standards in computer networks specify common interfaces and conventions for transferring execution to remote locations, messaging between end points, and for service discovery [P2]. The main objective is to abstract the heterogeneities in hardware platforms, communication protocols, OSs, and programming languages [174]. Common Object Request Broker Architecture (CORBA) [174], Java Remote Method Invocation (RMI) [160], and Microsoft's Distributed Component Object Model (DCOM) [73] are object-oriented architectures for remote processing. They rely on client-server model and abstraction through interface specifications. Java Message Service (JMS) [159] is an API of a Java-based Message Oriented Middleware (MOM) for communication between remote clients. Also service discovery approaches utilize client-server architecture. Service Location Protocol (SLP) [58] implements discovery in Transmission Control Protocol (TCP)/Internet Protocol (IP) networks and Jini [158] in Java environment. UPnP [107] targets to general interoperability.

In general, the memory requirements of these technologies exceed the resources available on sensor nodes [P2]. Therefore, these standards mainly steer the functional requirements for WSN systems software.

2.3 Unique Characteristics of WSNs

Even though the WSNs possess some similarities with the other wireless ad-hoc networks, there are several characteristics that set WSNs apart from other communication networks [6]. While all of these characteristics may not be applied together, the following lists the main unique properties of WSNs.

Communication Paradigm: The type of service provided by WSNs differs from traditional wired and wireless communication networks [43]. Compared to e.g. TCP/IP , the communication does not occur between specific endpoints but it merely originates according to geographical locations or data content

(e.g. "nodes in Tampere region" or "nodes with measured temperature value above 20 °C") [153, P5]. This kind of data-centric nature makes individual node Identifiers (IDs) unimportant.

Application-specific: A single end device in a computer network may have multiple applications, and the network should be able to serve each application according to its Quality of Service (QoS) requirements for throughput, delay, and reliability. Conversely, a WSN is deployed to perform a specific task or a small set of tasks. This makes it possible to use application-dependent node platforms, communication protocols, data aggregation, and in-network processing and decision making [153].

Unpredictability: WSN are subject to a number of uncertainty factors [153]. First, nodes are error-prone due to harsh operating conditions. Communication links are unreliable because of node errors, simple modulations, mobility of nodes, and external or internal interferences. Also, WSN protocols have a built-in dynamic nature caused by continuously changing data structures, e.g. routing tables, used for decision making. These aspects make even static WSNs unpredictable [153].

Scale and Density: Compared to other wireless networks, in most WSN scenarios the number of nodes and their density is few orders of magnitude larger [5, 43]. These factors depend on the needed sensing coverage and robustness (redundancy).

Resource constraints: The nodes in low-power WSNs are small sized and battery powered. As hardware design is guided by these factors, the computation, communication, memory, and energy resources of nodes are very limited [6, 153, P2].

Nature of deployment: The deployment of nodes can be random to harsh or hostile environments [5]. This hinders the maintenance, and makes the replacement of nodes impractical [157]. Still, the requirements and applications of the WSN may change also during the deployment. This implicates that runtime reconfiguration and reprogramming are needed [37, 126].

2.4 Design Factors for WSNs

Several efforts for formalizing the design factors for WSNs have been published [5, 6, 85, 132, 144, 168]. These concentrate mainly on the functional issues related

to the communication protocols [5, 6, 85, 168]. In order to cover also system level aspects, the functional factors are complemented with deployment factors characterizing physical WSN deployments [132].

The design factors define the requirements for the WSN design. Further, they are suitable for comparing different deployments and for identifying similarities between applications [5, 132]. This facilitates the reuse and selection of existing node platforms, communication protocols, and algorithms for different application scenarios.

Deployment Factors

The deployment factors extract the main requirements for the physical deployment. They guide the selection of node platforms, and their placement to the target environment.

Network Deployment: The actual deployment process can be random or nodes can be manually set to designated places. Nodes are either placed only once or nodes and batteries can be iteratively replaced [132].

Cost, size, resources, and energy: A node can be about the size of a brick, matchbox, or grain. Nodes are either mains powered, or the operating energy can be stored in batteries, or scavenged from the environment. Manufacturing costs of nodes can vary from few cents to hundreds of euros [5, 132].

Communication modality: Typically, the communication modality in WSNs is radio waves but other modalities are available. These include light, inductive coupling that is used in Radio Frequency Identification (RFID) systems, capacitive coupling, sound, or ultrasound [6, 132].

Connectivity and coverage: The connectivity of the network depends on the radio coverage, while the coverage itself in this context defines how certainly and reliably an event can be detected with physical sensors. These can vary from fully connected and covered to sporadic, in which case only parts of the area can be monitored and nodes are only occasionally in the communication range of other nodes [132].

Functional Factors

The functional factors define the guidelines for protocol and application behavior. While the deployment factors have most influence before a WSN is deployed, these

have more effect on the active operation of the network.

Fault tolerance: The overall operation of a WSN must be robust against failures of individual nodes. In addition to algorithms increasing reliability, this can be addressed by redundant deployments, when more nodes than strictly necessary are used [6, 85].

Autonomous operation: The possibilities for manual administration of a WSN depend on the application. However, self-organization, self-configuration, and error recovery are typically required mechanisms in WSNs due to the large number of nodes and their high density [85].

Lifetime: The definition of the lifetime depends on the application. It can be defined as the time until half of the nodes die or when network stops delivering application data reliably [168]. In general, the lifetime of battery powered nodes can be from hours to several years [132].

Dynamic nature: Mobility and interference from the surroundings create a dynamic operating environment for WSNs. Mobile nodes can be attached to moving objects, influenced by forces of nature (e.g. wind, water, earthquake, avalanche, landslide), or have actuators enabling node mobility [132, 168].

Network performance: In low-level, the network performance can be characterized by throughput, latency, jitter, and the number of lost and duplicated packets [71]. Typically, improved performance results in increased resource consumption. This trade-off needs to be adapted according to the application requirements [4, 129, 153, 168].

Accuracy of results: The required accuracy of results depend on the application. In several WSN scenarios, distributed collaboration among nodes is required to obtain needed accuracy [85, P2].

Security: The need for security and privacy is evident in certain application domains e.g. in health care and military [119, S2].

3. SYSTEMS SOFTWARE FOR WSNS

From a design point of view, the systems software (term system software is used interchangeably) creates a runtime environment that supports design time abstractions in the final implementation on node platforms. For WSNs, a major challenge for the systems software is how to provide generalized abstractions for wide variety of applications in the constraints set by node resources. Typically, this results in trade-offs between the level of abstraction, expressiveness, and the efficiency of implementation [129].

In WSNs, the separation between OSs and middleware is not explicit. In computer systems, middleware sits between applications and OS that provides already a rich set of functionality and well-defined interfaces. In WSNs, the definition and implementation of such abstractions and interfaces in a layered manner is difficult due to the limited resources of nodes. Therefore, OS and middleware functionality can be combined to a single framework [129]. Regardless of the separation between OS and middleware in an actual implementation, in this chapter they are consider as separate entities.

This chapter discusses requirements and basic concepts of both OSs and middleware for WSNs. Further, related research is surveyed on both areas. An OS is a lower level component for hardware abstraction that handles resource management and supports concurrent execution for managing parallel data and control flows [69, 85, P2]. On top of OS, middleware provides an abstraction that supports desired programming paradigms and hides the details of underlying network infrastructure. Moreover, middleware controls application task execution and coordinates collaborative in-network processing [131, P2].

3.1 Operating Systems

In computer systems, OSs facilitate the implementation and use of computer programs and resources for both application programmers and end users. OS services

include the concurrent execution and communication of multiple programs, access and management of Input/Output (I/O) devices and permanent data storage (file system), and control and protection of system access between multiple users [152]. In WSNs, these services are only partially required [153].

3.1.1 OS Requirements

WSN OSs should enable concurrent execution of multiple realtime applications and protocols with a considerably smaller memory footprint than that of full feature general purpose OSs [153]. The main requirements for OSs are the management of limited resources and the facilitating of the development of WSN applications [69]. The requirements are summarized more specifically in the following list.

Small memory footprint: A typical WSN node has 128 KB code and less than 10 KB of data memory [68, 157, P2]. In order to preserve most of the memory resources for application and protocol implementation, available memory for OSs is very limited [44, 69].

Concurrent programming: WSN nodes handle concurrent data flows that must be served in timed manner [69, 85, 153]. Data is originated by sensors or routed from other nodes. OS needs to support concurrency of tasks in order to interleave high-level processing of parallel data flows with low-level events and communication protocol functionality [69].

Energy management: System level energy management is required due to the battery powered nature and lifetime requirements of WSN nodes. An OS must control the sleep states, shutdown unused peripherals, and use Dynamic Voltage Scaling (DVS) techniques in order to make efficient use of energy resources [44, 85].

Realtime operation: The tight relation to real world inherently sets tight timing constraints to WSN operation [153]. A realtime operation is required for meeting deadlines associated to data and event processing [44].

Memory management: WSN communication requires local data buffering and variable sized control data structures for network maintenance. In order to get a full use of data memory, dynamic memory allocation is needed [44].

Hardware abstraction: One of the main services of OSs is to abstract hardware access behind a unified interface [152]. This concerns MCU and peripheral devices, such as sensors, actuators, radios, and other I/O devices.

Robustness: An OS needs to be reliable in its operation and support reliable execution of applications, since fault tolerance is one of the key requirements for WSN operation [69].

Modularity: In addition to protocol stack optimization, the diversity of application requirements necessitates also application-specific tailoring of OSs. Thus, OS services and device drivers that are not required by deployed applications and protocols should be excluded from the tailored OS [69].

3.1.2 Basic Concepts of WSN OSs

General purpose OSs follow either monolithic or microkernel architecture. A monolithic OS implements all system services inside the kernel. A microkernel includes only the most essential features of OS, while other services are implemented in external processes or servers [152]. In WSNs, such architectural choices are not as evident, since OS protection and interfaces are not well-established [44, 55].

Even though networking is a key service in WSNs, there are varying opinions whether it should be embedded to the OS kernel or not [19, 24, 44, 69]. While an integrated network stack eases the application development, it hinders the interchangeability and configuration of protocols [69]. A modular stack keeps the kernel smaller and simpler, but it has to compete of Central Processing Unit (CPU) time with application tasks, which increases the risk of malfunction and starvation.

Currently, the configuration of a WSN is mainly considered as a design time issue. Yet, the dynamic nature of networks requires also runtime reconfiguration [177]. Therefore, a dynamic loading of both application and OS modules are useful in WSNs [24, 37, 38, 61].

Basically, two main architectural choices are utilized for concurrent processing in WSN OSs. A process-based *preemptive multithreading* is adapted from computer systems. Another approach is *event-based*, which is motivated by the event-driven nature of WSNs [85].

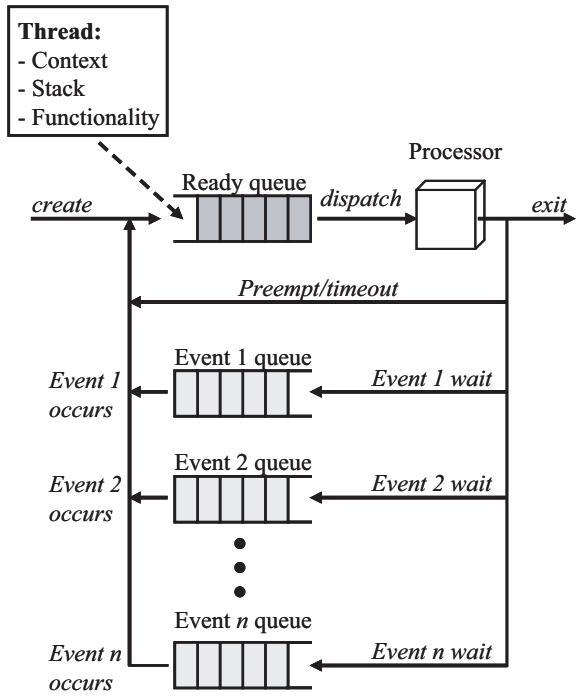


Fig. 4: Thread management in preemptive multithreading OSs [152].

Preemptive Multithreading

Preemptive multithreading offers a common programming model to application developers. Fig. 4 illustrates the queuing of threads and their state changes in a preemptive scheduler. Each process or thread (thread is used from now on) is an independent functional unit that is executed in its own context. A running thread may be swapped from CPU due to preemption or a blocking wait operation initiated by an OS system call. In former, the thread is put back to the ready queue, and in latter to an event-specific wait queue. When the event occurs, the thread is put to the ready queue [152].

Preemptive multithreading offers a seemingly parallel execution of threads and enables priority-based scheduling, in which the tasks with highest importance can be scheduled at defined timestamps [44]. However, in WSNs the context switching adds considerable overhead to the processing of typically very simple application tasks. Also, a preemption requires that each thread has a dedicated memory for its context and stack [38].

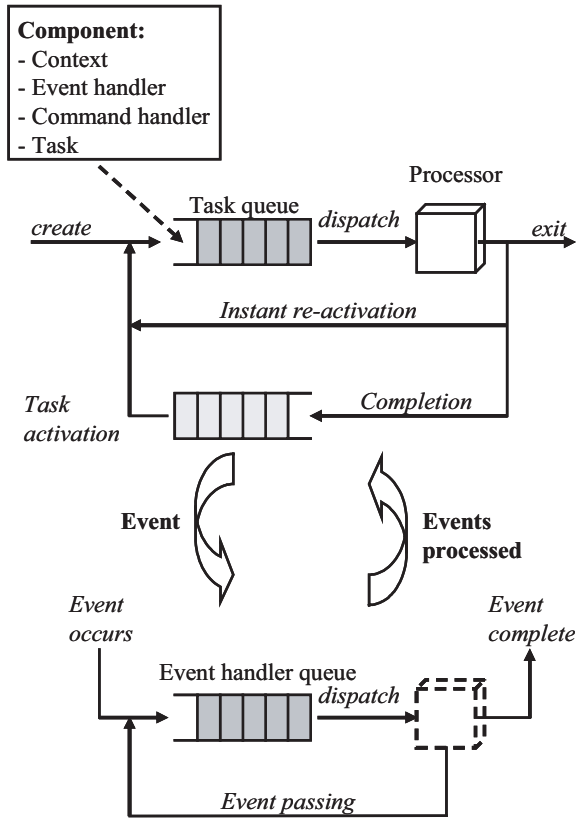


Fig. 5: Scheduling of events and tasks in event-based OSs.

Event-based Kernels

Event-based kernels are built around event-handlers, as depicted in Fig. 5. An event-handler reacts to an event, which is initiated by a hardware interrupt that indicates e.g. a reception of data from radio, a timer event, or available data from a sensor. An event is propagated upwards through a layered architecture. Data is processed in tasks designated for regular processing. When a task needs services from lower layers e.g. for sending data, it issues a command that traverses downwards in hierarchy [69].

Each event and command handler, and regular tasks run to completion. Thus, context-saving is not required. Reactiveness is obtained through event-handlers that preempt currently running task or command processing [69]. As a consequence, especially the data memory usage of such OSs is low. As a drawback, an event-driven programming model is somewhat difficult to comprehend [38, 44, 85]. Since each task runs to completion, lengthy task processing needs to be partitioned at the application level

[38].

3.1.3 Related Research on WSN OSs

Embedded RTOSs, such as OSE, QNX Neutrino, VxWorks, and Symbian OS are widely used in industrial control and telecommunication systems. However, their memory consumption is too large for resource constrained sensor nodes [19, 69]. Small memory footprint general purpose RTOSs, such as FreeRTOS or μC/OS-II, do not meet the strict timing and power mode utilization requirements of WSNs [19].

Existing OS proposals for WSNs are summarized in Table 3. SensorOS presented in [P3] is included to facilitate comparison. The dimensions in the table highlight the resource consumption, realtime capability, and configurability of OSs. The second column defines kernel type adopted by the OS and the third one the principal MCU architecture, to which OS has been implemented. Two following columns present OS resource consumption with the given target hardware. The sixth column defines whether OS guarantees soft or hard deadlines, or none at all. Reconfiguration is typically supported either by allowing bootloader type reprogramming of the whole code image, or by enabling dynamic loading of tasks. The last column lists additional services provided by the OS. The values given in the table are from cited reference publications, unless stated otherwise. Therefore, e.g. memory consumption may not be valid for latest release of OS, or it may depend greatly on application-specific configuration.

Preemptive Multithreading OSs

Preemptive multithreading for sensor nodes with POSIX style API is implemented in MOS [19]. MOS implements priority-based scheduling, synchronization between threads, and power saving features. Network stack is implemented as an OS service. The data memory consumption of MOS kernel is only 500 B but this does not include the stacks of application threads [19].

Similar approach to MOS is taken in nano-RK [44], which implements also a fixed priority preemptive scheduling. In addition to the features implemented in MOS, nano-RK introduces resource reservation policy, through which application threads can allocate CPU time, network bandwidth, and sensor resources [44].

RETOS [24] extends MOS and nano-RK features by supporting dual mode operation that separates kernel and user modes. RETOS functionality can be reconfigured by

Table 3: Comparison of existing WSN OSs.

OS	Main approach	MCU	Code (KB)	Data (B)	Realtime	Runtime reprogramming	Other services
SensorOS [P3]	preemptive	PIC18	6.8	115	soft	none	μs resolution timing
MOS [19]	preemptive	Atmel AVR	~14 [1)]	~500 [1)]	soft	none	network stack, remote command shell
nano-RK [44]	preemptive	Atmel AVR	~10 [2)]	~2000 [2)]	hard	none	network stack
RETOS [24]	preemptive	Atmel AVR	23.7	1125	soft	module relocation	network interface, memory protection
t-kernel [55]	preemptive	Atmel AVR	28.2	~2000	none	none	virtual memory
TinyOS [69]	event-based	Atmel AVR	3.4 [3)]	226 [3)]	none	none	active messages allowing RPC
SOS [61]	event-based	Atmel AVR	20.0	1163	none	dynamic loading of components	dynamic memory pool, runtime integrity check
Bertha [98]	event-based	8051	~10 [1)]	~1500 [1)]	none	tasks	bulletin board system for RPC
BTnodes OS [17]	event-based	Atmel AVR	34.7 [18]	1029 [18]	none	full system reprogramming	Bluetooth stack, tuple space, Java smoblets
CORMOS [214]	event-based	Atmel AVR	5.5	130	none	none	transparent RPC
Contiki [38]	event-based / preemptive	Atmel AVR	3.8	>230 [4)]	none	dynamic loading of processes	Service abstraction

1) Accurate value not given

2) Includes eight tasks, eight mutexes, and four 16B network queues

3) Includes a simple multi-hop protocol and temperature sensing application

4) Data memory depends on number of processes, maximum event queue length, and number of multithreaded tasks

runtime loading of modules. OS does not include full feature network stack, instead it offers layered interfaces for networking [24].

Like RETOS, *t-kernel* [55] incorporates OS protection. The protection is implemented by modifying application code during runtime. The same approach is used also for virtual memory abstraction. Yet, runtime code modification may incur unexpected delays and increases code size [24,55].

Event-based OSs

A widely known WSN OS, TinyOS [69], uses a component-based event-driven approach for task scheduling. Software is divided into components encapsulated in frames. Each component has a separate command handler for upper layer requests and an event handler for lower layer events. The processing is done in atomic tasks. The frame defines the context in which the event and command handlers, and the tasks related to the component are executed [69].

In TinyOS, components and their communication is statically defined at compile time. A more dynamic approach is taken in SOS [61] that adopts the component model from TinyOS but allows runtime loading and unloading of components. SOS kernel maintains modules, handles memory management, and implements communication between modules [61].

Similar architecture to SOS is implemented in Bertha, OS for Pushpin nodes [98]. The components, referred to as process fragments, can communicate with other process fragments in the same and neighbor nodes through a bulletin board system. Runtime loading and unloading service allows the migration of process fragments between nodes [98].

A lightweight OS for BTnodes [17] does not support runtime configuration of events or tasks. In BTnodes OS, all application processing is done within the event handlers. BTnodes can share their sensory data through a distributed tuple space that abstract the origins of data [17].

The basic architecture of CORMOS [214] consists of events and event-handlers that are organized to modules. The communication events between modules are signaled through event paths. An event path can exist between local or remote modules, which makes the communication between distributed application modules possible [214].

Event-handler based kernel of Contiki [38] supports dynamic loading of modules. A program module that contains required relocation information can be linked to the

OS kernel during runtime. Unlike in other event-handlers, an interrupt handler does not generate an event directly but sets a flag that is checked by a polling mechanism. In addition, Contiki implements a support for preemptive multithreading through a library on top of the event-handler kernel [38].

Summary of WSN OSs

As stated, in WSNs also OS and its parameters need to be configured application-specifically [85]. The suitability of OS depends on the resources of node platforms and the requirements of applications and protocols. Thus, a commercial embedded RTOS may be the best selection, if WSN nodes have enough resources and application requires performance and services of such OS.

In general, the memory used by event-based kernels is smaller than that of preemptive ones. On the other hand, event-based OSs suffer from the programming model and incapability to support long-term processing [38, 95]. More general purpose architecture of preemptive OSs supports design time abstractions better. In Contiki [38], a designer can freely map the tasks to event-handler or to multithreading depending on the requirements and available resources [38].

A priority-based preemptive scheduling allows time critical operation of high priority tasks. In event-based kernels, only event-handlers can be guaranteed to run in time-critical manner [69, 85]. An event-handler needs to be short and mainly pass an event to a task for processing. This may not be sufficient for time critical signal processing applications or networking protocols.

SensorOS [P3] has an accurate time concept for implementing time critical functionality. Further, compared to other preemptive multithreading OSs, SensorOS has a considerably smaller memory footprint.

3.2 Middleware

The main objective of computer system middleware, such as CORBA, is to allow the execution of distributed systems on top of heterogeneous hardware and software architectures [174]. A middleware implements an abstraction layer between applications and network protocols and OSs [174]. Since WSNs lack standardized network and OS interface, a unified basis for WSN middleware is absent [131].

3.2.1 Middleware Requirements

As in case of OSs, legacy computer system middleware is not suitable for WSNs due to the differences in programming paradigms and the constraints set by limited resources [129, 153, 218]. A WSN middleware should implement a runtime environment that supports development, maintenance, deployment, and execution of WSN applications [59, 131]. The main requirements derived from these functionalities are listed below.

Abstraction: Similarly to computer system middleware, WSN middleware should abstract the heterogeneity of underlying hardware and software architectures [59, 131].

Task formulation: The requirements of a deployed WSN may vary, which changes the tasks the network needs to perform [49]. Therefore, a formulation is required for defining the tasks of the network [131].

Task allocation: A middleware should support programming paradigms and data-centric nature of WSNs [131]. Middleware is responsible for translating used programming paradigms to tasks and allocating these to individual sensor nodes [56, 153].

Aggregation, data fusion, and in-network processing: A WSN middleware possesses application knowledge that is required for controlling in-network processing [59, 131]. Therefore, middleware manages data aggregation and fusion [59, 129].

Access interface to external networks: In addition to in-network control, WSN middleware should extend the network abstraction also for external networks and devices. This can be used for accessing WSN data but also for outsourcing WSN computation to more powerful platforms [129, 131].

Application QoS: In this context, the application QoS defines how different design factors can be satisfied by the WSN. A middleware adapts application QoS to networking QoS, makes necessary trade-offs, and takes actions that are needed to meet the required level of service [60, 65].

3.2.2 WSN Middleware Approaches

Due to the immaturity of technology, there are no unified classification for WSN middleware. In the classification presented in Fig. 6, WSN middleware is first cat-

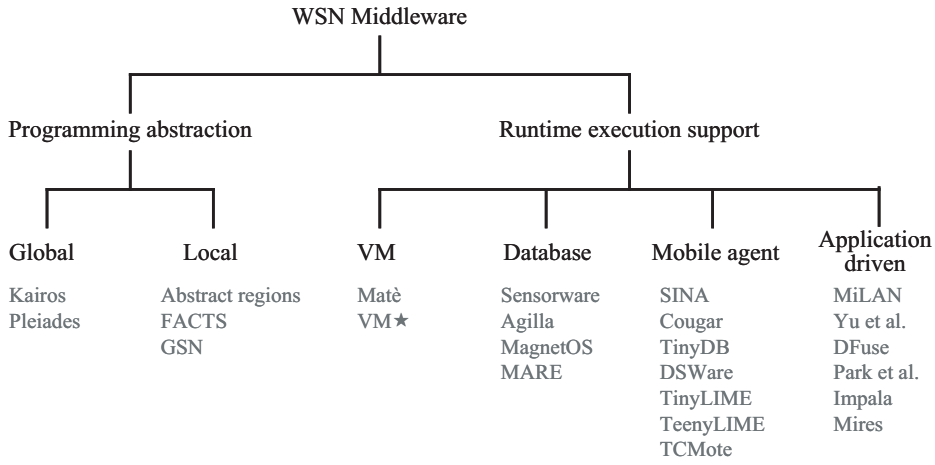


Fig. 6: Classification of WSN middleware [56, 59].

egorized to programming abstractions and runtime support depending on the main objective. The former creates an abstracted view of the network and sensor data, while the latter focuses on runtime systems and mechanisms for supporting application execution [56,59]. However, the classification is not unambiguous, since runtime environments may provide some sort of abstractions for WSN, and on the other hand, programming abstractions require runtime support for realization [59].

The programming abstractions are sub-categorized to global and local, depending whether the network behavior is viewed as a single entity or divided into local groups defined by criteria, such as data content, location, or network topology [56, 59]. Nevertheless, these should not be mixed up with programming languages, such as nesC [52] that are discussed in Chapter 4. While programming languages are abstractions of the capabilities of computer hardware, these are abstractions of network capabilities.

The runtime execution support is divided into four subcategories depending on the system architecture. Again, the classification to Virtual Machines (VMs), databases, mobile agents, and application-driven middleware may partly overlap. For example, mobile agents typically execute on top of VM. VMs abstract underlying hardware and software through a virtualized environment, while database middleware view WSN as a data storage that can be queried [153]. Mobile agents realize the mobile code paradigm, in which the code is moved to data origins for local processing instead of data [51]. Application-driven middleware control and allocate tasks to nodes based on the QoS requirements and guidelines set by the application [59].

3.2.3 WSN Middleware Proposals

The categorization of WSN middleware proposals is depicted in Fig. 6. A summary of their main features is presented in Table 4 and Table 5, classified according to the classes presented above. WSN node middleware presented in [P4] is included in Table 5. The columns assess the resource requirements of the middleware and how they satisfy the requirements defined in Section 3.2.1.

The second column of the tables defines the requirements set to the implementation platform. *Tiny* denotes a Mote-class device [68], *small* e.g. Intel XScale class devices, while *large* means Personal Computer (PC) or PDA type of devices. The heterogeneity abstraction defines the level of independency from underlying hardware and software, roughly categorized to *full*, *partial*, or *none*. The fourth column specifies a method for describing application tasks for middleware. The task allocation column defines whether application tasks are assigned to nodes statically at design time or dynamically, and whether the allocation is centralized or distributed. The aggregation column depicts the level of aggregation support, again roughly classified to full, partial, and none. The next one presents the middleware interface for external WSN data access. A term *custom interface* denotes a highly implementation dependent function or message interface. The last column lists other key features.

Virtual Machines

Matè [95] is a simple custom bytecode interpreter for very resource constrained nodes. Matè instructions express high-level WSN operations, which incurs to a very small code size. Small-sized Matè applications can be efficiently injected to WSN. Yet, these capsules do not possess features of mobile agents but instead disperse and infect the whole network [95].

VM$\star$ [91] implements a component-based subset of Java VM for resource constrained nodes. The runtime VM engine contains only the services needed by the application. Both, the application and the VM can be reconfigured with incremental updates.

Database Middleware

In database middleware, WSN data is accessed by queries that are served by distributed query processors [153]. Basically, Structured Query Language (SQL) data-

Table 4: Summary of VM, database, and mobile agent proposals for WSN middleware comparison.

Middleware	Target platform	Heterogeneity abstraction	Task formulation	Task allocation	Aggregation	External data access	Miscellaneous
Matè [95]	tiny	partial	none	none	none	code capsules	limited expressivity
VM⋆ [91]	tiny	full	Java class	none	none	Java class	configurable engine
SINA [142]	large	full	SQTL script	dynamic, distributed	partial	SQTL scripts	
Cougar [215]	tiny	partial	query plan	dynamic centralized	full	queries	
TinyDB [105]	tiny	partial	SQL type queries	dynamic, centralized	full	basic and event-based SQL type queries	
DSWare [97]	N/A	partial	none	dynamic, centralized	partial	SQL type queries, event detection	data caching
TinyLIME [33]	tiny	partial	data template	dynamic, centralized	none	tuple read	
TeenyLIME [29]	tiny	full	capability tuples	dynamic, distributed	none	tuple read	tuple space shared to one-hop neighbors
TCMote [35]	tiny	partial	attributes	dynamic, centralized	none	tuple get	
SensorWare [22]	small	full	TCL script	dynamic, distributed	none	TCL scripts	
Agilla [49]	tiny	partial	agent code	dynamic, distributed	none	mobile agent	local tuple space for data sharing
MagnetOS [14]	large	full	none	dynamic, distributed	none	none	single system image
MARE	large	full	task descriptor	dynamic, distributed	none	tuple access	geared for mobile environments

Table 5: Summary of application-driven middleware and programming abstraction proposals for WSNs.

Middleware	Target platform	Heterogeneity abstraction	Task formulation	Task allocation	Aggregation	External data access	Miscellaneous
WSN node middleware [P4]	tiny	partial	QoS levels	dynamic, distributed	none	custom interface	tuple space for task and QoS data sharing
MiLAN [65]	N/A	partial	state and QoS graphs	dynamic, distributed	none	custom interface	
Yu et al. [218]	N/A	partial	QoS specification	dynamic, distributed	none	custom interface	
DFuse [94]	large	partial	task graph, fusion function	dynamic, distributed	full	fusion API	
Park et al. [118]	small	partial	task graph	static, centralized	partial	custom interface	mobile agents for runtime migration
Impala [102]	large	partial	parameter table	dynamic, distributed	none	custom interface	supports runtime reprogramming
Mires [150]	tiny	partial	topic	none	partial	publish / subscribe	
Kairos [56]	small	full	macro-programming	static, centralized	none	remote data access interface	preprocessor code modification
Pleiades [92]	tiny	partial	macro-programming	dynamic, distributed	partial	root node interface	constructs for concurrent execution
Abstract regions [181]	tiny	full	region abstractions	none	full	none	
FACTS [167]	small	full	rule	none	partial	none	functions for low level optimization
GSN [1]	large	full	virtual sensor	dynamic, centralized	partial	SQL-type queries	

base queries identify the requested data. These may be supplemented with mechanisms for more complex in-network processing [105,215].

In SINA [142], database queries are injected to network as Sensor Query and Tasking Language (SQTL) [82] scripts. These scripts migrate from node to node depending on their parameters. The allocation of queries to individual nodes is implemented by an execution environment that compares SQTL script parameters to node attributes and executes script only if these match. The expressivity of SQTL scripts allows also more complex tasks, e.g. by timer utilization for execution triggering [82,142].

In Cougar [215], a query optimizer at the gateway node determines energy efficient query routes. Query plans generated by the query optimizer are parsed in the nodes by a query proxy. Based on the data flow and computation plan specifications defined in query plans, local query proxies make sensing, aggregation, and communication decisions [215].

TinyDB [105] takes a similar approach to Cougar. In TinyDB, a query processor in a node supports basic SQL type query operations and data aggregation for improving network energy efficiency. In addition, TinyDB supports event-based queries that are initiated in-network after the occurrence of a specified event is detected [105].

A similar event detection service to TinyDB is implemented by DSWare [97]. In addition, DSWare improves data availability by distributing frequently queried data to the network and increases robustness by grouping nodes with similar objectives for management [97].

TinyLIME [33] extends LIME middleware [109] to WSNs. In LIME, data and communication use tuple space [53] for sharing information. A LIME client can access sensor tuple in a tuple space through TinyLIME. A TinyLIME instance queries data from the node that serves the queried tuple. The scalability of TinyLIME is limited to a star topology [33].

While TinyLime relies heavily on LIME, TeenyLIME [29] implements the tuple space concepts in WSN environment. A tuple space is shared only among one-hop neighbors, thus each node has an own view of the tuple space. Each node can advertise its characteristics and sensing capabilities to the neighbors in specific tuples. These can be used for controlling and activating tasks within the network [29].

TCMote middleware [35] takes similar approach to that of TinyLIME, but does not extend the tuple space outside the WSN. The queries and predefined alerts utilize tuple channels for information exchange. Similarly to TinyLIME, The TCMote does not support multi-hop WSNs [35].

Mobile Agent Middleware

A mobile agent is an object that carries executable code, its internal state, and data [123]. A mobile agent makes its migration and processing decisions autonomously [51]. In order to obtain platform independency and relatively small object size, mobile agents are typically implemented on top of VMs [51].

In Sensorware [22], application tasks are implemented as Tool Command Language (TCL) scripts. A user requests for WSN data and services by injecting a script to the network. Each script contains task functionality and an algorithm that controls the script migration. The small size of TCL scripts allows their efficient migration [22].

Mobile agents in Agilla [49] are implemented with custom bytecode, quite similar to Matè [95]. High-level instructions result to small-sized agents, which migrate efficiently between nodes. The agents communicate through a local tuple space [53]. Reactions are used for notifying an agent of the events it is interested in [49].

In MagnetOS, [14] application tasks are implemented as Java objects. MagnetOS utilizes automatic object placements algorithms that aim to minimize network communication load by moving Java objects nearer to the data source. MagnetOS is implemented on top of a distributed VM [147] abstracting network resources as a single Java VM. Resource requirements make MagnetOS suitable only for high-end sensor nodes [59].

MARE middleware [155] is merely targeted to mobile ad-hoc networks, but its resource requirements are comparable to MagnetOS. In MARE mobile agents move towards data sources and perform communication locally. A tuple space [53] is used for resource discovery and communication between agents [155].

Application-driven Middleware

The application-driven middleware covers proposals that perform networking and execution control with the aid of guidelines initiated by an application [59]. Typically, the application awareness is realized by a task allocation functionality that distributes processing to nodes.

MiLAN [65] takes an application QoS specification as an input. The QoS specification contains e.g. the required accuracies and monitoring intervals for application tasks, and their dependencies on other tasks. During runtime, MiLAN adapts network operation to meet the application QoS and to lengthen network lifetime. Multiple applications are interleaved according to their importance. MiLAN is tightly integrated

to the underlying protocol stack, which enables networking adaptation according to the application needs [65].

Like MiLAN, a cluster-based middleware [218] uses application QoS specification for runtime control of network operations. The middleware manages underlying network topology by forming clusters and controls resources and task allocation in them. Tasks are allocated by a heuristic algorithm that attempts to minimize computation and communication energy costs [219].

An application task graph defines the guidelines for task and role assignments in DFuse [94]. Application tasks are implemented as fusion functions that are allocated to the nodes by a distributed algorithm. While the operation is initiated by a root node, the allocations are dynamically re-evaluated in order to find a more optimal configuration [94].

Application-driven task allocation is combined with mobile agents in [118]. Communication and dependency graphs are extracted from the application at design time. The graphs are modified for minimizing the communication costs by link and schedule optimization. During runtime, the execution is adapted by mobile agents [118].

The application adaptor of Impala [102] changes active tasks and the nodes executing them based on the application parameters. The changes are coordinated by a state machine describing application functions and the trigger conditions for state transitions. Application tasks can be updated during runtime by a centralized control entity [102].

Mires [150] is merely a MOM but its actions are controlled by the application. Mires uses a publish/subscribe architecture, in which nodes advertise their tasks (topics) and external user applications select desired tasks for execution [150].

Programming Abstractions

A global abstraction is also referred to as macroprogramming since it allows the programming of WSN as a whole and hides the individual nodes from the application. Local abstractions focus on the nature of sensed data within a local context [59].

Kairos [56] creates a global programming environment with node, neighbor, and data access abstractions. Nodes and neighbors are accessible independent of the underlying topology. A distributed program implemented with the abstractions is converted in compile time to individual instances executed in nodes on top of Kairos runtime.

The runtime environment manages global and shared variables and communication between program instances [56].

Pleiades [92] takes quite similar approach to Kairos. Applications are programmed for complete WSN, not for individual nodes. Pleiades language includes constructs for accessing the state of nodes and for supporting concurrent execution. A compiler generates code modules for individual nodes from the Pleiades programs. The execution of the modules within the network is controlled by a runtime system [92].

Abstract regions [181] are a communication abstraction among local groups of nodes. A runtime environment supports neighbor querying and data sharing that can be utilized for distributed application programming. The algorithms for region construction can be dynamically changed [181].

The main abstractions in FACTS [167] middleware are facts and rules. Facts abstract data representation and communication, while rules define the means for data processing. A rule is executed on top of an interpreter, a rule engine. The execution is fired by an event with a prerequisite that the conditions (e.g. availability of data) of the rule are satisfied. A function is an abstraction that can be used for accessing low level resources and for the efficient implementation of algorithms using machine code [167].

A local abstraction is provided by a GSN middleware [1] that allows the programming of virtual sensors using Extensible Markup Language (XML). Virtual sensors abstract both the sensor data access and communication from the application programmers. Runtime support is implemented by Java-based containers that host and manage virtual sensors [1].

There are few other programming abstractions, such as Protothreads [39] and TML [112], that convert the abstractions during compile time. However, since these do not include runtime support, they are not further discussed.

Summary of WSN Middleware

WSN middleware is still evolving and many proposals are early stage architectural and algorithmic explorations without an existing implementation on node platforms [59]. Further, the resources of target hardware, objectives, and assumptions vary considerably. Therefore, a fair comparison of WSN middleware is difficult.

A framework for middleware evaluation is proposed in [59]. In this Thesis, the summary presented in Table 4 and in Table 5 follows the main requirements presented in

Section 3.2.1. Thus, the tables create the basis for comparison. In general, middleware support is mainly targeted to applications, thus they do not facilitate protocol development. This is not entirely true for programming abstractions, which extend also to the lower layers [56, 181].

The suitability of the different approaches depend on the applications and resources of the nodes. Database middleware suit to static networks with structured data queries. VMs and mobile agents are beneficial in applications that require task mobility. The programming abstractions facilitate the description of application functionality, but their support for runtime actions is more restricted [59].

While these approaches create suitable abstractions for WSNs and integrate the network level awareness to the operation of a single node, they lack common, lightweight methods for formulating and allocating application tasks in dynamic networks. This is best solved by application-driven middleware that steers network and single node operation depending on the surrounding conditions and application requirements.

Compared to the WSN node middleware presented in this Thesis, other application-driven middleware are the closest. The approaches taken in MiLAN and in [218] are quite similar to that of WSN node middleware. Compared to these, the tuple space and a lightweight task allocation algorithm of WSN node middleware are applicable for tiny WSN nodes. Further, unlike the task allocation of [218], which assumes homogeneous node resources and communication links [219], WSN node middleware adapts to dynamic operating conditions typical for WSNs [157].

4. DESIGN OF WSNS

This chapter outlines the concepts and related work on methodologies and tools that facilitate the WSN design. First, a generic design flow, which defines the phases that need to be considered in WSN design process, is introduced. Then, different MoCs for abstract system description are discussed shortly. Finally, the related work in system level design is discussed, focusing on the design methodologies and tools.

4.1 WSN Design Flow

A design flow specifies the series of steps for system implementation. Typically some steps involve Computer Aided Design (CAD) tools [187]. A related concept is design methodology, which is a method of proceeding through the levels of abstraction in order to complete a design [187].

A widely adopted design approach in embedded systems is HW/SW co-design flow [187]. The main phases in the flow are depicted on the left hand side of Fig. 7. Based on the requirements, system architecture is defined by identifying the main components. After that, hardware and software components are designed relatively independently. Last two steps are system integration and testing. A design phase itself can be divided to smaller, similar design cycles [187].

In embedded system domain, several tools are available for managing the design flow from abstract modeling to the final implementation [84]. The fundamentals behind the methodologies can be generalized also for WSNs but the methods and abstractions need to be specialized [21, 135]. While the details of the underlying network architecture should be abstracted from a designer, it is essential that a WSN design methodology considers the differences in paradigms, resource constraints, and unpredictability throughout the design flow [11, 21].

The unique characteristics of WSNs need to be taken account especially during the design phase. First, the existing hardware and software components are configurable

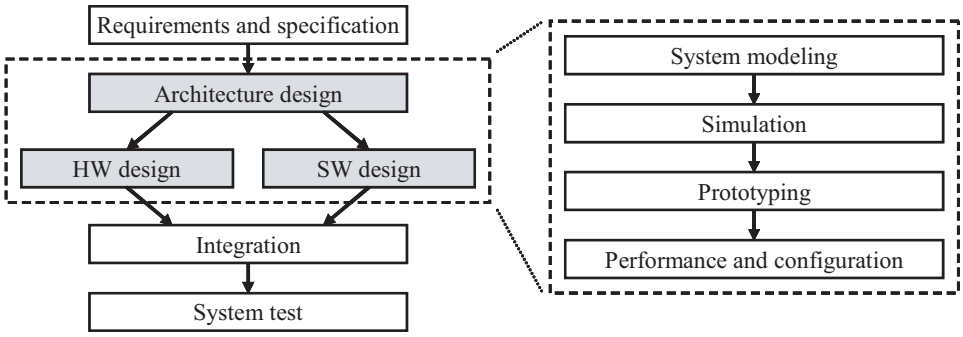


Fig. 7: A HW/SW co-design flow and the WSN design flow used in this Thesis for the comparison of design methodologies.

by their nature [21, 143]. Therefore, a component reuse with a tailored configuration is a notable option. Second, potentially very large scale and random operating conditions make full system prototyping and testing infeasible [100, 121, 162, P1]. Therefore, more scalable verification methods are required, preferably earlier in the design flow.

The WSN design flow used for the comparison of existing design methodologies is presented in Fig. 7. In general, the flow adapts the HW/SW co-design methodology [187], but the design phase considers the above mentioned WSN specific characteristics. Right hand side of the figure shows the phases of the system level design. The detailed design of hardware and software follows the co-design flow.

Phases of the WSN design flow are introduced below. The design flow considers only the actual sensor network, not for example backbone network access or support infrastructure. In the explanation, a term designer refers to one or more persons who are responsible of the technical specification, design, implementation, and testing of the system.

Requirements and specification: The first step in the design flow is a requirement analysis that determines the basic characteristics of the system [187]. The requirements can be formalized with the design factors discussed in Section 2.4.

System modeling: In this phase, a designer models system functionality with a set of abstractions that allow implementation independent system description [21]. The system modeling phase includes the description of both software

functionality and hardware capabilities. The abstractions enable scalability between different applications and node platforms [21], and hide the implementation details from application designers [11].

Simulation: The simulation phase analyzes the suitability of models for the application scenario [146]. With simulations, the applicability and performance of modeled algorithms and components can be evaluated rapidly with a large number of nodes and in a long-term operation. Further, in a simulation environment, the operation of the system can be monitored and controlled easily [41]. The simulations are run either with abstract models or a low-level implementation.

Prototyping: Due to the unpredictable environment and severely constrained node platforms, simulations cannot realistically mimic all features of WSN deployments. Therefore, prototyping with a real hardware in a realistic environment is crucial for system assessment [41]. This phase may discover several new aspects that were not visible in simulation phase. These can be for example radio interference or unpredictable errors due to resource constraints, such as stack overflows or out-of-memory conditions [41].

Performance and configuration: In this phase, the obtained performance results from the simulation and prototyping phases are assessed and evaluated against the non-functional requirements of the application. If results are not acceptable, model configurations need to be altered, or components replaced or redesigned. Otherwise, the design produces a configuration for deployment with realistic performance estimates.

Integration: In the WSN design, integration phase combines the different components to a full feature system, i.e. a deployment. In this phase, tasks are mapped to the deployed nodes.

System testing: Even though a comprehensive testing of a full feature WSN deployment is challenging and infeasible [41, 162], a set of tests for validating network operation need to be conducted. Due to the complexity involved, the number of tests during this phase should be minimized. Testing can be alleviated by automated tools for network diagnostics gathering and analyzing [157].

4.2 Models of Computation

In the system modeling phase, the functionality of the system is described by abstract models. In general, a model can be defined as a simplification of the modeled entity. The model describes only those characteristics and properties of the entity that are essential for the design [83].

In order to serve a design task, a model should incorporate an appropriate level of detail. Thus, it should be simple enough to allow e.g. fast simulation of the system, but accurate enough to provide required results about the system operation. In system design, the rules and constraints for the description of models are defined by the used MoC [83, 84].

A MoC defines the computation, communication, synchronization, and the relative timing of concurrent processes comprising a system. The relative importance of the different aspects varies depending on the modeled system. This variance is also visible in different MoCs, some of which focus mainly on describing the computation while others concentrate on the communication and synchronization of concurrent processes [83].

Several different kinds of MoCs have been developed for system design. The most common ones are Finite State Machines (FSMs), Petri Nets, and Kahn process networks. FSMs are best suited for modeling the computation the system, whereas Petri Nets focus on the communication and concurrency of the system. Kahn process networks combine these by connecting state-aware processes through unbounded First In First Out (FIFO) channels. Each of these MoCs have many variants and descendants that can be considered as separate MoCs [83, 84].

4.2.1 Finite State Machines

The design methodology presented in this Thesis utilizes an FSM MoC that is well-suited for describing communication protocol functionality [72, 176]. FSMs are a simplification of a Turing machine. An FSM consist of a finite number of states and transitions between the states. Each state transition is triggered by an input event. The output of the system can either be associated to states or to state transitions. An FSM can be deterministic, when a same input in a certain state causes always the same transition, or nondeterministic, in which case there can several transitions from a state with same input, and the realized one can be any of these [83].

In their basic form, FSMs are quite limited in their expressiveness. The limitations

have been rectified by different variants that improve data processing capability and hierarchy of FSMs. Examples of these are FSMs with datapath, statecharts, and Extended Finite State Machines (EFSMs). The first one annotates state transitions with arithmetic operations in order to simplify the representation of data processing and conditional execution. Statecharts are based on the hierarchy of concurrent FSMs that run synchronously in a lockstep mode. This means that outputs of an FSM are visible in the inputs of other FSMs at the next time cycle. EFSMs are essentially statecharts with two main differences. First, they are asynchronous and second, they support more advanced data processing and storing through variables [83, 84].

4.3 Related Research on WSN Design

Due to the iterative nature of the design phase and the challenges involved in the final deployment, several tools and methodologies have been proposed for facilitating WSN design and evaluation. A major challenge for these tools is the integration of the design phases from abstract modeling to a low level implementation on node platforms.

Due to the immaturity of WSN technology and the lack of established practices there is no common basis for the categorization of design tools and methodologies. The methodologies vary in target objectives and domains as well as in taken approaches. Further, quantitative metrics for the comparison of design methodologies cannot be easily derived, since some of the main objectives, such as easiness and expressivity, depend merely on the subjective experiences of a designer [84].

The boundary between design and runtime environments targeted to WSN domain is not always evident. This section discusses the efforts targeting to the system level design and evaluation of WSNs. More specifically, the focus is on tools and methodologies that aid the development of complete WSN deployments, and on simulation environments.

4.3.1 WSN Design Methodologies

In WSN domain, research considering structured design approaches is still evolving. In system level issues, the research has merely focused on runtime environments discussed closely in Chapter 3.

The proposals discussed in this section cover different phases in WSN design flow. In general, they provide abstractions, methods, and tools that aid a designer to manage

the process. Fundamental issues, such as programming paradigms and approaches for data extraction and network control are not discussed [32,54,188].

Main characteristics of existing design methodologies are outlined in Table 6. In general, approaches can be categorized to model-driven, component-based, and platform-based design methodologies. Integrated development environments are considered as a separate subclass. The WISENES design environment presented in Chapter 5 and in [P1,P6] is included in the table for comparison. The dimensions of the table assess the tool support in different phases of the WSN design flow, and highlight the scope and the main properties of the methodologies in order to facilitate their comparison.

The first column of the table identifies the methodology and the second defines the main approach. The next one lists the design flow phases that are supported. Prototyping denotes the implementation on physical node platforms. The abstraction defines the interfaces and models that hide the low-level implementation details. Components considered by the design methodology are listed in the fifth column. Possible components are application (app), communication protocols (comm), and node platforms (node). The evaluation column lists a method or methods that are used for assessing the selected design. The last two columns indicate, whether the given property is supported by the design methodology. Design space exploration means either automated or manual configuration optimization for the given deployment.

Model-driven Design

A model-driven design uses domain-specific, tailored models and abstractions for aiding the design process [138]. In this context, the model-driven design methodologies include those proposals that support WSN design with existing models or metamodels.

In [101], internals of WSN operation are abstracted by collaboration groups, which are collections of nodes or more abstract entities, agents. An application is designed on top of this abstraction using a state-centric programming model. Application performance can be estimated with high level simulations.

Collision free and aware models are used for abstracting communication in [217]. The methodology supports analytical evaluation of application algorithms on top of communication models. The exploration of suitable network parameters is performed manually [217].

Tinker [42] does not directly fall into the category of model-driven design methodologies but this is the closest in the used classification. Tinker focuses on application

Table 6: Comparison of WSN design methodologies.

Methodology	Approach	Covered phases	Abstraction	Components	Evaluation	Graphical design	Design space exploration
WISENES [P1,P6]	model-driven	design, evaluation, prototyping	deployment models, EFSM	app, comm, node	simulation	SDL	manual
Liu et al. [101]	model-driven	design, evaluation	collaboration group, state	app	simulation	none	none
Yu et al. [217]	model-driven	design	communication model	app	analytical	none	manual
Tinker [42]	data-centric	design	application-level data streams	app	simulation	none	manual
nesC [52]	component-based	prototyping	component interfaces	app, comm	code analysis and optimization	none	none
GRATIS [175]	component-based	design, evaluation	component model	app, comm	component validation	component assembly	none
Viptos [25]	component-based	design, evaluation, prototyping	ptolemy II MoCs	app, comm, node	simulation	ptolemy II	manual
Bonivento et al. [21]	platform-based	design, exploration	interface abstractions	app, comm, node	none	none	automatic
Bakshi et al. [11]	platform-based	design	virtual architecture	app	none	none	manual
Shen et al. [143]	platform-based	design, evaluation, prototyping	platform models	app, node	model-based simulation	none	automatic
Emstar [41]	integrated development	evaluation, prototyping	runtime environment	app, comm	simulation	none	manual
Worldens [50]	integrated development	evaluation	none	app, comm, node	simulation	none	manual

data and it abstracts communication and node platforms by few simple loss models. The main objective of the tool is the exploration of different data processing algorithms for input data streams [42].

Component-based Design

In WSNs, component-based design is motivated by TinyOS architecture [69]. The main abstraction in design process is a software component implementing a dedicated function. A component is accessed through a well-defined interface [52]. A reuse of existing designs is directly enabled by the component-based architecture, assuming that the component interfaces and specifications are standardized [9, 28].

All existing proposals in this category exploit or complete TinyOS. A key technology is nesC [52] programming language, which extends C language with built-in component model as well as concurrency and communication support. While characterized as a language, nesC aims to holistic system design [52].

In the generic WSN design flow, nesC language addresses only the prototyping phase. A graphical interface for component design is provided by GRATIS [175] and Viptos [25]. GRATIS offers a graphical interface for assembling the components to a complete system, and generates runtime code for connecting the components automatically [175].

Viptos is built on top of ptolemy II design environment [122] and it supports all phases in the design flow. Viptos integrates the MoCs available in ptolemy II to nesC, TinyOS, and TOSSIM [96, P1]. Viptos complements the functionality of VisualSense simulation environment [12, P1]. Graphical designs can be simulated with VisualSense, or generated as executables for TOSSIM simulations or TinyOS nodes [25].

Platform-based Design

In platform-based design, an application specification is refined on top of the abstraction of potential implementation platforms [21, 84, 86]. Typically, this methodology includes an automatic exploration of platform configurations for the given application [84]. In WSN context, the platform includes also communication protocols in addition to physical node parameters [21].

In [21], three abstraction layers for applications, protocols, and node platforms separate the design to distinct models. The key concept is the exploration of the protocol

parameters so that application requirements are met in constraints set by node platforms. The requirements are extracted with a Rialto tool that analyzes all possible communication combinations of the application [21].

Algorithm design method proposed in [11] possesses similar characteristics. Node platform and communication are modeled coarsely by a virtual architecture, on top of which application algorithms can be synthesized to actual programs allocated to the nodes. Each phase is performed manually [11].

Models for rapid prototyping of WSNs are introduced in [143]. In a design phase, the methodology abstracts nodes and networking using a set of energy models and connectivity graphs. The design space exploration to obtain mainly hardware parameters is based on the simulation of applications with the energy models. Prototyping is used for validating the accuracy of evaluation [143].

Integrated Environments

The development environments covered in this section cannot be considered as design methodologies, since they do not offer any abstractions for system design. Instead, they provide tools for supporting different phases of the design process.

Emstar [41] is a Linux-based development environment for protocol and application software development. The main component in Emstar is a runtime environment, which runs on diverse execution platforms. This allows evaluation of the final executables with simulations, prototyping, or their combination [41].

A quite similar approach is taken in Worldsens [50]. Compared to Emstar, Worldsens incorporates two simulators, one for large scale network simulations and other for cycle accurate simulation of a single node. Both simulators run native target platform code. Further, the simulators can be synchronized for cycle accurate large scale network simulation [50].

Summary of WSN Design Methodologies

The comparison of design methodologies is difficult because of the lack of unambiguous metrics, and contradictory goals and approaches. The parameters defined in Table 6 set a starting point for qualitative comparison of WSN design environments.

Most of the methodologies support abstract design phase, but only few cover also evaluation and prototyping. There are divergent opinions, whether there should be

a separate high-abstraction level design, and a low-level implementation phase [50]. While separate environments make the creation of higher level abstractions easier, the porting of the models to the final implementation may introduce additional bugs [173].

In summary, model-driven and component-based design methodologies provide suitable abstractions for WSN design. Platform-based design methodologies support automatic exploration of different network or node platform configurations. While integrated development environment introduce a rich set of tools for prototyping phase, they lack design abstractions.

In WSNs, the most fundamental design choices need to be made before a large scale deployment. Thus, the evaluation in the early phase of the design flow has a significant importance in the process. If its results are reasonably accurate, the design choices can be based on a solid foundation.

From the related design methodologies, Viptos [25] incorporates quite similar features compared to WISENES. Both enable a graphical design of models, and the evaluation of the models through simulations. Viptos allows WSN design with several different MoCs and it has an integrated support for code generation on top of TinyOS [69], while in WISENES the EFSM models need to be ported manually. In comparison to other methodologies and tools, the main benefits of WISENES are the accuracy design time performance evaluation and the back-annotation of measured information from prototypes for further refining the result accuracy [P1].

4.3.2 WSN Simulation Tools

The principles and requirements for WSN simulation are thoroughly discussed in [P1]. Further, related research on the area is presented and analyzed in detail in the publication. Yet, some new WSN simulation environments have been published. These include three networking oriented simulators, namely SenQ [173], WSNet [50], and GTSNetS [117], as well as three sensor node simulators, Avrora [169], VMNet [210], and DiSenS [182].

SenQ [173] is a descendant of sQualnet [151] reviewed in [P1]. Compared to sQualnet, SenQ has a more accurate node model that take time drifting of internal node oscillators into account. SenQ emulates over 1 000-node networks with final executables developed as SOS [61] operating system threads. In SenQ, it is also possible to simulate WSN protocols parallel with IP-based nodes [173].

WSNet [50] is a discrete event simulator with a custom engine. The simulations scale to more than 3 000 nodes. The transmission medium is modeled in detail, but sensing is not considered. The node model incorporates a linear battery model. WSNet is parameterized with a set of configuration files that define radio and node models [189]. During runtime, the simulations can be monitored through a Graphical User Interface (GUI) and events are logged to trace files [189]. WSNet can be used in conjunction with Worldsens [50] development environment for simulating code that is directly applicable for final implementation.

GTSNetS [117] is built on top of GTNetS [127] simulator. With sparse network topologies, the simulator scales up to 100 000-node networks. Incorporated sensing model is accurate, while radio and power models are simple. The simulator visualizes execution through a GUI and outputs accurate log files. Simulated code is not applicable for sensor nodes [117].

Avrora [169] is a scalable cycle accurate sensor node simulator, that can simulate up to 10 000-node networks. Avrora contains a simple models for transmission medium and sensing, and optionally also an accurate energy model. Avrora is parameterized by defining possibly multiple simulated object files during startup [10]. Output is controlled by instrumenting the simulated system with function-specific monitors [10]. As typical for sensor node simulators, simulated code is directly applicable for final implementation [169].

Similarly, VMNet [210] is a cycle-accurate simulator that scales approximately to 500-node networks. The transmission medium, node, and power models of the simulator are realistic. A sensing model reads sensor inputs from files. Simulations are configured by input files and output results are stored to logs. Executables simulated in VMNet are directly applicable for final implementation [210].

DiSenS [182] is also a cycle-accurate simulator that executes binaries directly applicable for different types of sensor nodes. DiSenS support simulations of thousands of nodes, and due to the distributed simulation engine, the scalability can be increased by adding more workstations. Simple sensing, radio, and power models are implemented as plug-in modules [182].

In comparison to WISENES, the networking oriented simulators contain quite similar features. Even though the reported scalability of GTSNetS is an order of magnitude larger than that of other simulators, the sparse topologies with which the results are obtained are not realistic in typical WSN scenarios. Compared to other simulators, the distinct features of WISENES are the possibility to simulate virtually any hardware platform still maintaining the realistic model of node resources, the accuracy of

design time simulation results, and the integration to a complete design flow.

5. DESIGN AND IMPLEMENTATION OF WSNS WITH WISENES

This chapter presents the design of application-specific WSNs using WISENES and summarizes the results of this Thesis. The details of concepts, implementations, and results presented in this chapter are given in publications [P1-P6]. The chapter focuses on the proposed tools and methodologies for the design and prototyping of WSN protocols and applications. Two application scenarios are presented for highlighting challenges involved and for illustrating the phases in the application and protocol design.

The organization of the chapter follows the order of publications. First, WISENES design environment is presented, including the design flow, models, and results. Systems software for prototyping in WISENES design flow is provided by SensorOS and WSN node middleware. Their implementations and results are discussed prior to the application scenarios.

5.1 WISENES Design Environment

WISENES is targeted to the system level design of fully functional large scale WSN deployments. WISENES defines a methodology and models for the design, and a framework for design time performance evaluation through simulations. WISENES framework is presented in [P1] and design abstractions in [P6].

WISENES design flow, depicted in Fig. 8, specifies the steps for the realization of an application-specific WSN. First, a designer defines deployment models that describe key components and their functionality. These models are simulated in WISENES framework in order to obtain an accurate estimation of the network performance with a given configuration. The configuration is optimized manually until an acceptable one is found. The simulation phase is followed by prototyping with node platforms. The prototyping is facilitated by SensorOS and WSN node middleware that implement similar abstractions for deployment models. The accuracy of design time model simulations is improved by back-annotating performance results from the prototypes

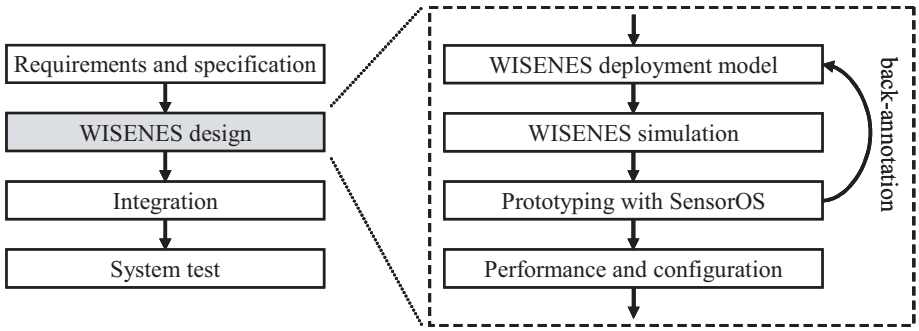


Fig. 8: WISENES design flow and the phases in WISENES design.

to the WISENES framework. Again, required phases are iterated until a configuration meets application requirements.

5.1.1 WISENES Deployment Models

WISENES deployment models abstract the main functional components in WSN design. A designer defines these abstractions, which are *application model*, *communication model*, *node model*, and *environment model*. The relations and main properties of the models are depicted in Fig. 9.

An environment model describes the characteristics of the environment that surrounds WSN deployment. A node model incorporates physical node capabilities and programming and hardware access interfaces. Thus, it models also OS and middleware functionality. A communication model defines the protocol stack and its configuration. The core application functionality is specified by the application model.

5.1.2 WISENES Framework

WISENES framework implements the design environment by supporting deployment model abstractions. MoC used for describing applications and protocols in WISENES is EFSM. Currently, the MoC is implemented using Specification and Description Language (SDL) [140]. WISENES tool uses a commercial Telelogic TAU SDL Suite [166] environment that allows graphical SDL design and automatic code generation for simulations. The executables generated from WISENES are too resource hungry for the most constrained nodes [S1]. The resource effectiveness can be improved by tightly integrating SDL models to SensorOS [176].

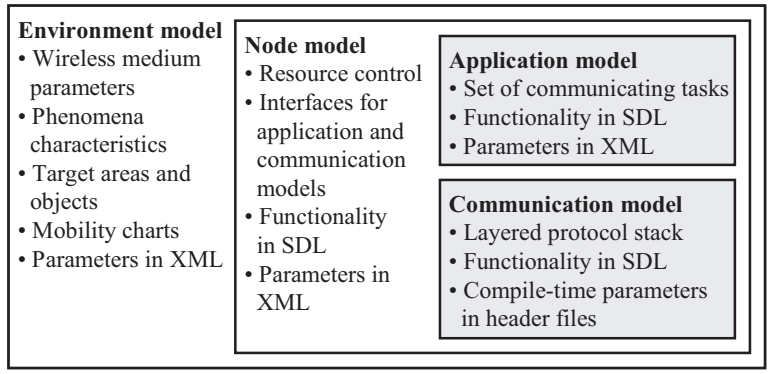


Fig. 9: Overview and main features of WISENES deployment models.

The functionality of protocols and applications is defined in EFSMs that are implemented as SDL processes and procedures. A detailed introduction to SDL and WISENES environment description for protocol design are presented in [P1]. For simulations, WISENES framework and deployment models are parameterized in a set of XML configuration files. The input parameters and output results generated from WISENES simulations are depicted in Fig. 10. During runtime, the progress of simulations can be monitored through a GUI, while detailed networking and data events are stored to log files for postprocessing.

5.1.3 Existing Protocol Designs in WISENES

WISENES design capabilities are demonstrated by two full-feature protocol stack implementations, TUTWSN and ZigBee . Protocol functionality is described in SDL processes that are grouped to SDL packages according to the protocol layers. Such a modular approach makes it possible to exchange a protocol layer by only adapting its interfaces.

TUTWSN

The WISENES implementation of TUTWSN protocol stack consists of three layers. A Time Division Multiple Access (TDMA) MAC protocol is implemented at the data link layer and cost-aware routing at the network layer. A middleware layer has two alternative designs, a request-based TUTWSN API [S3] and the WSN node middleware, described in [P4].

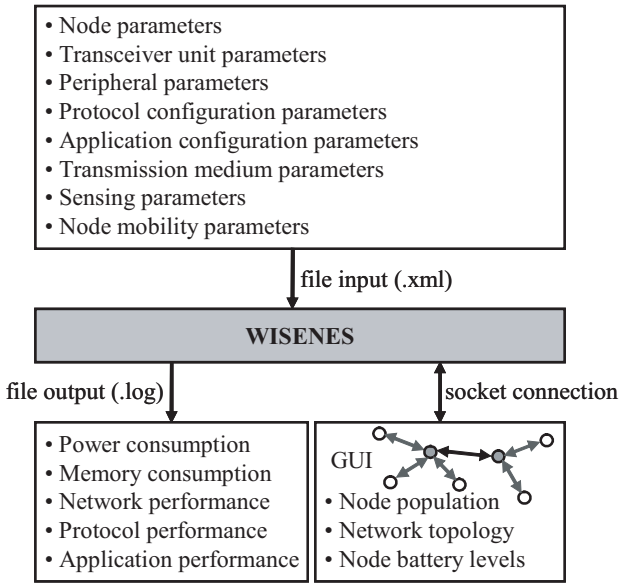


Fig. 10: WISENES input parameters and output results.

The main design objectives of TUTWSN MAC are energy efficiency, scalability, and autonomous operation [88,157]. These are realized by a clustered topology, which is maintained by tightly synchronized TDMA operation. The scalability and application QoS are supported by distributed algorithms for channel and time slot allocation, neighbor discovery, and link capacity optimization [S6].

TUTWSN routing protocol creates and maintains multi-hop paths to one or more sink nodes. Each node selects several alternative routes according to routing costs. The costs are calculated using cost functions that are weighted with traffic class dependent parameters. Route maintenance overhead is minimized by embedding cost updates to MAC layer network maintenance signaling [S4].

ZigBee

At the data link layer of WISENES ZigBee stack, the IEEE 802.15.4 LR-WPAN MAC protocol supports beacon-enabled mode [77]. At the network layer, WISENES ZigBee stack implements a cluster-tree topology with hierarchical addressing and routing [223]. The ZigBee stack implementation incorporates a simple middleware that adapts the stack to the WISENES application model. In addition to [P1], more implementation details and simulation results about the ZigBee stack in WISENES

Table 7: Statistics of WISENES SDL implementations for TUTWSN and ZigBee.

SDL Process	Decl.	States	Tra.	Proc.	Sym.	Stat.	Exec.
TUTWSN MAC							
Channel access	1122	20	194	129	2079	3141	3072
Management entity	195	9	32	13	297	381	130
TUTWSN routing							
Route management	479	10	81	43	838	1290	1276
Data service	158	3	41	21	314	373	125
TUTWSN API							
API service	150	4	25	10	180	283	71
IEEE 802.15.4 LR-WPAN MAC							
Channel access	834	34	163	75	2202	2732	2205
ZigBee NWK layer							
NWK management entity	327	12	55	30	574	899	366
NWK data entity	277	4	47	26	393	519	184
Adaptation middleware							
Interface adaptor	72	4	20	4	167	185	105

can be found from [S5].

5.1.4 WISENES Design Results

The quality of a design methodology and abstraction are subjective matters. The comparison of WISENES and related design methodologies is presented in Section 4.3.1. As shown in [P1] and [P6], environment and node model parameters are defined in structured XML configuration files. The design of a communication model consisting of a complete WSN protocol stack is straightforward due to the hierarchical structure of the SDL and modularity of the WISENES framework. Further, graphical design with EFSMs suits well for protocol modeling [P1]. Event-driven application model design introduced in [P6] conforms to the basic nature of the WSN [69, 85].

The feasibility of WISENES design environment is proved in [P1]. The author of this Thesis implemented the IEEE 802.15.4 LR-WPAN MAC protocol within two working weeks. In spite of the familiarity of WISENES and protocol internals, the development cycle was considerably shorter than projected.

In order to visualize the complexity of WSN protocols, the statistics of the current TUTWSN and ZigBee implementations in WISENES are presented in Table 7. The statistics are gathered by a complexity measurement tool incorporated in Telelogic TAU SDL Suite [166]. SDL related terms are further explained in [P1].

The first column, declarations (*Decl.*), gives the count of all procedure, data type, variable, and other entity declarations in the SDL process. The following two columns present the number of distinct *states* and transitions (*Tra.*), which start from a state on a reception of an input signal and end to another state. The following column gives the number of procedures (*Proc.*) that are used for hierarchical design. The sixth column depicts the number of graphical symbols (*Sym.*), which can be e.g. tasks containing several statements, procedure calls, or output signals. The following column defines the count of statements (*Stat.*) in the SDL process. A statement is for example an assignment within a symbol. The number of execution paths (*Exec.*) gives a good estimation of the process complexity by specifying a close approximation of the alternative paths from the start of a transition to its end.

The number of states, procedures, and statements in Table 7 shows the extend of the protocol designs. As indicated by the figures, especially the channel access protocols are complex consisting of a number of states and hierarchical SDL procedures. Yet, as the selected MoC suits well to the design of communication protocols, the implementations are well structured and can be easily comprehended.

5.1.5 WISENES Framework Results

In WISENES, the performance evaluation of WSNs during the design time is allowed by the simulation of deployment models. Detailed node model parameters define the capabilities and runtime characteristics of node platforms. In simulations, these capabilities are emulated accurately in WISENES framework. In addition, transmission medium parameters in the environment model are derived from measured signal attenuation graphs. These create a realistic basis for the simulations.

WISENES scales well to the simulation of thousands of nodes. Largest WISENES simulations have been 10 000-node networks [S1]. In order to assess the accuracy of the WISENES output results, a simulated power consumption is compared with the measured results from prototypes in Fig. 11. The average difference between measured and simulated results is 6.7 %. The difference results from small absolute values with subnodes and from the slight discrepancies in timing modeling. The simulated models are further refined by back-annotating prototype results to WISENES through deployment models and framework services. In the back-annotation, static power consumptions and execution related benchmark information are manually set to the configuration files defining deployment models and given as input parameters to WISENES framework routines that emulate low-level execution.

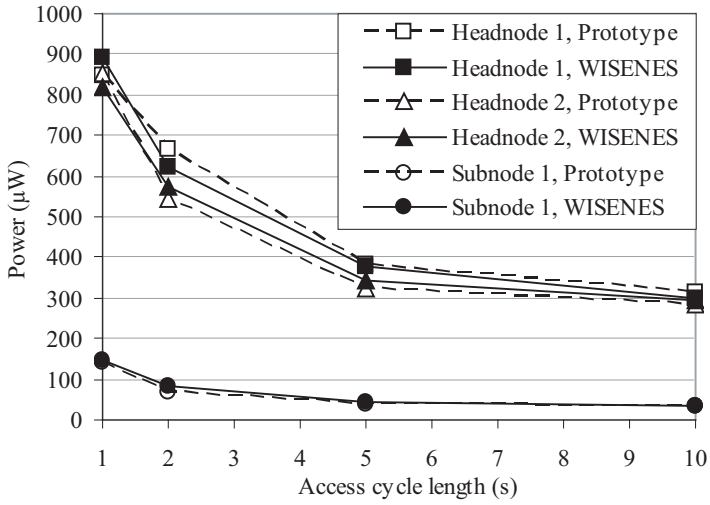


Fig. 11: Simulated WISENES power consumption results in comparison to measured prototype power consumption.

Compared to [P1], a more advanced and dynamic version of TUTWSN stack is simulated in [P6]. With this version, the difference between measured and simulated power consumptions is 6.6 %. The accuracy of networking delay is 9.5 %, in average. Since TUTWSN MAC and routing protocols adapt the network topology and routes dynamically, simulated and prototyped network configurations are not completely identical.

In addition to [P1] and [P6], simulation results about the performance of WISENES and existing protocol designs are presented in [S1,S5]. Common results show node power consumption, data throughput and delay, and network lifetime. Moreover, protocol designs can be freely instrumented to obtain desired results.

The key benefit of WISENES compared to the other WSN simulators is that the evaluation of the network and application performance is carried out during the high abstraction level design phase. While this is possible also in VisualSense, its simulation results are significantly less accurate than those of WISENES due to the more inaccurate modeling of execution and physical environment. Moreover, in WISENES prototyping results are back-annotated to the simulator to further improve simulator accuracy.

5.2 Systems Software for Prototyping

For prototyping with physical node platforms, the design time abstractions are implemented by systems software. The systems software support for distributed processing in WSNs is reviewed in [P2]. The publication focuses on runtime abstraction of application distribution by assessing services for service discovery, task allocation, remote task communication, and task migration.

As a conclusion, a need for OS and middleware support in WSNs is emphasized. The recommended architecture consists of a preemptive multithreading OS and a middleware performing network level task allocation and control. For WISENES design flow these services are implemented by SensorOS and WSN node middleware.

5.2.1 SensorOS

SensorOS implements a runtime environment for WSN protocols and applications. It manages resource usage, timeliness, and coexistence of multiple tasks on a single node, and offers interfaces for hardware access and OS services. SensorOS design and implementation on TUTWSN node platforms is presented in [P3].

The main objectives for SensorOS are low resource consumption and accurate time concept for implementation of time critical protocols and applications. The approach taken by SensorOS is a preemptive multithreading kernel with a priority-based scheduler. A traditional programming model with a POSIX-like API is easy to use and suits to the modeling of different kinds of design abstractions. The message-passing Inter-Process Communication (IPC) and centralized event service make it possible to adapt a state machine based WISENES application and communication models on top of SensorOS. Further, SensorOS services are modeled in WISENES, which eases the integration of design time models to the final implementation.

SensorOS Architecture

Both application tasks and WSN protocols are implemented as SensorOS threads. The SensorOS architecture with main OS components is depicted in Fig. 12. OS kernel provides services for message-passing IPC, thread scheduling, timer management, synchronization through mutexes, and memory and power management. Interrupt-driven device drivers are integrated to the kernel while other peripherals are controlled directly by the accessing thread.

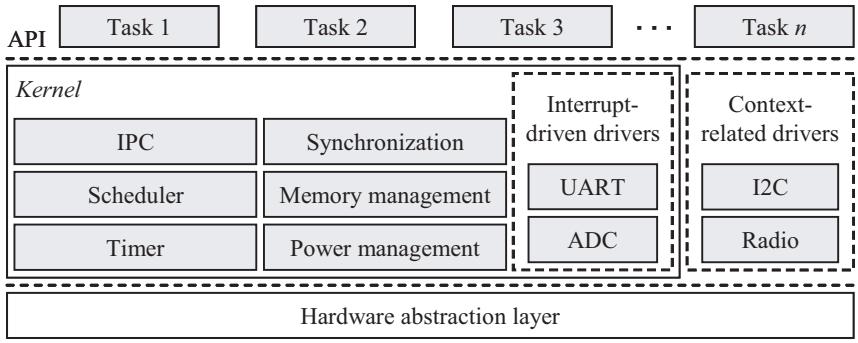


Fig. 12: Overview of SensorOS architecture.

SensorOS Results

The feasibility of SensorOS API is shown in [P6], which presents a code for an application thread that implements a motion detection task. The API is easy to use and it preserves WISENES design abstractions. The comparison of SensorOS to the related OSs is given in Section 3.1.3.

The performance of SensorOS is evaluated in [P3]. Excluding the strictly necessary low-level Hardware Abstraction Layer (HAL) operations for context switching and hardware register access, SensorOS is implemented in C. Compared to strictly optimized assembly implementation, this may impose some performance drawbacks but improves portability. On a TUTWSN PIC node, SensorOS requires at minimum 6 964 B of code memory. With a sophisticated memory management and an I/O library, the code memory usage is 8 586 B. Depending on the configuration, SensorOS kernel requires 115–134 B of data memory. A per thread stack takes typically 128 B. Since the size of a dynamic memory pool is configurable, SensorOS can host over 20 threads simultaneously on a TUTWSN PIC node.

The execution times of kernel functions are presented in [P3]. The variances in operation times are quite minor, which is important for a time-sensitive OSs. SensorOS implements an accurate wait service for high priority threads. The mean error for the deadlines of the wait service is below 2 μs and the maximum error less than 5 μs when compared to the wall clock time.

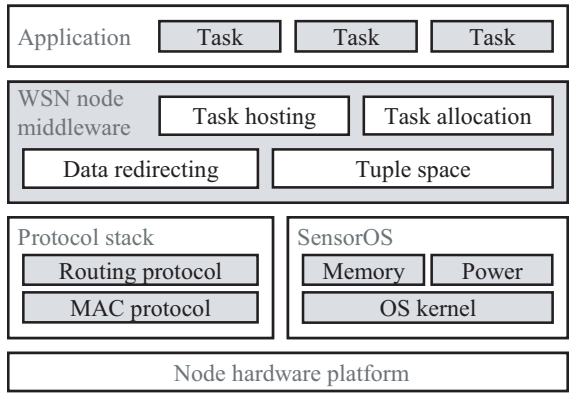


Fig. 13: Overview of WSN node middleware architecture.

5.2.2 WSN Node Middleware

WSN node middleware presented in [P4] extends the single node environment created by SensorOS with a network level control. The middleware creates an abstraction, on top of which application tasks can communicate without knowledge of the physical nodes hosting them.

WSN node middleware aims in maximizing the network lifetime by allocating application tasks and network topology maintenance roles for nodes. The allocation is coordinated within the boundaries set by virtual clusters that are directly mapped to underlying network topology or freely formed. Middleware actions are coordinated by an application QoS specification that defines the requirements and relations of tasks.

Middleware Architecture

The architecture of WSN node middleware is illustrated in Fig. 13. The middleware sits on top of SensorOS and WSN protocol stack. The middleware functionality is implemented by four components that are task allocation, task hosting, tuple space, and data redirecting.

Task allocation uses a lightweight algorithm for finding a set of nodes for application task execution and for electing the controller of a virtual cluster. Task hosting manages binaries and if necessary transfers them to other nodes. Tuple space is used for sharing internal middleware data between the nodes of a virtual cluster. The commu-

nication between application tasks is targeted to correct nodes by the data redirecting component.

Middleware Simulation Results

The concept and performance of WSN node middleware is evaluated with WISENES. The WISENES implementation of the middleware follows the architecture presented above. Two application scenarios are created for assessing the ability of the middleware to balance loading and to extend network lifetime. Middleware operation is adjusted by parameterizing the fairness of load balancing.

Middleware performance results are compared to a reference implementation, which does not incorporate any network level control. With a simple network and application, the middleware extends the time before the first node runs out of energy by a factor 6.85. In a more dynamic scenario, the factor is 3.9. Absolute lifetimes given in [P4] are short since nodes are powered by 0.22 F capacitors in order to demonstrate the performance of the middleware. The dynamic data memory consumption of the middleware is below 200 B.

5.3 Application-specific Tailoring of WSNs: Case Studies

The methodologies and tools presented in this Thesis are motivated by WSN applications. Two application scenarios have been implemented for illustrating application-specific tailoring of WSNs. Detailed results of the case studies are presented in publications [P5] and [P6].

Both applications are implemented with TUTWSN protocols and node platforms. In spite of the built-in support for operation adaptation and configuration, TUTWSN solutions and algorithms are best suited for low data rate monitoring networks [88]. In addition to [P1,P3-P6], TUTWSN protocol stack and prototype platforms are introduced in [87–90, 157, S2-S4,S6].

5.3.1 TCP/IP Experiments

Publication [P5] analyzes and implements a TUTWSN configuration for TCP/IP communication relaying. In an outlined use scenario, a low-power monitoring WSN is occasionally configured for improved throughput and delay, while still preserving overall energy efficiency.

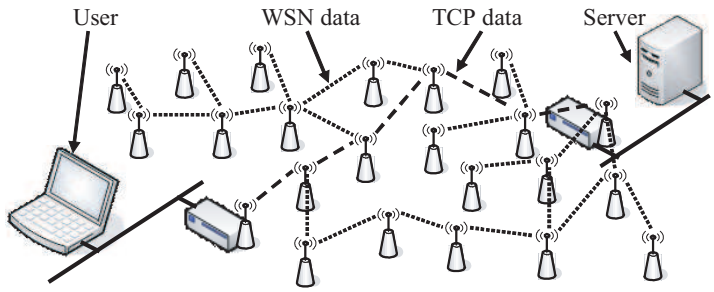


Fig. 14: Network architecture for TCP/IP communication in WSN.

Design Choices

The networking architecture for the application is depicted in Fig. 14. When a need for TCP/IP connection between two external users arises, a route for TCP data is created. Since TCP communication affects only to a subset of nodes, full feature TCP/IP stacks available for low resource MCUs are not an ultimate solution [36,211]. The connection maintenance and control signaling in TCP generates unnecessary network traffic for nodes not related to the link [27]. Therefore, TCP packets are fragmented and sent as application data within the network.

TUTWSN protocols are manually configured for the application. TUTWSN MAC protocol is adapted to higher throughput with a more frequent duty cycle. This does not interfere with the basic WSN operation and can be realized with a reasonable effort.

TCP/IP Experiment Results

A significant contribution of [P5] is the analysis and suitability assessment of TCP/IP for low-power WSNs. The adaptation results to trade-offs between TCP/IP performance and energy efficiency of WSNs. The analysis shows that legacy TCP flow control algorithms perform poorly in WSNs since packet losses are not caused by congestion but merely bit and node errors.

The performance of TUTWSN in the application-specific configuration is assessed by measuring Round-Trip Time (RTT), throughput, and power consumption with different network configurations. An average RTT for a 100 B packet over 10-hop network is below six seconds, which is an acceptable delay. Yet, RTT depends on the packet size and number of hops. These do not affect as much to the throughput, which is

2.3 kbps. The power consumption of a node participating to TCP/IP communication varies between 1.73 mW and 3.82 mW depending on the activity.

5.3.2 Indoor Surveillance WSN

In publication [P6], WISENES design flow and tools are used for designing and implementing an indoor surveillance WSN. First, the suitability of different TUTWSN configurations is manually explored with WISENES simulations. After the results indicate that a configuration meets application requirements for network longevity and communication delay, WSN is prototyped on top of SensorOS.

Design Principles

The indoor surveillance network has two application tasks, one for periodic temperature measurements for HVAC control and another for detecting motion in monitored premises. The periodic measurements have a low priority, but motion alerts require delay sensitive operation. The objective in the design is to find a TUTWSN configuration that adapts to the reactive operation required by motion detection but still preserves overall energy efficiency.

In order to show an example of configuration evaluation in WISENES, a bandwidth allocation algorithm in TUTWSN MAC protocol is parameterized with different guaranteed reservations. After the simulations are completed and the configuration found, application tasks are implemented as SensorOS threads and TUTWSN protocols parameterized accordingly.

Indoor Surveillance WSN Results

In order to assess application requirements for lifetime and reactivity, simulation models and prototypes are instrumented for the monitoring of power consumption and data routing delay. The simulated results show that with a configuration that grants guaranteed bandwidth for each node every sixth second, an average delay per hop is around four seconds. With the same configuration, a node equipped with a motion detection sensor consumes 443 μW in average. Moreover, average simulated power consumption is 650 μW for a node performing active data routing. This configuration is selected for prototyping, since its results indicate shortest per hop delay.

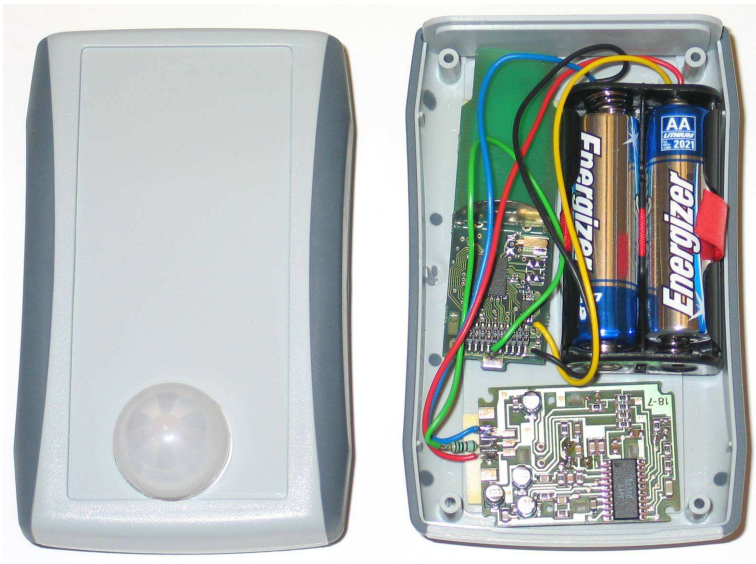


Fig. 15: TUTWSN PIC node with an integrated PIR sensor.

In prototyping phase, the same aspects are measured with physical TUTWSN prototypes. The measurements are done with TUTWSN PIC nodes with integrated PIR sensors. Fig. 15 depicts a TUTWSN PIR node on a plastic enclosure. Delay and power consumption results correspond accurately to simulated ones, as shown in Section 5.1.5. The prototype implementation of the motion detection task with TUTWSN stack supporting subnode functionality consumes 38.1 KB of code and 2 253 B of data memory on TUTWSN PIC nodes.

6. SUMMARY OF PUBLICATIONS

The publications of this thesis are based on the work of the author during years between 2003 and early 2007. This chapter summarizes the contents of the publications and clarifies the contribution of the author. The co-authors have seen and agree with the descriptions. None of the publications have previously been used as a part of a doctoral thesis.

The publications can be divided into two main groups. The publications [P1-P4] outline the requirements, and propose tools and methods for the design and implementation of WSNs. The second group consists of the publications [P5, P6] that present application-specific tailoring of WSN configurations.

Publication [P1] presents WISENES design and evaluation environment. The main focus of the publication is on the WISENES framework and its utilization for WSN design and evaluation. The accuracy of WISENES evaluation results is proved by TUTWSN simulations and prototype measurements. The performance of WISENES in large scale simulations is evaluated by a temperature measurement application with TUTWSN and a simplified version of ZigBee and IEEE 802.15.4 LR-WPAN protocol stack.

The author designed and implemented WISENES framework as well as the protocols and applications for the simulations. The publication was written by the author. Prof. Marko Hännikäinen and Prof. Timo D. Hämäläinen outlined the functional requirements for WISENES and revised the draft version of the publication.

Publication [P2] is a survey of systems software support for distributed processing in WSNs. The publication reviews and evaluates existing algorithms and methods that aim in facilitating the implementation of distributed applications. Further, requirements and implementation guidelines for systems software in WSNs are outlined.

The author conducted the background study, analyzed the material, and made conclusions according to the findings. The text was written by the author. Prof. Marko Hännikäinen and Prof. Timo D. Hämäläinen gave ideas for the survey and revised the text.

Publication [P3] presents the design and implementation of SensorOS, a lightweight OS for resource constrained WSN nodes. SensorOS is designed according to guidelines given in [P2]. SensorOS uses a preemptive priority-based scheduler and is suitable for time critical WSN applications and protocols. The publication illustrates the minimal resources consumption and timing accuracy of the OS, and gives an example of SensorOS usage in a simple WSN application scenario.

The author is the main architect of SensorOS. The implementation of SensorOS for TUTWSN nodes was mainly carried out by Mr. Timo Alho, M.Sc, assisted by the author. The author wrote the publication. Prof. Marko Hännikäinen and Prof. Timo D. Hämäläinen improved the writing style.

Publication [P4] proposes a task allocation middleware for distributed processing of WSN applications. Similarly to SensorOS, WSN node middleware is based on the findings of [P2]. WSN node middleware utilizes a simple but fair algorithm for balancing communication and computation load between nodes. The middleware abstracts underlying network topology by defining application communication primitives between tasks instead of nodes. WSN node middleware is designed and implemented in WISENES environment and evaluated by simulations.

The author designed and implemented the WSN node middleware in WISENES. The simulations, result analysis, and writing of the publication were done by the author. Prof. Marko Hännikäinen gave comments during the research and revised the text. Prof. Timo D. Hämäläinen improved the writing style.

Publication [P5] discusses and analyzes the suitability and approaches for integrating TCP/IP communication to low-power WSNs. In general, the communication profiles of TCP/IP and WSNs differ considerably, which sets challenges for the integration. The publication presents the configuration of TUTWSN protocols for higher bandwidth and shorter delay. The configuration is made manually without WISENES or SensorOS. The results indicate that such a configuration results to the trade-off between energy efficiency and performance.

The author configured and modified TUTWSN protocol stack for TCP/IP and implemented required software components for Linux. TUTWSN protocols are designed by Prof. Marko Hännikäinen, Mr. Mikko Kohvakka, M.Sc, Mr. Jukka Suhonen, M.Sc, and the author. TUTWSN node prototypes are designed by Mr. Kohvakka. Mr. Suhonen has implemented the general purpose version of the TUTWSN protocols for node prototypes. The publication was written by the author. Prof. Marko Hännikäinen supervised the work and revised the text. Prof. Timo D. Hämäläinen improved the text.

Publication [P6] composes the tools and methods presented in this Thesis. In the publication, an indoor surveillance WSN is designed and evaluated with WISENES. Based on the WISENES results, a suitable configuration is implemented on top of SensorOS. The publication shows the analogy between WISENES and SensorOS abstractions and illustrates the straightforward transition from the design to the implementation phase.

The author defined the application scenario, designed it in WISENES environment, and wrote the publication. The SensorOS design and implementation were performed by the author and Mr. Timo Alho, M.Sc. TUTWSN nodes are designed by Mr. Mikko Kohvakka, M.Sc, and protocols implemented by Mr. Jukka Suhonen, M.Sc. Mr. Teemu Laukkarinen modified TUTWSN protocol implementation for SensorOS. Mr. Jari Juntunen, M.Sc, implemented the indoor surveillance application and performed the measurements with node prototypes together with the author. Prof. Marko Hännikäinen provided comments and revised the text, Prof. Timo D. Hämäläinen suggested improvements to the draft version of the publication.

Supplementary publications

Supplementary publications [S1-S6] are not included into the Thesis since they are not essential to present the main contributions of this work. However, they give further insights to the research area. Of the publications, [S1] and [S5] present additional WISENES capabilities and simulations. The publication [S2] concentrates on the security issues in WSNs and shows the configuration of TUTWSN for security-enabled mode. TUTWSN protocols, algorithms, and interfaces used in the application examples of the Thesis are presented in [S3, S4, S6].

The supplementary publication [S1] explores the scalability of WISENES for WSNs with large number of nodes. The possibilities for generating executable code from high level SDL descriptions in WISENES to the node platforms are also discussed.

In [S2], the security threats faced in WSN deployments, and the architectures proposed for securing WSNs are reviewed. TUTWSN protocol stack is configured to support a centralized method to distribute secret keys for encryption and authentication.

The publication [S3] presents the design and implementation of a data and service access API. In the application scenarios presented in this Thesis, this WSN API has been used for data gathering and runtime configuration of the TUTWSN protocols.

The supplementary publications [S4] and [S6] present TUTWSN protocols and algorithms used in application examples of the Thesis. The principles and implementation of the TUTWSN routing protocol are discussed in [S4]. An algorithm for dynamic communication capacity allocation within the network is introduced in [S6].

In [S5], a mathematical analysis that evaluates the suitability of IEEE 802.15.4 LR-WPAN and ZigBee for large scale WSNs is given. WISENES simulations are used to validate the analysis models.

7. CONCLUSIONS

WSNs are envisioned to have a huge potential for diversity of applications that bridge humans, computers, and physical world. At its current state, the technology is not mature enough for realizing the far-reaching visions. However, within the last decade the advances in manufacturing technologies have made low-cost and low-power communication and computation devices possible.

The current devices enable a wide variety of applications, even though they do not yet meet the vision of "smart dust". Instead, most of the challenges lay on software, including communication protocols, applications, and systems software. Existing software architectures used e.g. in Internet do not suit for WSNs due to the extreme resource constraints. Further, standardization may not be an optimal solution for WSNs, since the divergent requirements call for application-specific solutions. While ZigBee standardizes a configurable protocol stack, it can address only a small subset of possible applications.

Even though the application-oriented nature and application-specific tailoring of deployments are commonly acknowledged characteristics of WSNs, there are no well-defined methods for structuring application requirements nor common methodologies for WSN design. The diversity of applications together with the possible configurations of node platforms and software components constitute a huge design space that need to be managed when designing a new WSN deployment.

As with maturing technologies in general, the abstraction level in WSN development increases as the technology evolves. The need for a low-level bit optimization and OS support remains, but higher level abstractions are required for the design and implementation of constantly larger and more complex WSNs. This necessitates systematic design methodologies and tools that support these abstractions throughout the design flow.

This Thesis presented a design methodology that provides abstract models for the description of WSN protocol and application functionality. These abstract deployment models are evaluated with simulations for assessing the performance of different de-

ployment configurations. A runtime environment extends the model abstractions also to the prototyping phase.

The design abstractions proposed in this Thesis are based on the WISENES deployment models. These models were found suitable for WSN functionality description at a high abstraction level. The main features of WISENES framework are the support for graphical design of deployment models using EFSM MoC, the evaluation of the network performance with accurate simulations of the models, and the back-annotation of performance results from prototypes.

SensorOS and WSN node middleware extend the WISENES design abstractions to the prototyping phase on real node platforms. SensorOS implements preemptive multithreading with a very small memory footprint. Further, its accurate time concept serves the needs of time-sensitive WSN applications and protocols. WSN node middleware creates a seemingly distributed application processing environment on top of SensorOS. Its lightweight but efficient task allocation algorithm lengthens network lifetime by balancing application and networking processing between nodes. Even though the code generation from WISENES to the final implementation was not considered in this Thesis, the required technology and tools are already available.

Two case studies were shown to illustrate the challenges involved, and the feasibility of selected abstractions and tools. Both application scenarios were relative simple but typical for current WSNs. A performance optimized network configuration is suitable also for other applications than TCP/IP. The indoor surveillance application represented applications that have multiple tasks with contradictory requirements.

The main benefit of WISENES compared to the related work is that the accuracy of simulations makes it possible to verify the WSN performance reliably already during the high abstraction level design phase. Together with well-defined deployment models and graphical design environment, this hastens the design and configuration of suitable protocols and algorithms for a WSN deployment. The systems software supporting the same MoC on node platforms facilitates the transition from the design phase to the prototyping and final implementation.

Current WSN applications are hindered by the resource and programming model limitations set by the platforms and runtime environments. Since the resource constraints will most likely persist also in future due to the trend towards a smaller physical size, more complex and demanding applications need to be enabled by evolving programming models. The system level design methodology proposed in this Thesis solves this by defining high level abstractions for the description of application functionality and distribution, and by supporting these abstractions throughout the design flow.

BIBLIOGRAPHY

[1] K. Aberer, M. Hauswirth, and A. Salehi, "The global sensor network middleware for efficient and flexible deployment and interconnection of sensor networks," LSIR, Tech. Rep. 2006-006, April 2006. [Online]. Available: http://infoscience.epfl.ch/

[2] K. Akkaya and M. Younis, "A survey on routing protocols for wireless sensor networks," *Elsevier Ad hoc Networks*, vol. 3, no. 3, pp. 325–349, May 2005.

[3] I. F. Akyildiz and I. H. Kasimoglu, "Wireless sensor and actor networks: Research challenges," *Elsevier Ad Hoc Networks*, vol. 2, no. 4, pp. 351–367, October 2004.

[4] I. F. Akyildiz, T. Melodia, and K. R. Chowdhury, "A survey on wireless multimedia sensor networks," *Elsevier Computer Networks*, vol. 51, no. 4, pp. 921–960, March 2007.

[5] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "A survey on sensor networks," *IEEE Communications Magazine*, vol. 40, no. 8, pp. 102–114, August 2002.

[6] ——, "Wireless sensor networks: a survey," *Elsevier Computer Networks*, vol. 38, no. 4, pp. 393–422, March 2002.

[7] J. N. Al-Karaki and A. E. Kamal, "Routing techniques in wireless sensor networks: a survey," *IEEE Wireless Communications*, vol. 11, no. 6, pp. 6–28, December 2004.

[8] (2007) This is ant, a wireless personal area network solution website. [Online]. Available: http://www.thisisant.com/

[9] W. Archer, P. Levis, and J. Regehr, "Interface contracts for tinyos," in *Proc. 6th international Conference on information Processing in Sensor Networks (IPSN'07)*, Cambridge, MA, USA, April 25-27 2007, pp. 158–165.

[10] (2007) Avrora website. [Online]. Available: http://compilers.cs.ucla.edu/avrora/

[11] A. Bakshi and V. K. Prasanna, "Algorithm design and synthesis for wireless sensor networks," in *Proc. 2004 International Conference on Parallel Processing*, Montreal, Quebec, Canada, August 15–18 2004, pp. 423–430.

[12] P. Baldwin, S. Kohli, E. A. Lee, X. Liu, and Y. Zhao, "Modeling of sensor nets in ptolemy II," in *Proc. 3rd International Symposium on Information Processing in Sensor Networks (IPSN'04)*, Berkeley, CA, USA, April 26–27 2004, pp. 359–368.

[13] P. Baronti, P. Pillai, V. W. Chook, S. Chessa, A. Gotta, and Y. F. Hu, "Wireless sensor networks: A survey on the state of the art and the 802.15.4 and ZigBee standards," *Elsevier Computer Communications*, vol. 30, no. 7, pp. 1655–1695, May 2007.

[14] R. Barr, J. C. Bicket, D. S. Dantas, B. Du, T. D. Kim, B. Zhou, and E. G. Sirer, "On the need for system-level support for ad hoc and sensor networks," *ACM SIGOPS Newsletter on Operating Systems Review*, vol. 36, no. 2, pp. 1–5, April 2002.

[15] M. A. Batalin, M. Rahimi, Y. Yu, D. Liu, A. Kansal, G. S. Sukhatme, W. J. Kaiser, M. Hansen, G. J. Pottie, M. Srivastava, and D. Estrin, "Call and response: experiments in sampling the environment," in *Proc. 2nd International Conference on Embedded Networked Sensor Systems (SenSys'04)*, Baltimore, MD, USA, November 3–5 2004, pp. 25–38.

[16] R. Beckwith, D. Teibel, and P. Bowen, "Report from the field: Results from an agricultural wireless sensor network," in *Proc. 29th Annual IEEE International Conference on Local Computer Networks (LCN'04)*, Tampa, FL, USA, November 16–18 2004, pp. 471–478.

[17] J. Beutel, O. Kasten, F. Mattern, K. Römer, F. Siegemund, and L. Thiele, "Prototyping wireless sensor network applications with btnodes," in *Wireless Sensor Networks, First European Workshop, EWSN 2004*, ser. Lecture Notes in Computer Science, H. Karl, A. Willig, and A. Wolisz, Eds. Springer, 2004, vol. 2920, pp. 323–338.

[18] J. Beutel, O. Kasten, M. Ringwald, F. Siegemund, and L. Thiele, "Bluetooth smart nodes for ad-hoc networks," Computer Engineering and Networks

Laboratory, ETH Zurich, Tech. Rep. 167, April 2003. [Online]. Available: http://www.tik.ee.ethz.ch/

[19] S. Bhatti, J. Carlson, H. Dai, J. Deng, J. Rose, A. Sheth, B. Shucker, C. Gruenwald, A. Torgerson, and R. Han, "MANTIS OS: An embedded multithreaded operating system for wireless micro sensor platforms," *Mobile Networks and Applications*, vol. 10, no. 4, pp. 563–579, August 2005.

[20] *Specification of the Bluetooth System*, Bluetooth Special Interest Group (SIG) Std. 2.0 + EDR, 2004.

[21] A. Bonivento, L. P. Carloni, and A. Sangiovanni-Vincentelli, "Platform based design for wireless sensor networks," *Mobile Networks and Applications*, vol. 11, no. 4, pp. 469–485, August 2006.

[22] A. Boulis, C.-C. Han, and M. B. Srivastava, "Design and implementation of a framework for efficient and programmable sensor networks," in *Proc. 1st International Conference on on Mobile Systems, Applications, and Service (MobiSys'03)*, San Francisco, CA, USA, May 5–8 2003, pp. 187–200.

[23] X. Carcelle, T. Dang, and C. Devic, "Industrial wireless technologies: applications for the electrical utilities," in *Proc. IEEE International Conference on Industrial Informatics (INDIN'06)*, Singapore, August 16–18 2006, pp. 108–113.

[24] H. Cha, S. Choi, I. Jung, H. Kim, H. Shin, J. Yoo, and C. Yoon, "RETOS: resilient, expandable, and threaded operating system for wireless sensor networks," in *Proc. 6th international Conference on information Processing in Sensor Networks (IPSN'07)*, Cambridge, MA, USA, April 25-27 2007, pp. 148–157.

[25] E. Cheong, E. A. Lee, and Y. Zhao, "Viptos: A graphical development and simulation environment for tinyos-based wireless sensor," UCB, Tech. Rep. UCB/EECS-2006-15, February 2006. [Online]. Available: http://www.eecs.berkeley.edu/

[26] C.-Y. Chong and S. P. Kumar, "Sensor networks: Evolution, opportunities, and challenges," *Proceedings of the IEEE*, vol. 91, no. 8, pp. 1247–1256, August 2003.

[27] W. Chonggang, K. Sohraby, H. Yueming, L. Bo, and T. Weiwen, "Issues of transport control protocols for wireless sensor networks," in *Proc. 2005 Inter-*

national Conference on Communications, Circuits and Systems (ICCCAS'05), HongKong, China, May 27–30 2005, pp. 442–426.

[28] D. Chu, K. Lin, A. Linares, G. Nguyen, and J. M. Hellerstein, "sdlib: a sensor network data and communications library for rapid and robust application development," in *Proc. 5th international Conference on information Processing in Sensor Networks (IPSN'06)*, Nashville, TN, USA, April 19-21 2006, pp. 432–440.

[29] P. Costa, L. Mottola, A. L. Murphy, and G. P. Picco, "TeenyLIME: transiently shared tuple space middleware for wireless sensor networks," in *Proc. International Workshop on Middleware For Sensor Networks (MidSens'06)*, Melbourne, Australia, November 28 2006, pp. 43–48.

[30] K. Crowley, J. Frisby, S. Murphy, M. Roantree, and D. Diamond, "Web-based real-time temperature monitoring of shellfish catches using a wireless sensor network," *Sensors and Actuators A: Physical*, vol. 122, no. 2, pp. 222–230, August 2005.

[31] D. Culler, D. Estrin, and M. B. Srivastava, "Overview of sensor networks," *IEEE Computer*, vol. 37, no. 8, pp. 41–49, August 2004.

[32] D. E. Culler, J. Hill, P. Buonadonna, R. Szewczyk, and A. Woo, "A network-centric approach to embedded software for tiny devices," in *Lecture Notes in Computer Science*. Springer, 2001, vol. 2211, pp. 114–130.

[33] C. Curino, M. Giani, M. Giorgetta, A. Giusti, A. L. Murphy, and G. P. Picco, "TinyLIME: bridging mobile and sensor networks through middleware," in *Proc. 3rd IEEE International Conference on Pervasive Computing and Communications (PerCom'05)*, Kauai Island, HI, USA, March 8–12 2005, pp. 61–72.

[34] A. D'Costa and A. M. Sayeed, "Collaborative signal processing for distributed classification in sensor networks," in *Information Processing in Sensor Networks: 2nd International Workshop, IPSN 2003*, ser. Lecture Notes in Computer Science, F. Zhao and L. J. Guibas, Eds. Springer, 2005, vol. 2634, pp. 126–140.

[35] M. Diaz, B. Rubio, and J. M. Troya, "A coordination middleware for wireless sensor networks," in *Proc. 2005 Systems Communications (ICW'05, ICHSN'05, ICMCS'05, SENET'05))*, Montreal, Quebec, Canada, August 14–17 2005, pp. 377–382.

[36] A. Dunkels, "Full TCP/IP for 8-bit architectures," in *Proc. 1st International Conference on Mobile Systems, Applications and Services (MobiSys'03)*, San Francisco, CA, USA, May 5–8 2003, pp. 85–98.

[37] A. Dunkels, N. Finne, J. Eriksson, and T. Voigt, "Run-time dynamic linking for reprogramming wireless sensor networks," in *Proc. 4th International Conference on Embedded Networked Sensor Systems (SenSys'06)*, Boulder, CO, USA, October 31–November 3 2006, pp. 15–28.

[38] A. Dunkels, B. Grönvall, and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors," in *Proc. 29th Annual IEEE International Conference on Local Computer Networks (1st IEEE Workshop on Embedded Networked Sensors) (EmNetS,04)*, Tampa, FL, USA, November 16–18 2004, pp. 455–462.

[39] A. Dunkels, O. Schmidt, T. Voigt, and M. Ali, "Protothreads: Simplifying event-driven programming of memory-constrained embedded systems," in *Proc. 4th International Conference on Embedded Networked Sensor Systems (SenSys'06)*, Boulder, CO, USA, October 31–November 3 2006, pp. 29–42.

[40] J. Elson and D. Estrin, *Sensor Networks: A Bridge to the Physical World*. Springer, 2004, pp. 3–20.

[41] J. Elson, L. Girod, and D. Estrin, "EmStar: Development with high system visibility," *IEEE Wireless Communications*, vol. 11, no. 6, pp. 70–77, December 2004.

[42] J. Elson and A. Parker, "Tinker: a tool for designing data-centric sensor networks," in *Proc. 5th international Conference on information Processing in Sensor Networks (IPSN'06)*, Nashville, TN, USA, April 19-21 2006, pp. 350–357.

[43] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar, "Next century challenges: Scalable coordination in sensor networks," in *Proc. 5th annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'99)*, Seattle, WA, USA, August 15–20 1999, pp. 263–270.

[44] A. Eswaran, A. Rowe, and R. Rajkumar, "Nano-RK: An energy-aware resource-centric RTOS for sensor networks," in *26th IEEE International Real-Time Systems Symposium (RTSS'05)*, Miami, FL, USA, December 5–8 2005, pp. 256–265.

[45] (2007) HiperLAN website. [Online]. Available: http://portal.etsi.org/radio/HiperLAN/HiperLAN.asp

[46] (2007) HiperMAN website. [Online]. Available: http://portal.etsi.org/radio/HiperMAN/HiperMAN.asp

[47] E. Fasolo, M. Rossi, J. Widmer, and M. Zorzi, "In-network aggregation techniques for wireless sensor networks: a survey," *IEEE Wireless Communications*, vol. 14, no. 2, pp. 70–87, April 2007.

[48] D. Flowers and Y. Yang, "MiWi wireless networking protocol stack," Microchip Technology Inc., Tech. Rep. AN1066, February 2007. [Online]. Available: http://www.microchip.com/

[49] C.-L. Fok, G.-C. Roman, and C. Lu, "Rapid development and flexible deployment of adaptive wireless sensor network applications," in *Proc. 25th IEEE International Conference on Distributed Computing Systems (ICDCS'05)*, Columbus, OH, USA, June 6–10 2005, pp. 653–662.

[50] A. Fraboulet, G. Chelius, and E. Fleury, "Worldsens: development and prototyping tools for application specific wireless sensors networks," in *Proc. 6th international Conference on information Processing in Sensor Networks (IPSN'07)*, Cambridge, MA, USA, April 25-27 2007, pp. 176–185.

[51] A. Fuggetta, G. P. Picco, and G. Vigna, "Understanding code mobility," *IEEE Transactions on Software Engineering*, vol. 24, no. 5, pp. 342–361, May 1998.

[52] D. Gay, P. Levis, R. v. Behren, M. Welsh, E. Brewer, and D. Culler, "The nesC language: A holistic approach to networked embedded systems," in *Proc. ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation (PLDI'03)*, San Diego, CA, USA, June 9–11 2003, pp. 1–11.

[53] D. Gelernter, "Generative communication in linda," *ACM Programming Languages and Systems*, vol. 7, no. 1, pp. 80–112, January 1985.

[54] D. Gracanin, M. Eltoweissy, A. Wadaa, and L. A. DaSilva, "A service-centric model for wireless sensor networks," *IEEE Journal on Selected Areas in Communications*, vol. 23, no. 6, pp. 1159–1166, June 2005.

[55] L. Gu and J. A. Stankovic, "t-kernel: providing reliable OS support to wireless sensor networks," in *Proc. 4th International Conference on Embedded Networked Sensor Systems (SenSys'06)*, Boulder, CO, USA, October 31–November 3 2006, pp. 1–14.

[56] R. Gummadi, O. Gnawali, and R. Govindan, "Macro-programming wireless sensor networks using kairos," in *Distributed Computing in Sensor Systems, 1st IEEE International Conference, DCOSS 2005*, ser. Lecture Notes in Computer Science, V. K. Prasanna, S. S. Iyengar, P. G. Spirakis, and M. Welsh, Eds. Springer, 2005, vol. 3560, pp. 126–140.

[57] J. A. Gutiérrez, E. H. Callaway, Jr., and R. L. Barrett, Jr., *Low-Rate Wireless Personal Area Networks ... Enabling Wireless Sensors with IEEE 802.15.4*, ser. IEEE Standards Wireless Networks Series. IEEE Press, 2004.

[58] E. Guttman, C. Perkins, J. Veizades, and M. Day, *Service Location Protocol, version 2*, IETF Std. RFC 2608, 1999.

[59] S. Hadim and N. Mohamed, "Middleware challenges and approaches for wireless sensor networks," *IEEE Distributed Systems Online*, vol. 7, no. 3, pp. 1–15, March 2006.

[60] ——, "Middleware for wireless sensor networks: A survey," in *Proc. 1st International Conference on Communication System Software and Middleware (Comsware'06)*, New Delhi, India, January 8–12 2006.

[61] C.-C. Han, R. Kumar, R. Shea, E. Kohler, and M. Srivastava, "A dynamic operating system for sensor nodes," in *Proc. 3rd International Conference on Mobile Systems, Applications, and Services (MobiSys'05)*, Seattle, WA, USA, June 6–8 2005, pp. 163–176.

[62] J. K. Hart and K. Martinez, "Environmental sensor networks: A revolution in the earth system science?" *Earth-Science Reviews*, vol. 78, no. 3-4, pp. 177–191, October 2006.

[63] T. He, S. Krishnamurthy, L. Luo, T. Yan, L. Gu, R. Stoleru, G. Zhou, Q. Cao, P. Vicaire, J. A. Stankovic, T. F. Abdelzaher, J. Hui, and B. Krogh, "VigilNet: An integrated sensor network system for energy-efficient surveillance," *ACM Transactions on Sensor Networks*, vol. 2, no. 1, pp. 1–38, February 2006.

[64] W. B. Heinzelman, A. P. Chandrakasan, and H. Balakrishnan, "An application-specific protocol architecture for wireless microsensor networks," *IEEE Transactions on Wireless Communications*, vol. 1, no. 4, pp. 660–670, October 2002.

[65] W. B. Heinzelman, A. L. Murphy, H. S. Carvalho, and M. A. Perillo, "Middleware to support sensor network applications," *IEEE Network*, vol. 1, no. 18, pp. 6–14, January/February 2004.

[66] K. Henricksen and R. Robinson, "A survey of middleware for sensor networks: state-of-the-art and future directions," in *Proc. International Workshop on Middleware For Sensor Networks (MidSens'06)*, Melbourne, Australia, November 28 2006, pp. 60–65.

[67] J. Hill and D. E. Culler, "Mica: a wireless platform for deeply embedded networks," *IEEE Micro*, vol. 22, no. 6, pp. 12–24, November/December 2002.

[68] J. Hill, M. Horton, R. Kling, and L. Krishnamurthy, "The platforms enabling wireless sensor networks," *Communications of the ACM*, vol. 47, no. 6, pp. 41–46, June 2004.

[69] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, "System architecture directions for networked sensors," in *Proc. 9th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'00)*, Cambridge, MA, USA, November 12–15 2000, pp. 94–103.

[70] P. Hämäläinen, "Cryptographic security designs and hardware architectures for wireless local area networks," Ph.D. dissertation, Tampere University of Technology, Tampere, Finland, December 2006.

[71] M. Hännikäinen, "Design of quality of service support for wireless local area networks," Ph.D. dissertation, Tampere University of Technology, Tampere, Finland, November 2002.

[72] M. Hännikäinen, J. Knuutila, T. D. Hämäläinen, and J. Saarinen, "Using SDL for implementing a wireless medium access control protocol," in *Proc. IEEE International Symbosium on Multimedia Software Engineering(MSE'00)*, Taipei, Taiwan, December 11–13 2000, pp. 229–236.

[73] M. Horstmann and M. Kirtland. (1997) Dcom architecture. Microsoft Corporation. [Online]. Available: http://msdn2.microsoft.com/en-us/library/

[74] *IEEE Standard for Information Technology—Telecommunications and Information Exchange Between Systems—Local and Metropolitan Area Networks—Specific Requirements—Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, IEEE Std. 802.11, 1999.

[75] *IEEE Standard for Information Technology - Portable Operating System Interface (POSIX)*, IEEE Std. 1003.1, 2004.

[76] *IEEE Standard for Local and Metropolitan Area Networks – Part 16: Air Interface for Fixed Broadband Wireless Access Systems*, IEEE Std. 802.16, 2004.

[77] *IEEE Standard for Information Technology—Telecommunications and Information Exchange Between Systems—Local and Metropolitan Area Networks—Specific Requirements—Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPAN)*, IEEE Std. 802.15.4, 2006.

[78] (2007) IEEE 1451.5 project website. [Online]. Available: http://grouper.ieee.org/groups/1451/5/

[79] (2007) IEEE 802.11 working group website. [Online]. Available: http://grouper.ieee.org/groups/802/11/

[80] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva, "Directed diffusion for wireless sensor networking," *IEEE/ACM Transactions on Networking*, vol. 11, no. 1, pp. 2–16, February 2003.

[81] *Information Technology–Open Systems Interconnection–Basic Reference Model: The Basic Model*, ISO/IEC Std. 7498-1, 1994.

[82] C. Jaikaeo, C. Srisathapornphat, and C.-C. Shen, "Querying and tasking in sensor networks," in *SPIE's 14th Annual International Symposium on Aerospace/Defense Sensing, Simulation, and Control (Digitization of the Battlespace V)*, vol. 4037, Orlando, FL, USA, April 24–28 2000, pp. 184–197.

[83] A. Jantsch, *Modeling Embedded Systems and SoCs: Concurrency and Time in Models of Computation*. Morgan Kaufmann Publishers, 2004.

[84] T. Kangas, "Methods and implementations for automated system on chip architecture exploration," Ph.D. dissertation, Tampere University of Technology, Tampere, Finland, September 2006.

[85] H. Karl and A. Willig, *Protocols and Architectures for Wireless Sensor Networks*. John Wiley & Sons Ltd, 2005.

[86] K. Keutzer, S. Malik, A. R. Newton, J. M. Rabaey, and A. Sangiovanni-Vincentelli, "System-level design: Orthogonalization of concerns and platform-based design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no. 12, pp. 1523–1543, December 2000.

[87] M. Kohvakka, M. Hännikäinen, and T. D. Hämäläinen, "Energy optimized beacon transmission rate in a wireless sensor network," in *Proc. The 16th Annual IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC'05)*, Berlin, Germany, September 11–14 2005, pp. 1269–1273.

[88] ——, "Ultra low energy wireless temperature sensor network implementation," in *Proc. The 16th Annual IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC'05)*, Berlin, Germany, September 11–14 2005, pp. 801–805.

[89] ——, "Wireless sensor network implementation for industrial linear position metering," in *Proc. 8th Euromicro Conference on Digital System Design (DSD'05)*, Porto, Portugal, August 30–September 3 2005, pp. 267–273.

[90] M. Kohvakka, J. Suhonen, M. Hännikäinen, and T. D. Hämäläinen, "Transmission power based path loss metering for wireless sensor networks," in *Proc. The 17th Annual IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC'06)*, Helsinki, Finland, September 11–14 2006.

[91] J. Koshy and R. Pandey, "VMstar: synthesizing scalable runtime environments for sensor networks," in *Proc. 3rd International Conference on Embedded Networked Sensor Systems (SenSys'05)*, San Diego, CA, USA, November 2–4 2005, pp. 243–254.

[92] N. Kothari, R. Gummadi, T. Millstein, and R. Govindan, "Reliable and efficient programming abstractions for wireless sensor networks," in *Proc. 2007 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'07)*, San Diego, CA, USA, June 10–13 2007, pp. 200–210.

[93] J. Kulik, W. B. Heinzelman, and H. Balakrishnan, "Negotiation-based protocols for disseminating information in wireless sensor networks," *Kluwer Wireless Networks*, vol. 8, no. 2, pp. 169–185, May 2002.

[94] R. Kumar, M. Wolenetz, B. Agarwalla, J. Shin, P. Hutto, A. Paul, and U. Ramachandran, "DFuse: a framework for distributed data fusion," in *Proc. 1st International Conference on Embedded Networked Sensor Systems (SenSys'03)*, Los Angeles, CA, USA, November 5–7 2003, pp. 114–125.

[95] P. Levis and D. Culler, "Maté: a tiny virtual machine for sensor networks," in *Proc. 10th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'02)*, San Jose, CA, USA, October 5–9 2002, pp. 85–95.

[96] P. Levis, N. Lee, M. Welsh, and D. Culler, "TOSSIM: accurate and scalable simulation of entire TinyOS applications," in *Proc. 1st International Conference on Embedded Networked Sensor Systems (SenSys'03)*, Los Angeles, CA, USA, November 5–7 2003, pp. 126–137.

[97] S. Li, Y. Lin, S. H. Son, J. A. Stankovic, and Y. Wei, "Event detection services using data service middleware in distributed sensor networks," *Telecommunication Systems*, vol. 26, no. 2, pp. 351–368, June 2004.

[98] J. Lifton, D. Seetharam, M. Broxton, and J. Paradiso, "Pushpin computing system overview: a platform for distributed, embedded, ubiquitous sensor networks," in *Proc. 1st International Conference on Pervasive Computing (Pervasive'02)*, Zurich, Switzerland, August 26–28 2002, pp. 139–151.

[99] (2006) Linux standard base, version 3.1. Linux Foundation. [Online]. Available: http://www.linuxbase.org/

[100] J. Liu, L. F. Perrone, D. M. Nicol, M. Liljenstam, C. Elliott, and D. Pearson, "Simulation modeling of large-scale ad-hoc sensor networks," in *Proc. 2001 Simulation Interoperability Workshop*, Harrow, Middlesex, UK, June 25–27 2001.

[101] J. Liu, M. Chu, J. Liu, J. Reich, and F. Zhao, "State-centric programming for sensor-actuator network systems," *IEEE Pervasive Computing*, vol. 2, no. 4, pp. 50–62, October–December 2003.

[102] T. Liu and M. Martonosi, "Impala: a middleware system for managing autonomic, parallel sensor systems," in *Proc. 9th ACM SIGPLAN symposium on Principles and Practice of Parallel Programming (PPoPP'03)*, San Diego, CA, USA, June 11–13 2003, pp. 107–118.

[103] Y. Liu and S. K. Das, "Information-intensive wireless sensor networks: potential and challenges," *IEEE Communications Magazine*, vol. 44, no. 11, pp. 142–147, November 2006.

[104] C. G. Luca Benini, Elisabetta Farella, "Wireless sensor networks: Enabling technology for ambient intelligence," *Microelectronics Journal*, vol. 37, no. 12, pp. 1639–1649, December 2006.

[105] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "The design of an acquisitional query processor for sensor networks," in *Proc. ACM International Conference on Management of Data (SIGMOD'03)*, San Diego, CA, USA, June 9–12 2003, pp. 491–502.

[106] K. Martinez, P. Padhy, A. Riddoch, R. Ong, and J. Hart, "Glacial environment monitoring using sensor networks," in *Proc. REALWSN 2005*, Stockholm, Sweden, June 20–21 2005.

[107] (2006) UPnP device architecture. Microsoft Corporation. [Online]. Available: http://www.upnp.org/resources/documents.asp

[108] A. Mihovska, F. Platbrood, G. Karetsos, S. Kyriazakos, R. V. Muijen, R. Guarneri, and J. M. Pereira, "Towards the wireless 2010 vision: A technology roadmap," *Wireless Personal Communications*, vol. 42, no. 3, pp. 303–336, August 2007.

[109] A. L. Murphy and G. P. P. and, "LIME: a middleware for physical and logical mobility," in *Proc. 21st International Conference on Distributed Computing Systems (ICDCS-21)*, Phoenix, AZ, USA, April 16–19 2001, pp. 524–533.

[110] (2007) Wireless ad hoc sensor networks website. National Institute of Standards and Technology – Advanced Network Technologies Division. [Online]. Available: http://w3.antd.nist.gov/wahn_ssn.shtml

[111] Y. Neuvo, "Future wireless technologies," in *Proc. IEEE 6th CAS Symposium on Emerging Technologies: Frontiers of Mobile and Wireless Communication*, Shanghai, China, May 31–June 2 2004, pp. I_1–I_3.

[112] R. Newton, Arvind, and M. Welsh, "Building up to macroprogramming: an intermediate language for sensor networks," in *Proc. 4th international Conference on information Processing in Sensor Networks (IPSN'05)*, Los Angeles, CA, USA, April 24–27 2005.

[113] D. Niculescu, "Communication paradigms for sensor networks," *IEEE Communications Magazine*, vol. 43, no. 3, pp. 116–122, March 2005.

[114] S. Oh, P. Chen, M. Manzo, and S. Sastry, "Instrumenting wireless sensor networks for real-time surveillance," in *Proc. 2006 IEEE International Conference on Robotics and Automation (ICRA'06)*, Orlando, FL, USA, May 15–19 2006, pp. 3128–3133.

[115] S. Olariu, A. Wada, L. Wilson, and M. Eltoweissy, "Wireless sensor networks: leveraging the virtual infrastructure," *IEEE Network*, vol. 4, no. 18, pp. 51–56, July/August 2004.

[116] (2004) Single unix specification, version 3, 2004 edition. The Open group. [Online]. Available: http://www.opengroup.org/

[117] E. Ould-Ahmed-Vall, G. F. Riley, B. S. Heck, and D. Reddy, "Simulation of large-scale sensor networks using GTSNetS," in *Proc. 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'05)*, Atlanta, GA, USA, September 27–29 2005, pp. 211–218.

[118] H. Park and M. B. Srivastava, "Energy-efficient task assignment framework for wireless sensor networks," CENS, Tech. Rep. 0026, September 2003. [Online]. Available: http://research.cens.ucla.edu

[119] A. Perrig, J. Stankovic, and D. Wagner, "Security in wireless sensor networks," *Communications of the ACM*, vol. 47, no. 6, pp. 53–57, June 2004.

[120] J. Polastre, J. Hill, and D. Culler, "Versatile low power media access for wireless sensor networks," in *Proc. 2nd International Conference on Embedded Networked Sensor Systems (SenSys'04)*, Baltimore, MD, USA, November 3–5 2004, pp. 95–107.

[121] J. Polley, D. Blazakis, J. McGee, D. Rusk, J. S. Baras, and M. Karir, "ATEMU: A fine-grained sensor network simulator," in *Proc. 1st IEEE International Conference on Sensor and Ad Hoc Communication Networks (SECON'04)*, Santa Clara, CA, USA, October 4–7 2004, pp. 145–152.

[122] (2006) Ptolemy II website. [Online]. Available: http://ptolemy.eecs.berkeley.edu/ptolemyII

[123] H. Qi, Y. Xu, and X. Wang, "Mobile-agent-based collaborative signal and information processing in sensor networks," *Proceedings of the IEEE*, vol. 91, no. 8, pp. 1172–1183, August 2003.

[124] J. M. Rabaey, M. J. Ammer, J. L. da Silva Jr., D. Patel, and S. Roundy, "Picoradio supports ad hoc ultra-low power wireless networking," *IEEE Computer*, vol. 33, no. 7, pp. 42–48, July 2000.

[125] J. M. Reason and J. M. Rabaey, "A study of energy consumption and reliability in a multi-hop sensor network," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 8, no. 1, pp. 84–97, January 2004.

[126] N. Reijers and K. Langendoen, "Efficient code distribution in wireless sensor networks," in *Proc. 2nd International Workshop on Wireless Sensor Networks and Applications (WSNA'03)*, San Diego, CA, USA, September 19 2003, pp. 60–67.

[127] G. F. Riley, "Large-scale network simulations with GTNetS," in *Proc. 35th Winter Simulation Conference (WSC'03)*, New Orleans, LA, USA, December 7–10 2003, pp. 676–684.

[128] O. Riva and C. Borcea, "The urbanet revolution: Sensor power to the people!" *IEEE Pervasive Computing*, vol. 6, no. 2, pp. 41–49, April–June 2007.

[129] K. Römer, "Programming paradigms and middleware for sensor networks," in *Proc. GI/ITG Workshop on Sensor Networks*, Karlsruhe, Germany, February 26–27 2004, pp. 49–54.

[130] ——, "Tracking real-world phenomena with smart dust," in *Wireless Sensor Networks, First European Workshop, EWSN 2004*, ser. Lecture Notes in Computer Science, H. Karl, A. Willig, and A. Wolisz, Eds. Springer, 2004, vol. 2920, pp. 28–43.

[131] K. Römer, O. Kasten, and F. Mattern, "Middleware challenges for wireless sensor networks," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 6, no. 4, pp. 59–61, October 2002.

[132] K. Römer and F. Mattern, "The design space of wireless sensor networks," *IEEE Wireless Communications*, vol. 11, no. 6, pp. 54–61, December 2004.

[133] C. Rowen, *Engineering the Complex SOC - Fast, Flexible Design with Configurable Processors*, ser. Prentice Hall Modern Semiconductor Design Series. Prentice Hall, 2004.

[134] N. Sadagopan, B. Krishnamachari, and A. Helmy, "Active query forwarding in sensor networks," *Elsevier Ad Hoc Networks*, vol. 3, no. 1, pp. 91–113, January 2005.

[135] A. Sangiovanni-Vincentelli, "Quo vadis, SLD? reasoning about the trends and challenges of system level design," *Proceedings of the IEEE*, vol. 95, no. 3, pp. 467–506, March 2007.

[136] P. Santi, "Topology control in wireless ad hoc and sensor networks," *ACM Computing Surveys*, vol. 37, no. 2, pp. 164–194, June 2005.

[137] T. Schmid, H. Dubois-Ferriere, and M. Vetterli, "Sensorscope: Experiences with a wireless building monitoring sensor network," in *Proc. REALWSN 2005*, Stockholm, Sweden, June 20–21 2005.

[138] D. C. Schmidt, "Model driven engineering," *IEEE Computer*, vol. 39, no. 2, pp. 25–31, February 2006.

[139] L. Schwiebert, S. K. Gupta, and J. Weinmann, "Research challenges in wireless networks of biomedical sensors," in *Proc. 7th annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'01)*, Rome, Italy, July 16–21 2001, pp. 151–165.

[140] (2007) SDL Forum Society website. [Online]. Available: http://www.sdl-forum.org/

[141] C. Sharp, S. Schaffert, A. Woo, N. Sastry, C. Karlof, S. Sastry, and D. Culler, "Design and implementation of a sensor network system for vehicle tracking and autonomous interception," in *Proc. 2nd European Workshop on Wireless Sensor Networks (EWSN'05)*, Istanbul, Turkey, January 31–February 2 2005, pp. 93–107.

[142] C.-C. Shen, C. Srisathapornphat, and C. Jaikaeo, "Sensor information networking architecture and applications," *IEEE Personal communications*, vol. 8, no. 4, pp. 52–59, August 2001.

[143] C.-C. Shen, C. Badr, K. Kordari, S. S. Bhattacharyya, G. L. Blankenship, and N. Goldsman, "A rapid prototyping methodology for application-specific sensor networks," in *Proc. IEEE International Workshop on Computer Architecture for Machine Perception and Sensing (CAMPS'06)*, Montreal, Quebec, Canada, September 18–20 2006.

[144] A. Sikora, "Design challenges for short-range wireless networks," *IEE Proc. Communications*, vol. 151, no. 5, pp. 473–479, October 2004.

[145] G. Simon, M. Maróti, Ákos Lédeczi, G. Balogh, B. Kusy, A. Nádas, G. Pap, J. Sallai, and K. Frampton, "Sensor network-based countersniper system," in *Proc. 2nd International Conference on Embedded Networked Sensor Systems (SenSys'04)*, Baltimore, MD, USA, November 3–5 2004, pp. 1–12.

[146] G. Simon, P. Völgyesi, M. Maróti, and Ákos Lédeczi, "Simulation-based optimization of communication protocols for large-scale wireless sensor networks," in *Proc. 2003 IEEE Aerospace Conference (AeroConf'03)*, vol. 3, Big Sky, MT, USA, March 8–15 2003, pp. 1339–1346.

[147] E. G. Sirer, R. Grimm, A. J. Gregory, and B. N. Bershad, "Design and implementation of a distributed virtual machine for networked computers," in *Proc. 17th ACM Symposium on Operating Systems Principles (SOSP'99)*, Kiawah Island, SC, USA, December 12–15 1999, pp. 202–216.

[148] K. Sohrabi, J. Gao, V. Ailawadhi, and G. J. Pottie, "Protocols for self-organization of a wireless sensor network," *IEEE Personal Communications*, vol. 7, no. 5, pp. 16–27, October 2000.

[149] L. Song and D. Hatzinakos, "A cross-layer architecture of wireless sensor networks for target tracking," *IEEE/ACM Transactions on Networking*, vol. 15, no. 1, pp. 145–158, February 2007.

[150] E. Souto, G. Guimarães, G. Vasconcelos, M. Vieira, N. Rosa, C. Ferraz, and J. Kelner, "Mires: a publish/subscribe middleware for sensor networks," *Personal and Ubiquitous Computing*, vol. 10, no. 1, pp. 37–44, February 2005.

[151] (2006) sQualnet website. [Online]. Available: http://nesl.ee.ucla.edu/projects/squalnet/

[152] W. Stallings, *Operating Systems Internals and Design Principles*, 5th ed. Prentice-Hall, 2005.

[153] J. A. Stankovic, T. F. Abdelzaher, C. Lu, L. Sha, and J. C. Hou, "Real-time communication and coordination in embedded sensor networks," *Proceedings of the IEEE*, vol. 91, no. 7, pp. 1002–1022, July 2003.

[154] D. C. Steere, A. Baptista, D. McNamee, C. Pu, and J. Walpole, "Research challenges in environmental observation and forecasting systems," in *Proc. 6th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'00)*, Boston, MA, USA, August 6–11 2000, pp. 292–299.

[155] M. Storey, G. Blair, and A. Friday, "MARE: resource discovery and configuration in ad hoc networks," *Mobile Networks and Applications*, vol. 7, pp. 377–387, October 2002.

[156] W. Su, Özgür B. Akan, and E. Cayirci, *Communication Protocols for Sensor Networks*. Springer, 2004, pp. 21–50.

[157] J. Suhonen, M. Kohvakka, M. Hännikäinen, and T. D. Hämäläinen, "Design, implementation, and experiments on outdoor deployment of wireless sensor network for environmental monitoring," in *Embedded Computer Systems: Architectures, Modeling, and Simulation, 6th International Workshop, SAMOS 2006*, ser. Lecture Notes in Computer Science, S. Vassiliadis, S. Wong, and T. D. Hämäläinen, Eds. Springer, 2006, vol. 4017, pp. 109–121.

[158] (2001) Jini architecture specification, version 1.2. Sun Microsystems. [Online]. Available: http://www.sun.com/software/jini/

[159] (2002) Java message service specification - version 1.1. Sun Microsystems. [Online]. Available: http://java.sun.com/products/jms/

[160] (2003) Java remote method invocation specification. Sun Microsystems. [Online]. Available: http://java.sun.com/javase/technologies/core/basic/rmi/

[161] (2007) SunSPOTWorld website. [Online]. Available: http://www.sunspotworld.com/

[162] S. Sundresh, K. WooYoung, and A. Gul, "SENS: a sensor, environment and network simulator," in *Proc. 37th Annual Simulation Symposium (ANSS'04)*, Arlington, VA, USA, April 18–22 2004, pp. 221–228.

[163] D. Sweetser, V. Sweetser, and J. Nemeth-Johannes, "A modular approach to IEEE-1451.5 wireless sensor development," in *Proc. 2006 IEEE Sensors Applications Symposium (SAS'06)*, Houston, TX, USA, February 7–9 2006, pp. 82–87.

[164] R. Szewczyk, A. Mainwaring, J. Polastre, J. Anderson, and D. Culler, "An analysis of a large scale habitat monitoring application," in *Proc. 2nd International Conference on Embedded Networked Sensor Systems (SenSys'04)*, Baltimore, MD, USA, November 3–5 2004, pp. 214–226.

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

[166] (2007) Telelogic TAU SDL suite website. [Online]. Available: http://www.telelogic.com/products/tau/sdl/

[167] K. Terfloth, G. Wittenburg, and J. Schiller, "FACTS - a rule-based middleware architecture for wireless sensor networks," in *Proc. 1st International Conference on Communication System Software and Middleware (Comsware'06)*, New Delhi, India, January 8–12 2006.

[168] S. Tilak, N. B. Abu-Ghazaleh, and W. Heinzelman, "A taxonomy of wireless micro-sensor network models," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 6, no. 2, pp. 28–36, April 2002.

[169] B. L. Titzer, D. K. Lee, and J. Palsberg, "Avrora: scalable sensor network simulation with precise timing," in *Proc. 4th international Conference on information Processing in Sensor Networks (IPSN'05)*, Los Angeles, CA, USA, April 24-27 2005.

[170] G. Tolle, J. Polastre, R. Szewczyk, D. Culler, N. Turner, K. Tu, S. Burgess, T. Dawson, P. Buonadonna, D. Gay, and W. Hong, "A macroscope in the redwoods," in *Proc. 3rd International Conference on Embedded Networked Sensor Systems (SenSys'05)*, San Diego, CA, USA, November 2–4 2005, pp. 51–63.

[171] V. Turau, M. Witt, and C. Weyer, "Analysis of a real multi-hop sensor network deployment: The heathland experiment," in *Proc. 3rd International Conference on Networked Sensing Systems (INSS'06)*, Chicago, IL, USA, May 31–June 2 2006.

[172] T. van Dam and K. Langendoen, "An adaptive energy-efficient mac protocol for wireless sensor networks," in *Proc. 1st International Conference on Embedded Networked Sensor Systems (SenSys'03)*, Los Angeles, CA, USA, November 5–7 2003, pp. 171–180.

[173] M. Varshney, D. Xu, M. Srivastava, and R. Bagrodia, "SenQ: a scalable simulation and emulation environment for sensor networks," in *Proc. 6th international Conference on information Processing in Sensor Networks (IPSN'07)*, Cambridge, MA, USA, April 25-27 2007, pp. 196–205.

[174] S. Vinoski, "CORBA: integrating diverse applications within distributed heterogeneous environments," *IEEE Communications Magazine*, vol. 35, no. 2, pp. 46–55, February 1997.

[175] P. Völgyesi and Ákos Lédeczi, "Component-based development of networked embedded applications," in *Proc. 28th Euromicro Conference (EUROMICRO'02)*, Dortmund, Germany, September 4–6 2002, pp. 68–73.

[176] G. Wagenknecht, D. Dietterle, J.-P. Ebert, and R. Kraemer, "Transforming protocol specifications for wireless sensor networks into efficient embedded system," in *Wireless Sensor Networks, 3rd European Workshop, EWSN 2006*, ser. Lecture Notes in Computer Science, K. Römer, H. Karl, and F. Mattern, Eds. Springer, 2006, vol. 3868, pp. 228–243.

[177] Q. Wang, Y. Zhu, and L. Cheng, "Reprogramming wireless sensor networks: challenges and approaches," *IEEE Network*, vol. 20, no. 3, pp. 48–55, May/June 2006.

[178] N. Wanga, N. Zhangb, and M. Wang, "Wireless sensors in agriculture and food industry - recent development and future perspective," *Computers and Electronics in Agriculture*, vol. 50, no. 1, pp. 1–14, January 2006.

[179] M. Weiser, "Hot topics: ubiquitous computing," *IEEE Computer*, vol. 26, no. 10, pp. 71–72, October 1993.

[180] ——, "The computer for the 21st century," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 3, no. 3, pp. 3–11, July 1999.

[181] M. Welsh and G. Mainland, "Programming sensor networks using abstract regions," in *Proc. 1st Symposium on Networked Systems Design and Implementation (NSDI'04)*, San Francisco, CA, USA, March 29–31 2004.

[182] Y. Wen, R. Wolski, and G. Moore, "DiSenS: scalable distributed sensor network simulation," in *Proc. 12th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'07)*, San Jose, CA, USA, March 14–17 2007, pp. 24–34.

[183] (2007) Wi-Fi alliance website. [Online]. Available: http://www.wi-fi.org/

[184] (2007) Wibree website. [Online]. Available: http://www.wibree.com/

[185] A. Willig, K. Matheus, and A. Wolisz, "Wireless technology in industrial networks," *Proceedings of the IEEE*, vol. 93, no. 6, pp. 1130–1151, June 2005.

[186] (2007) WiMAX forum website. [Online]. Available: http://www.wimaxforum.org

[187] W. Wolf, *Computers as Components: Principles of Embedded Computing System Design*. Morgan Kaufmann Publishers, 2001.

[188] A. Woo, S. Madden, and R. Govindan, "Networking support for query processing in sensor networks," *Communications of the ACM*, vol. 47, no. 6, pp. 47–52, June 2004.

[189] (2007) Worldsens website. [Online]. Available: http://www.worldsens.net

[190] (2007) accsense website. [Online]. Available: http://www.accsense.com/

[191] (2007) Aginova inc. website. [Online]. Available: http://www.aginova.com/

[192] (2007) Archrock website. [Online]. Available: http://www.archrock.com/

[193] (2007) BTnodes website. [Online]. Available: http://www.btnode.ethz.ch/

[194] (2007) chip45 website. [Online]. Available: http://www.chip45.com/

[195] (2007) Cirronet website. [Online]. Available: http://www.cirronet.com/

[196] (2007) Crossbow technology website. [Online]. Available: http://www.xbow.com/

[197] (2007) Dust networks website. [Online]. Available: http://www.dustnetworks.com/

[198] (2007) Ember corporation website. [Online]. Available: http://www.ember.com/

[199] (2007) Emerson process management - smart wireless website. [Online]. Available: http://www.emersonprocess.com/smartwireless/

[200] (2007) Grape networks website. [Online]. Available: http://www.grapenetworks.com/

[201] (2007) Libelium website. [Online]. Available: http://www.libelium.com/

[202] (2007) MAXFOR technology inc. website. [Online]. Available: http://www.maxfor.co.kr/

[203] (2007) Millennial net website. [Online]. Available: http://www.millennial.net/

[204] (2007) Moteiv website. [Online]. Available: http://www.moteiv.com/

[205] (2007) Sensicast website. [Online]. Available: http://www.sensicast.com/

[206] (2007) Sensinode website. [Online]. Available: http://www.sensinode.com/

[207] (2007) Telegesis website. [Online]. Available: http://www.telegesis.com/

[208] (2007) TinyNode website. [Online]. Available: http://www.tinynode.com/

[209] (2007) WiSuite website. [Online]. Available: http://www.wisuite.com/

[210] H. Wu, Q. Luo, P. Zheng, and L. M. Ni, "VMNet: Realistic emulation of wireless sensor networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 2, pp. 277–288, February 2007.

[211] L. Xiaohua, Z. Kougen, P. Yunhe, and W. Zhaohui, "A TCP/IP implementation for wireless sensor networks," in *Proc. IEEE International Conference on Systems, Man and Cybernetics (SMC'04)*, vol. 7, Hague, Netherlands, October 10–13 2004, pp. 6081–6086.

[212] Z. Xiong, A. D. Liveris, and S. Cheng, "Distributed source coding for sensor networks," *IEEE Signal Processing Magazine*, vol. 21, no. 5, pp. 80–94, September 2004.

[213] N. Xu, S. Rangwala, K. K. Chintalapudi, D. Ganesan, A. Broad, R. Govindan, and D. Estrin, "A wireless sensor network for structural monitoring," in *Proc. 2nd International Conference on Embedded Networked Sensor Systems (SenSys'04)*, Baltimore, MD, USA, November 3–5 2004, pp. 13–24.

[214] J. Yannakopoulos and A. Bilas, "CORMOS: a communication-oriented runtime system for sensor networks," in *Proc. 2nd European Workshop on Wireless Sensor Networks (EWSN'05)*, Istanbul, Turkey, January 31–February 2 2005, pp. 342–353.

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

[216] W. Ye, J. Heidemann, and D. Estrin, "Medium access control with coordinated adaptive sleeping for wireless sensor networks," *IEEE/ACM Transactions on Networking*, vol. 3, no. 12, pp. 493–506, June 2004.

[217] Y. Yu, B. Hong, and V. K. Prasanna, "Communication models for algorithm design in networked sensor systems," in *Proc. 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05)*, Denver, CO, USA, April 4–8 2005.

[218] Y. Yu, B. Krishnamachari, and V. K. Prasanna, "Issues in designing middleware for wireless sensor networks," *IEEE Network*, vol. 1, no. 18, pp. 15–21, January/February 2004.

[219] Y. Yu and V. K. Prasanna, "Energy-balanced task allocation for collaborative processing in networked embedded systems," *Mobile Networks and applications*, vol. 1-2, no. 10, pp. 115–131, February 2005.

[220] (2007) Z-wave alliance website. [Online]. Available: http://www.z-wavealliance.org/

[221] P. Zhang, C. M. Sadler, S. A. Lyon, and M. Martonosi, "Hardware design experiences in zebranet," in *Proc. 2nd International Conference on Embedded networked sensor systems (SenSys'04)*, Baltimore, MD, USA, November 3–5 2004, pp. 227–238.

[222] J. Zhu, S. Papavassiliou, and J. Yang, "Adaptive localized qos-constrained data aggregation and processing in distributed sensor networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 17, no. 9, pp. 923–933, September 2006.

[223] *ZigBee Specification*, ZigBee Alliance Std. r13, 2006.

PUBLICATIONS

PUBLICATION 1

M. Kuorilehto, M. Hännikäinen, T. D. Hämäläinen, "Rapid Design and Evaluation Framework for Wireless Sensor Networks," *Ad Hoc Networks*, accepted, DOI: 10.1016/j.adhoc.2007.08.003.

Available online at www.sciencedirect.com
ScienceDirect
ELSEVIER
Ad Hoc Networks xxx (2007) xxx–xxx
Ad Hoc Networks
www.elsevier.com/locate/adhoc

Rapid design and evaluation framework for wireless sensor networks

Mauri Kuorilehto [a,*], Marko Hännikäinen [b], Timo D. Hämäläinen [b]

[a] Nokia Technology Platforms, Visiokatu 3, FIN 33720, Tampere, Finland
[b] Tampere University of Technology, Institute of Digital and Computer Systems, Korkeakoulunkatu 1, FIN 33720, Tampere, Finland

Received 29 October 2004; received in revised form 12 July 2006; accepted 21 August 2007

Abstract

The diversity of applications and typically scarce node resources set very tight constraints to Wireless Sensor Networks (WSN). It is not possible to fulfill all requirements with a general purpose WSN, for which reason the rapid development of application specific WSNs is preferred. We present a new framework called WIreless SEnsor NEtwork Simulator (WISENES) for the design, simulation, and evaluation of WSNs. The target WSN is designed in Specification and Description Language (SDL), simulated in WISENES, and implemented on target platform either through automatic code generation or manually. The high-level WSN model is back-annotated with the measured values from a real platform. In this way, very accurate WSN simulations can be performed with a rapid design cycle. WISENES itself has been verified with TUTWSN (Tampere University of Technology Wireless Sensor Network) and ZigBee protocols. The MAC protocol of ZigBee was designed in two weeks from scratch by one designer, which shows the effectiveness of WISENES. For accuracy comparison, the results show 6.7% difference between the modeled and measured TUTWSN prototype power consumption. WISENES hastens the evaluation of new protocol and application configurations, especially for the large scale and long-term WSN deployments.

Keywords: Wireless Sensor Network (WSN); Network simulation; Medium access control (MAC); Energy efficiency

1. Introduction

Wireless Sensor Network (WSN) applications are diverse ranging from toys to military systems. Typical challenges for WSN are large scale, constantly changing network topology, and error prone communications, while in WSN nodes processing and storage capacities, as well as energy resources are limited. Most often WSNs are demanded to be robust against environmental strains, and able to autonomously recover from error situations. Further, depending on the applications and the interaction with environment, time synchronization and security requirements can be strict [1,2].

Opposite to general expectations, an all-purpose WSN is not a reasonable goal, since it is impossible

* Corresponding author. Tel.: +358 71 800 8000; fax: +358 3 3115 4561.
E-mail addresses: mauri.kuorilehto@nokia.com (M. Kuorilehto), marko.hannikainen@tut.fi (M. Hännikäinen), timo.d.hamalainen@tut.fi (T.D. Hämäläinen).

1570-8705/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.adhoc.2007.08.003

to meet all the real life constraints simultaneously. Instead, WSN protocol layers and their configuration parameters must be tailored to meet the specific application requirements. However, the design space is very large and makes the design automation the most important challenge for real working WSNs. A designer simply cannot handle all the parameters, functions, and their complicated dependencies without a tool support.

Prototyping can be applied to a single node functionality and small scale WSN testing. However, prototypes are not applicable for verifying the operation of e.g. a thousand-node network during a five year deployment. Even moderate sized networks benefit from extensive simulations, but the accuracy of simulation is very important. According to our experiences on real WSNs, the smallest and a minor-looking issue might cause severe changes for example in the network power consumption. Therefore, the accuracy of the design and performance estimations is not an option but essential for any real WSN.

Several legacy computer network simulators exist for the testing and modeling of communication protocols, but they omit WSN specific aspects. Proposed WSN simulators vary in their implementation, scale, and in the accuracy and coverage of the modeling of node platforms, protocols, and real world phenomena. Common features are the models for dedicated platforms, sensing, and wireless networking. However, none of the previous WSN simulators offers a complete and seamless design flow from abstract sketching to the real implementation.

Our WIreless SEnsor NEtwork Simulator (WISENES) framework is the first tool that enables the design, simulation, implementation and evaluation of WSNs with measured back-annotated information. WISENES is targeted to the design of deployable, real WSN networks. The main difference to the other proposed frameworks is that there is no need to carry out a separate high abstraction WSN modeling project and another development project for the actual implementation. Instead, WISENES supports all phases in the design flow. However, if preferred, WISENES can also be used for the plain simulations like other WSN simulators. In all cases, WISENES eases the assessment of the protocol and application interoperability, and the evaluation their applicability for different sensor node platforms.

The key benefit of WISENES is that the evaluation of protocols, applications, and their different configurations is carried out starting from the design phase. The framework defines rules and interfaces for a designer to the protocol stack and application implementation. The functionality, type, or composition of the protocols is not limited by the framework. Sensor nodes, transmission medium, and inspected phenomena are modeled separately in the WISENES framework. The WSN protocols and applications are implemented in Specification and Description Language (SDL) [3]. The models of high abstraction level SDL are compiled to executables used for both simulation and final implementation. Unlike in the other WSN simulators, target node platforms are not restricted to a specific pre-defined platform.

WISENES has been tested and its own performance evaluated with large TUTWSN (Tampere University of Technology Wireless Sensor Network) [4] and ZigBee networks [5]. For the evaluation of WISENES accuracy, real and simulated TUTWSNs are compared. However, the comparison of different WSNs themselves is not the primary scope of this paper.

This paper is organized as follows. Section 2 discusses the related work in the area of WSN simulation and presents the comparison of WISENES with the other WSN simulators. WISENES design is presented in Section 3 and Section 4 introduces the WISENES framework. The use of WISENES for TUTWSN and ZigBee protocol implementation is presented in Section 5. Section 6 gives the evaluation of WISENES, and the TUTWSN and ZigBee simulation results. Finally, conclusions are given and future work projected.

2. Related work

Legacy computer network simulators, such as ns-2 [6], GloMoSim [7], Qualnet [8], OPNET [9], OMNeT++ [10], Scalable Simulation Framework (SSF) [11], and J-Sim [12] enable the simulation of wireless network behavior and protocol stack operation but lack accounting for WSN characteristics. This is overcome in the simulators proposed specifically for WSNs, which we have categorized to *networking oriented* and *sensor node* simulators. The networking oriented simulators model the transmission medium in detail and are more suitable for the large scale WSN simulations. The sensor node simulators mainly simulate the operation of a single node but implement a lightweight communication model. Currently, there exist eleven relevant proposals for the networking oriented WSN simulators

Please cite this article in press as: M. Kuorilehto et al., Rapid design and evaluation framework for ..., Ad Hoc Netw. (2007), doi:10.1016/j.adhoc.2007.08.003

and six proposals targeting to the sensor node modeling. These are compared with WISENES in Section 2.3.

2.1. Networking oriented simulators for WSN

Most of the proposed networking oriented simulators are based on the legacy computer network simulators. SensorSim [13] and Naval Research Laboratory's (NRL) sensor network simulator [14] extend ns-2 with general WSN features. sQualnet [15] is built on top of Qualnet and Simulator for Wireless Ad-hoc Networks (SWAN) [16] is based on SSF. SENSIM [17] and simulation template for EYES [18] utilize OmNet++ environment, while J-Sim sensor simulator [19] adds WSN features to its parent simulator. VisualSense [20] is an extension to Ptolemy II [21], Prowler [22] utilizes MATLAB, and H-MAS [23] and SENSE [24] implement custom simulation environments.

The most realistic transmission media and protocol stacks are available in SensorSim, NRL simulator, sQualnet, and J-Sim sensor simulator. While the first two rely on the models available in parent simulators, the last two include also a set of WSN protocols. VisualSense has several models for transmission medium, which vary in their accuracy. SWAN and Prowler include abstracted transmission media and lowest layer protocol models that estimate the network operation. SENSIM, EYES simulator, H-MAS, and SENSE have error free transmission medium models, in which the signal propagation is dependent only on the transmission range. Simple protocol stacks are available for SENSIM, H-MAS, and SENSE. In VisualSense and EYES simulator, the protocol stack is implemented by the designer.

A separate sensing channel containing also the sensed targets is used for phenomena modeling in SensorSim, NRL simulator, sQualnet, and J-Sim sensor simulator. VisualSense has also a dedicated channel for modeling the propagation of different phenomena. Of the other related simulators, SWAN models catastrophic plume dispersion, and H-MAS generates random sensor readings. Prowler, SENSIM, EYES simulator, and SENSE do not support phenomena sensing.

Concerning the node platform capabilities, only the power consumption is accounted in the related simulators. SensorSim, sQualnet, SENSE, SENSIM, and J-Sim sensor simulator have detailed power models, which consider battery discharge rate and relaxation. NRL simulator and EYES simulator use linear battery models, in which the maximum energy capacity is available independent on the discharge rate. Prowler and VisualSense estimate the power consumption based on activity, whereas in SWAN and H-MAS power consumption is not taken into account.

The simulated code is applicable directly for hardware platforms in SensorSim, sQualnet, J-sim sensor simulator, and partly in VisualSense. In SensorSim, simulated SensorWare applications are compatible with custom hardware platforms [25]. The other three simulators enable the execution of applications, and sQualnet also higher layer protocols, on Berkeley motes [26] on top of well-known WSN Operating System (OS) TinyOS [27].

VisualSense uses a graphical notation for the design and supports a combination of different Models of Computation (MoC) of Ptolemy II. Abstracted application scripts can be simulated also in Prowler.

2.2. Sensor node simulators

Most of the proposed sensor node simulators are targeted to TinyOS motes. Complete TinyOS system can be simulated with TinyOS SIMulator (TOSSIM) [28], ATmel EMUlator (ATEMU) [29], and TinyOS Scalable Simulation Framework (TOSSF). TOSSF itself is an extension to SWAN [30]. SENS [31] supports only TinyOS application simulation. EmSim implements a simulation environment for custom Em* Linux applications and protocols [32]. Sensor Network Asynchronous Processor (SNAP) [33] is a hardware emulator, which connects several processors on a Network-on-Chip (NoC).

The TOSSIM transmission medium model is directed graphs with individual bit error rates, whereas in ATEMU and EmSim the transmission medium is error free accounting only the transmission range. TOSSF utilizes a transmission medium model from SWAN and in SENS transmission medium and networking protocols are combined into a simple model. In SNAP, NoC models the transmission medium. Protocol stacks in TOSSIM, ATEMU, and TOSSF are dependent on TinyOS configuration. EmSim and SNAP implement simple protocols for the system testing.

Phenomena sensing is modeled in TOSSF by the plume dispersion model of SWAN. TOSSIM and EmSim retrieve a sensor reading from an external

Please cite this article in press as: M. Kuorilehto et al., Rapid design and evaluation framework for ..., Ad Hoc Netw. (2007), doi:10.1016/j.adhoc.2007.08.003

source. SENS incorporates a separate environment model that supports sensing and actuating. In SNAP and ATEMU phenomena are not modeled. None of the simulators models power consumption.

The applications from all sensor node simulators, and protocols from other than SENS, can be directly mapped to the hardware platforms. However, the platform is restricted to a specific one.

2.3. Comparison of WISENES with related simulators

The comparison of WISENES and the other WSN simulators is summarized in Table 1. The comparison is based on the public information available about each simulator and the possible parent simulator engine. If exact scalability information is not available, the simulator is assessed according to the largest reported simulations.

The scope of input parameterization is vital when comparing the configurability of simulators to different kinds of platforms, protocols, and applications. Also, the availability of Graphical User Interfaces (GUI) and the type of information output by the simulator are accounted in the comparison. The possibility to use simulated protocols and applications for the final implementations on physical platforms defines the applicability of a simulator as a complete design and development environment.

The term *accurate results* denotes a very close correspondence of the simulation results to the real world measurements with physical WSN prototypes. Accurate results in full-scale simulations need to combine at least realistic models for communications (application, transmission medium, transceiver unit, and low-level communication protocols) and node platform (energy, memory, peripheral I/O, and computation). The node state changes, peripheral activation, and leakage currents contribute to the accuracy of energy consumption. The memory allocation and thread scheduling in simulator depend on the accuracy of the OS model of the simulator.

As shown in the table, most of the simulators are capable of simulating WSN scenarios consisting of thousands of nodes. This is an acceptable limit for the current WSNs, but in future the capability to simulate networks with in order of magnitude larger scale is required.

Major differences between the simulators are in input and output parameterization. Although sensor node simulators emulate a single node platform in detail, they do not allow the testing and evaluation of applications and protocols on other types of sensor nodes. Further, they omit the modeling of node power consumption. From the simulators, WISENES incorporates the most comprehensive modeling of node platforms, which allows a detailed description for virtually any real node platform. In addition to power consumption, which is considered also in several other simulators, WISENES accounts memory and computation capacities. From the related simulators, the possibility to define different platforms is available only in sQualnet, but its modeling is not as detailed as that of WISENES.

Most of the simulators visualize network topology and key parameters through GUI. The event information is gathered to trace or log files with varying level of detail. From the related simulators, ns-2 based simulators output extensive trace files. Compared to the other simulators, WISENES outputs extensive event and data statistics but in addition also detailed information about the node resource usage. Further, in WISENES a designer can add own log items e.g. to obtain the values of a desired parameters during the simulated period.

The simulated code is directly applicable for node platforms in sensor node simulators. In networking oriented simulators, two approaches are taken for the final implementation. Either the simulated protocols and applications are converted to executables for node platforms, or an existing code library is used for emulating a node in a large scale simulation. In the latter, the node implementation already exists, and only the configuration parameters for the final implementation are acquired by the simulations. WISENES is the only simulator that supports both of these. The SDL generated C from WISENES with a custom lightweight kernel is also applicable for the resource constrained sensor nodes [34].

A rapid protocol evaluation for a specific application is possible only if the simulator protocol stack is modular and its layers interchangeable. Most of the simulators that descend from a legacy computer network simulator incorporate a modular protocol stack. In WISENES, the protocol layers communicate through pre-defined interfaces, which allow the replacement of simulated protocols at the different layers. Graphical design of protocols and applications is possible only in WISENES and VisualSense. In both simulators the high abstraction level designs can also be embedded to node platforms. However, WISENES provides significantly more accurate results compared to VisualSense.

Please cite this article in press as: M. Kuorilehto et al., Rapid design and evaluation framework for ..., Ad Hoc Netw. (2007), doi:10.1016/j.adhoc.2007.08.003

Table 1
Comparison of WISENES and existing WSN simulators

Simulator	Simulator engine	Scalability	Simulator input	Simulator output	Final implementation	Benefits	Deficiencies
WISENES	Extended telelogic TAU SDL	~10,000 [34]	Nodes, protocols, applications, mediums (XML)	GUI, log files, (data, energy, memory, CPU, errors)	SDL code generation/C modules directly	Graphical design, accurate results, modular, scalability, back-annotation	Sensing channel
Networking oriented simulators							
SensorSim	ns-2	~2000 [17]	Power model, protocols (TCL)	ns-2 nam UI, trace files (data, energy, errors)	Applications for SensorWare, ns-2 protocols	Accurate results, variety of protocols (ns-2), modular	No memory and CPU modeling
sQualnet	Qualnet	~10,000	Nodes, traffic, protocols (scripts)	Qualnet Visualizer, statistics files (data, energy)	nesC applications directly	Accurate results, integration to HW, modular, scalability	No memory and CPU modeling in simulator
NRL simulator	ns-2	~2000 [17]	Nodes, protocols, sensing (TCL)	ns-2 nam UI, trace files (data, energy, errors)	ns-2 protocols	Variety of protocols (ns-2), modular	No memory and CPU modeling
SWAN	DaSSF	~10,000	Nodes, plume dispersion (DML)	GUI, system printouts (data counters, delay)	WiroKit routing protocol directly	Scalability	Inaccurate medium model, no node and sensing modeling, no modularity
SENSIM	OmNet++	~5000	Protocols (ini-file (for OmNet++)	OmNet++ GUI (data)	None	Modular	No sensing, memory, and CPU modeling
EYES simulator	OmNet++	<1000	Protocols (ini-file (for OmNet++)	OmNet++ GUI (data,errors)	None	Modular	Inaccurate energy and medium modeling, no memory and CPU models
J-Sim sensor simulator	J-Sim	>1000 [35]	Protocols (script)	Text output, GUI possible, (data)[(1)]	Applications directly	Modular	No GUI, no memory and CPU modeling
VisualSense	Ptolemy II	~100	MoC configurations (Ptolemy II)	Ptolemy II GUI (topology, node information)	Algorithms integrated to TinyOS [36]	Graphical design, algorithm integration	No protocol stack, nodes modeled by power model
Prowler	MATLAB	<1000	Application (script)	GUI (data)	None	MATLAB for algorithm testing	Inaccurate protocol, node, and medium modeling
H-MAS	Custom	>100	Nodes, protocols (text)	GUI, event files (data)	None		Inaccurate protocol and medium modeling, no modularity, scalability
SENSE	Custom	~1000	Topology, traffic (script)	None[(2)]	None	Modular	No output, inaccurate medium modeling, no node and sensing modeling

(*continued on next page*)

Please cite this article in press as: M. Kuorilehto et al., Rapid design and evaluation framework for ..., Ad Hoc Netw. (2007), doi:10.1016/j.adhoc.2007.08.003

Table 1 (*continued*)

Simulator	Simulator engine	Scalability	Simulator input	Simulator output	Final implementation	Benefits	Deficiencies
Sensor node simulators							
TOSSIM	Custom	~10,000	TinyOS code	TinyViz, debug (data, node)	Directly	Applicability of code for mores	No energy modeling, same code for all nodes
ATEMU	Custom	~100	TinyOS code network, nodes (XML)	XATDB debugger (debug data)	Directly	Applicability of code for mores	No sensing and energy models, scalability
SENS	Custom	~10,000	Node profiles (text)	GUI, text files (data, energy, errors)	Applications directly	Scalability	Only for applications, node and medium models
TOSSF	DaSSF (SWAN)	~10,000	TinyOS code, SWAN parameters	SWAN output(3)	Directly	Scalability	Inaccurate medium model, no node and sensing modeling
Em* EmSim	Custom	<100	Simulation case (script) [37]	Debug traces (node)	Directly	Applicability of code	Inaccurate medium model, no node and sensing models, scalability
SNAP	FPGA Emulator	~100(4)	Configuration (for FPGA)	FPGA debug interface (node)	Directly	Emulation	Inaccurate medium model, no sensing modeling, scalability

(1) J-Sim outputs simulation related data to an "instrument channel", to which user can implement a custom UI.
(2) No information about output is given, only results given show simulation times and simulator memory consumption.
(3) TOSSF I/O is not specified, but we assumed SWAN I/O due to the relation.
(4) SNAP emulators can be connected to increase scalability, but no evaluation is given.

Please cite this article in press as: M. Kuorilehto et al., Rapid design and evaluation framework for ..., Ad Hoc Netw. (2007), doi:10.1016/j.adhoc.2007.08.003

Generally, the networking oriented simulators are more suitable for the network-wide evaluation, whereas the strength of the sensor node simulators lies in the testing and optimization of a single node operation. From the related simulators, SensorSim and sQualnet implement the most comprehensive simulation environment. Compared to WISENES, their battery and sensing models are currently more accurate. The distinctive features of WISENES are the complete design flow from the high abstraction level graphical models to the final node implementation, the accurate full-scale simulations with configurable protocol stack and node platform models, and the back-annotation of performance information from real platforms.

3. Designing WSNs with WISENES

WISENES defines the rules and interfaces for the WSN design and provides a library that contains existing protocols and a set of implementations for known functions. A reference WSN protocol stack used in WISENES is depicted in Fig. 1 in correspondence to the OSI model [38].

The key layer for WSN topology creation, channel access, and power management is the Medium Access Control (MAC) protocol. The topology in WSNs is either clustered or flat. In a clustered topology, nodes are grouped to clusters, in which a central cluster headnode coordinates the networking and associated subnodes. In a flat topology all nodes are equal. A routing protocol creates multihop paths for the end-to-end communication. A middleware layer hides the heterogeneities of underlying protocol stacks and node platforms from applications [1].

The composition of the protocol stack is network and application dependent, thus all layers are not mandatory in WISENES. The transceiver unit at the physical layer is part of the WISENES framework and the lowest layer protocol sends data to the transmission medium through its interface.

Multiple WSN applications can be simulated simultaneously in WISENES. Applications are designed as a set of tasks communicating together. Tasks initiate sensing, perform data processing and aggregation, and initiate data transfers. The application layer, which implements a host environment for application tasks, is a part of the WISENES framework.

Applications are implemented either in detail using SDL or by a task graph. The task graph is a simple data dependency graph described by the simulator input parameters. In addition to the task data dependencies, it defines the task activation frequencies, and task sensing and data characteristics. This approach enables the testing of different types of applications with minimum effort.

3.1. WISENES input and output

The input parameter and output result groups of WISENES are summarized in Fig. 2. WISENES input parameters are defined using eXtensible Markup Language (XML), each parameter set having a dedicated file with a pre-defined structure.

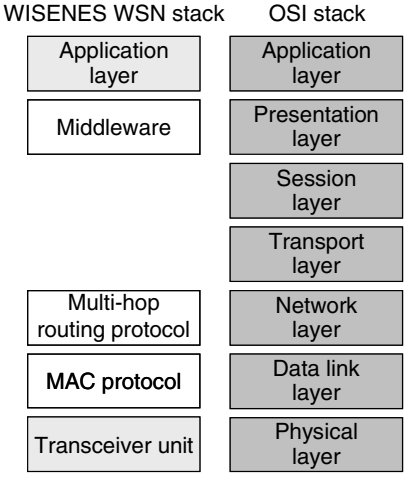


Fig. 1. WISENES WSN and OSI model protocol stacks.

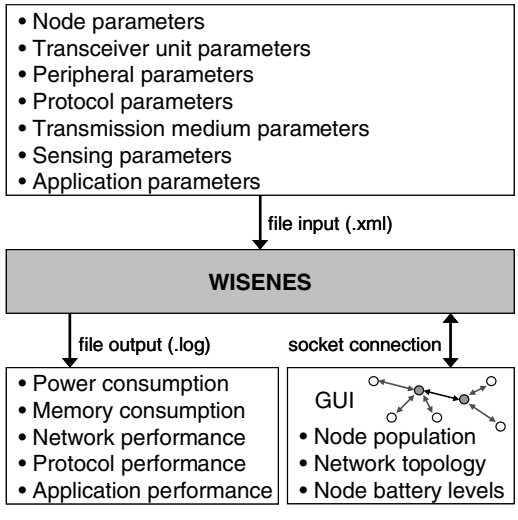


Fig. 2. WISENES input parameter groups and output results.

Please cite this article in press as: M. Kuorilehto et al., Rapid design and evaluation framework for ..., Ad Hoc Netw. (2007), doi:10.1016/j.adhoc.2007.08.003

Table 2
WISENES input parameters and their types

Parameter	Type	Description
Node parameters		
Execution capacity	Integer (MIPS)	Instructions CPU can execute in a second
Instruction memory size	Integer (instruction)	Available instruction memory
Data memory size	Integer (byte)	Available data memory
Power unit capacity	Float (mAh)	Initial energy capacity in node battery
Active state power consumption	Float (μW)	Consumed power in active state
Sleep state information	[Float, float] array (ms, μW)	Required idle time before activation and power consumption in sleep state
Harvesting capacity	Float (μW)	Power node can harvest from environment
Transceiver unit(s)	Integer array (identifier)	Transceiver unit(s) on the node
Peripherals	Integer array (identifier)	Peripherals attached to the node
Node coordinates	Float array (m)	Three-dimensional coordinates for node (x, y, z)
Transceiver unit parameters		
Throughput	Integer (bps)	Transceiver unit data rate
Start-up transient period	Float (ms)	Start-up transient time before receiver/transmitter is ready
Data loading information	[Integer, float] pair (bps, μW)	Throughput and power consumption during transceiver to CPU communication
Receiver power	Float (μW)	Power consumption while receiver is active
Transmit power levels	[Integer, float] array (dBm, μW)	Available transmit powers and corresponding power consumption
Carrier sensing power	Float (μW)	Power consumption during carrier sensing operation (if available in transceiver)
Peripheral parameters		
Type	Constant string (identifier)	Defines the peripheral type, e.g. sensor, ADC, location-finding system, mobilizer
Phenomena	Integer array (identifier)	Phenomena the peripheral is related to, if sensor
Relations	Integer array (identifier)	Relations to other peripherals
Activation time	Float (ms)	Time the peripheral is active once activated
Activation power	Float (μW)	Power consumption when peripheral is active
Protocol parameters (for each protocol layer)		
Instruction memory consumption	Integer (instruction)	Instruction memory required by the protocol layer
Data memory consumption	Integer array (byte)	Protocol static and dynamic data memory consumption
Transmission medium parameters		
Signal attenuation curve	Float array (constant)	Define signal attenuation curve coefficients (k, b, v)
Minimum PER	Float (constant)	Minimum PER in the transmission medium
Sensing parameters		
Active phenomena	Integer array (identifier)	Phenomena that can inspected
Limits	Float array (dependent unit)	Lower and upper limits separately for each sensed phenomenon value
Application parameters		
Task activation interval	Float (ms)	The interval between task activations
Task data activation	Integer (constant)	After how many activations task initiates a data transfer
Task data amount	Integer (byte)	The amount of data sent by the task
Task data relations	Integer array (identifier)	Task to which the data is sent
Task peripheral relations	Integer array (identifier)	Peripherals required by the task
Instruction memory consumption	Integer (instruction)	Task instruction memory consumption
Data memory consumption	Integer array (byte)	Task static and dynamic data memory consumption
Task executed operations	Integer (constant)	Executed operations per task activation
Task population	Integer array (identifier)	A list of node identifiers defining the nodes, in which the task binary is located

Node parameters are given in two separate files. The first defines the capabilities of node platforms, and the other per-node platform type and node coordinates.

Please cite this article in press as: M. Kuorilehto et al., Rapid design and evaluation framework for ..., Ad Hoc Netw. (2007), doi:10.1016/j.adhoc.2007.08.003

WISENES outputs information in two forms. Detailed information about the simulated WSN and nodes is collected to log files. Each protocol and data event during a simulation is logged with the parameters that define the cause and the consequence of the event. Log data are written to per-node directories, each protocol layer, application, and a control instance modeling OS routines having a dedicated *.log*-file. During an active simulation run, the progress of a simulation is illustrated through GUI presenting the node population, network topology, and node energy level.

Input parameters at each group and their types are presented in detail in Table 2. Node, transceiver unit, and peripheral parameters define the capabilities of sensor node platforms. Protocol related input parameters define the static memory consumption for each protocol, while the rest of the characteristics are specified in the protocol SDL implementation. Transmission medium parameters define its modeling and sensing parameters active phenomena. An application task graph and an initial application task population are described in the application parameters.

Table 3 shows the output results and their types. As the events are logged as they occur, the momentary values of the presented results are stored into the logs. Depending on the type of the result, the values are also accumulated or averaged to obtain an overall knowledge about the system behavior.

3.2. WISENES user interfaces

WISENES has two UIs for controlling and monitoring simulation runs. Simulations are started and controlled through a command line interface. The progress of the simulation and the topology of a simulated WSN are visualized in WISENES GUI. GUI is implemented in Java using Java foundation classes Swing packages [39]. The communication between GUI and WISENES is implemented by a socket interface. A screenshot of GUI visualizing a hundred-node TUTWSN simulation is depicted in Fig. 3.

Table 3
WISENES output results and their types

Result	Type	Description
Power consumption (final and variation during time)		
Node, total	Float (μW)	Total power consumption in node
CPU	Float array (μW)	Power consumption in CPU in execution and different sleep states
Transceiver unit	Float array (μW)	Transceiver unit power consumption in sending, receiving, and scanning
Peripherals	Float (μW)	Peripheral power consumption separately for each
Protocols	Float array μW	Power consumption in the execution of different protocols (network vs. node)
Application tasks	Float array (μW)	Power consumption in the execution of application tasks (network vs. node)
Memory consumption (final and variation during time)		
Application tasks, instruction	Integer array (instructions)	Instruction memory consumption in application tasks per node
Application tasks, data	Integer array (byte))	Static and dynamic data memory consumption in application tasks per node
Protocols, data	Integer array (byte)	Static and dynamic data memory consumption in protocols per node
Network performance (averaged and variation during time)		
Throughput	Integer array (bps)	Throughput per link in the network
Delays	Float array (ms)	Transfer delays per link
PER	Float array (constant)	Packet error rate per link
Collisions	Integer	Number of collisions in a node
Protocol performance (averaged and variation during time)		
Delays	Float array (ms)	Delays due to buffering and control in different protocol layers
Buffering	Integer array (constant)	Buffer lengths in the protocol
Throughput utilization	Float array (constant)	Utilized vs. available throughput
Application performance (averaged and variation during time)		
Delays	Float array (ms)	Communication delays between tasks
Activation accuracy	Float array (ms)	The variance in task activation times vs. to expected activation times
Data coherence	Float array (constant)	The accuracy and sufficiency of provided data

Please cite this article in press as: M. Kuorilehto et al., Rapid design and evaluation framework for ..., Ad Hoc Netw. (2007), doi:10.1016/j.adhoc.2007.08.003

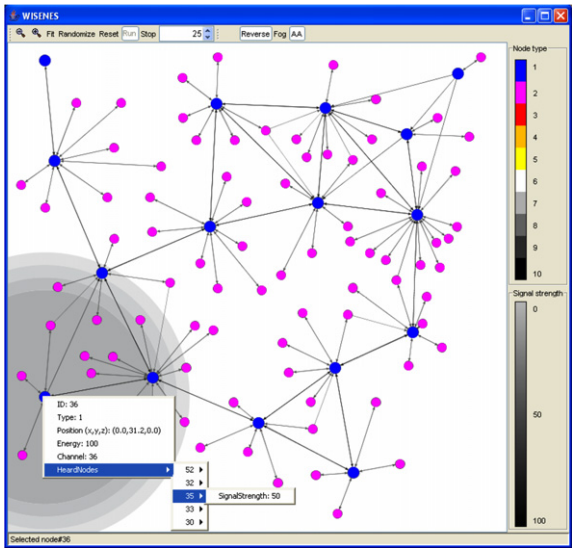


Fig. 3. WISENES GUI screenshot from a hundred-node TUT-WSN simulation.

Different node roles, in this case cluster headnodes and subnodes, are shown by different colors. Remaining energy levels and the received signal strengths of the active communication links are presented in a pop-up window when a node is selected. By double clicking a node, its transmission ranges with different transmit powers are illustrated. Furthermore, nodes can be moved by dragging and dropping them in GUI.

3.3. WISENES tools

A set of Tool Command Language (TCL) scripts is implemented for WISENES initiation and result handling. The initiation scripts facilitate a simulation case construction from different protocols and a random node population generation for large simulation cases. The possible relations between different types of platforms can be given as a parameter to the population generation script, e.g. in order to force ten low power nodes in the vicinity of a more powerful one.

The result handling scripts facilitate power consumption, data packet tracing, and link utility evaluations. The power consumption information is gathered from individual nodes and a listing defining detailed power characteristics for different components in each node is created. The packet tracing tracks the hops of a packet from its source to the destination node, and determines the delays on different hops and protocol layers. The link utility evaluates activity and congestion of node to node connections.

4. WISENES framework

The *WISENES framework* consists of models for transmission medium, sensing channel, sensor node, and of a central simulation control that manages simulations and handles simulator Input/Output (I/O). Networking protocols and node platform modeling are embedded to the sensor node model.

In addition to application tasks and protocols described by the designer, SDL is used for the implementation of the WISENES framework. The tool used for SDL development is Telelogic [40] TAU SDL Suite [41], version 4.5. The SDL suite uses a graphical notation for SDL design, and provides tools for simulation, integration, and implementation.

4.1. SDL introduction

SDL is used for designing systems ranging from general software to embedded applications. MoC in SDL is parallel communicating Extended Finite State Machines (EFSM). SDL hierarchy has multiple levels, of which the system level consists of a number of blocks that clarify the representation. They can be recursively divided into sub-blocks. The behavior of a block is implemented in processes described by EFSMs. The representation of a process can be simplified by implementing a part of the functionality in a procedure. Blocks and processes can be implemented using the type concept of SDL, which allows their instantiation. These type definitions can be included with other type definitions to SDL packages that facilitate modular system design. The maximum number of instantiated blocks must be defined at a compile time, whereas processes can be created dynamically during runtime [42].

Processes in a same or in different blocks communicate by asynchronous signals that can carry any number of parameters. Each process has an infinite First-In-First-Out (FIFO) buffer for incoming signals. Signal routes define which type of signals a process can send and to which processes. An outgoing signal is routed according to the signal route or a Process IDentifier (PID). Communication between processes can also be executed synchro-

Please cite this article in press as: M. Kuorilehto et al., Rapid design and evaluation framework for ..., Ad Hoc Netw. (2007), doi:10.1016/j.adhoc.2007.08.003

nously by calling remote procedures, which are exported on the process interface [42].

Due to its formality, SDL can be automatically converted for example to C source code, which can then be used to make an executable application or simulation. Telelogic TAU SDL simulation engine supports discrete event simulations and real-time simulations. In WISENES we utilize discrete event simulation, in which events are processed and handled in the order of occurrence. This makes the time concept fully parallel and avoids an active waiting during the idle times.

Environment functions are needed for the interaction between SDL and its execution environment. Dedicated functions are defined for environment initialization, unloading, signal output, and signal input. The output function is called when a signal is sent from the SDL system to the environment. Because a method for interruption is absent in SDL, the input function must be polled for receiving signals from the environment. In Telelogic TAU SDL simulation engine, the input function is called after every transaction, which is an execution flow from a state to another triggered by an incoming signal. An SDL procedure can be substituted by an external function, in a case where SDL lacks expressivity or a more efficient implementation is desired. Both environment and external functions are implemented in C for WISENES.

4.2. WISENES instantiation

The instantiation of WISENES is depicted in Fig. 4. The designer selects the protocols from the library or implements new ones in SDL and integrates them to the WISENES framework. The upper and lower interfaces of the protocol stack are the pre-defined interfaces to the application layer and transceiver unit, respectively. Application functionality is either implemented as SDL procedures or described by a task graph.

The protocol stack consists of data link, network, and middleware layers that are instances of block types implemented in SDL packages. The interfaces between the layers are fixed, but a layer can be bypassed, i.e. a network layer can communicate with the application layer at its upper interface. The internal implementation of layers is not restricted in any way.

Node platforms are parameterized in the XML configuration files that are parsed by *Central Simulation Control*. The parameters are passed to *Sensor Node* model. Node coordinates are relayed also to *Transmission Medium* and *Sensing Channel*, both of which have also dedicated parameters.

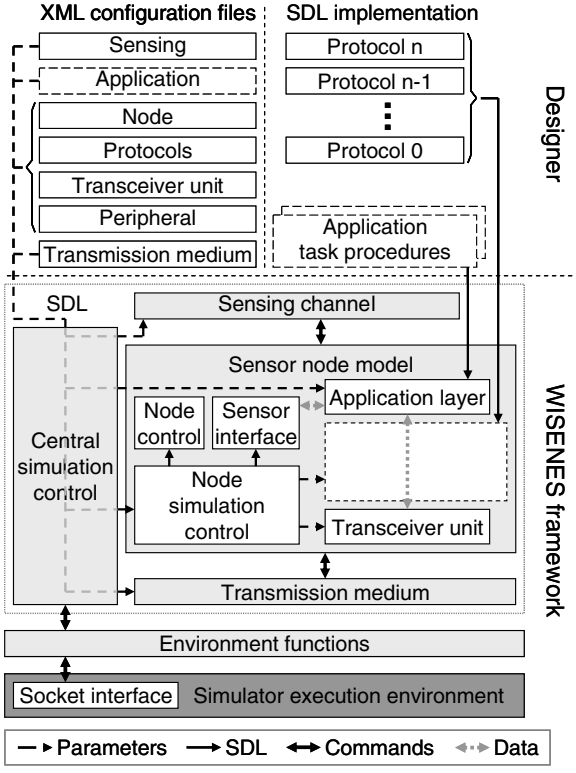


Fig. 4. The architecture of WISENES instantiation.

The interfacing of WISENES GUI is implemented in environment functions that maintain the socket connection. Information to GUI is updated only periodically in order to lessen communication. A data structure that defines sensor node parameters is sent to the environment functions as a signal parameter and parsed to the socket.

The SDL system of WISENES framework is illustrated in Fig. 5. The framework consists of SDL blocks that implement the main functional models and the central simulation control. The sensor node is a dynamic block of type *Node_Type*, and the number of its instances is specified by *NODE_COUNT*. The figure depicts also the signal routes between the blocks and a dedicated signal route to the environment.

4.3. Central simulation control

The central simulation control initiates the WISENES framework, controls active simulation

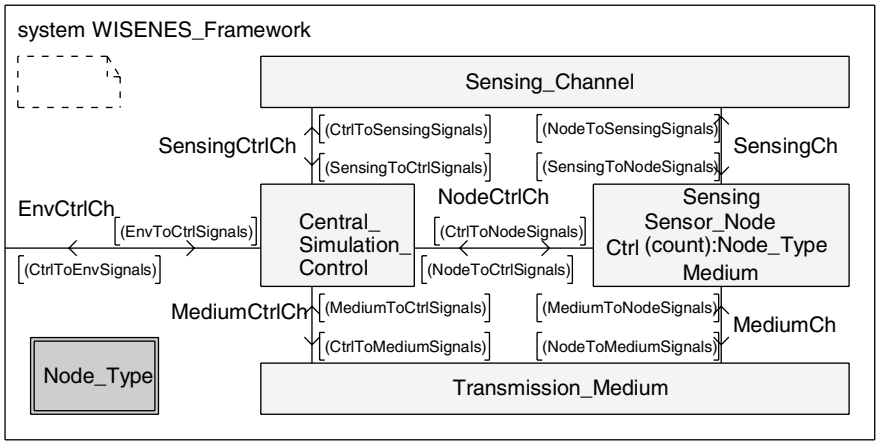


Fig. 5. The SDL system level of WISENES framework.

runs, and performs event logging. The information gathering for the control and logging is implemented in remote procedures that are presented with their parameters in Table 4.

The input parameters of *UpdateNodeInformation* procedure specify the node identifier, current battery level, role, and the connectivity to other nodes. The node role defines whether a node is a headnode, a subnode, or a sink. Sensor nodes call the procedure whenever any of the parameter values changes. The information is relayed to GUI and utilized for determining whether the simulation end condition is satisfied. The end condition is set by the designer and it defines a simulation time limit, a percentage of dead nodes, or e.g. a limit for an application task activation count.

Distinct remote procedures for event and data packet logging are exported for each protocol layer, application tasks, and for framework components. Their parameters vary depending on the layer. Logs are stored in dedicated data structures and written to files either periodically when WISENES memory consumption exceeds a pre-defined limit, or at the end of the simulation.

Table 4
The remote procedures exported by the central simulation control

Procedure	Parameters
UpdateNodeInformation	NodeId, NodeRole, BatteryState, TransceiverUnit, Connectivity
LogXxxEvent	NodeId, EventName, Cause, Consequence, ...
LogXxxData	NodeId, PacketId, DataLength, DataAction, ...

4.4. Transmission medium

The transmission medium model provides the connectivity between sensor nodes. It is implemented as an SDL process that redirects signals from a source to the destination sensor node SDL blocks. Sensor nodes register their node identifier and transceiver unit PID to the transmission medium for enabling the data redirecting. Due to the nature of SDL data typing, transmitted data are separately copied for each destination node.

The signal propagation in the transmission medium is based on the transceiver unit dependent signal attenuation. The curve S defining the Packet Error Rate (PER) is

$$S = kd - \left(b + \frac{P}{v}\right), \tag{1}$$

where d is the distance between the source and destination nodes (m), and P is the transmit power (dBm). Constants k, b, v are derived from the measured signal attenuation curve. PER for a packet is

$$\mathrm{PER} = \begin{Bmatrix} 1, & \text{if } S \geqslant 1 \\ S, & \text{if } L < S < 1 \\ L, & \text{if } S \leqslant L \end{Bmatrix}, \tag{2}$$

where L is the lower limit for the PER.

In order to realistically model the hidden node problem and collisions, S is calculated separately for each node within the coverage of the transmission. The transmit power is specified by the source node. After the PER evaluation, a random number $0 \leqslant r < 1$ is generated again separately for each node. A transmission is successful, if $r > \mathrm{PER}$. If

$S > 1$, the signal attenuates during the transmission, otherwise the signal is relayed to the destination. If $S < 1$ and $r <$ PER, the unsuccessful transmission is indicated to the recipient but the copying of the actual data is not performed.

A delay during a transmission is calculated by dividing the packet length by the transceiver unit throughput. The delay of the signal propagation in the medium is omitted. Thus, a packet is relayed to the destinations immediately after the transfer delay.

4.5. Sensing channel

The sensing channel model simulates physical phenomena. Similarly to the transmission medium, the sensing channel utilizes node coordinates. Each phenomenon is modeled separately with individual propagation characteristics. The propagation depends also on the media in the vicinity.

Our current sensing channel implementation generates random stimuli for phenomena, except for the location queries that return node coordinates. The upper and lower limits are defined for each phenomenon. Currently simulated phenomena are temperature, humidity, vibration, sound, luminance, and location information. The selected approach is applicable for environmental monitoring, but a more detailed sensing channel must be implemented for e.g. object tracking applications.

4.6. Sensor node

The sensor node SDL block implementing the node model is depicted in Fig. 6. On the sensor node model, *Physical layer*, *Sensor Interface*, *Application Layer*, and *Node Control* blocks are part of the WISENES framework, while the instantiated protocol layers are selected or implemented by the designer. The signal routes between the protocol layers are for data communications, whereas the signals sent from the node control to the other blocks are for the initiation and shutdown.

4.6.1. Physical layer

A transceiver unit process at the physical layer models the hardware and its device driver. The process implements the interface to the transmission medium, performs collision detection, and models the internal delay in a transceiver unit. Depending on the modeled hardware, additional features such as Cyclic Redundancy Check (CRC) [38] for error detection or an algorithm for encryption are incorporated to the model. No real CRC or encryption calculation is performed, but the data consistency is validated from the parameters of the signal received from the transmission medium. The upper interface of the transceiver unit defines signals for the lowest protocol layer. Dedicated signals are declared for the transceiver unit control, transmitter or receiver enabling, carrier sensing, data sending, send confirmation, and data reception indication.

Transferring data to or from the transceiver unit causes delay, because a transceiver unit is typically a distinct hardware component on a node platform. A delay, which is calculated by dividing the data length in bits by the interface bit rate, is generated when data is loaded to or from transceiver unit.

4.6.2. Sensor interface

The sensor interface block implements a process that models the Analog-to-Digital Converter (ADC) and sensor operations. When an application task initiates sensing, it sends a signal to the sensor interface process. The process activates the sensor and other required peripherals (e.g. ADC for sampling the analog sensor output) for that phenomenon. The sensor interface process acquires a value from the sensing channel by signal exchange. The operation delay depends on the associated sensor and possibly on the ADC sampling frequency, which are defined in the input parameters. The used peripherals are reserved during the operation.

4.6.3. Application layer

The application layer consists of a process that implements the scheduling of application tasks. This approach is selected to facilitate the task scheduling when they are implemented as SDL procedures. When an application is described as a task graph, the application layer process emulates the execution of tasks. In this case no real functionality apart from the sensing and data transfer initiation is implemented.

The application task procedures are implemented by the designer. They define the functionality of the tasks, while the task state control and scheduling are implemented in the application layer process. Task state is *running* when it is executed, *ready* when it is ready for execution but another task is running, or *wait* when the task requires either a timer, data, or sensor event to occur before running. Supported scheduling algorithms are round robin and static priority scheduling.

Please cite this article in press as: M. Kuorilehto et al., Rapid design and evaluation framework for ..., Ad Hoc Netw. (2007), doi:10.1016/j.adhoc.2007.08.003

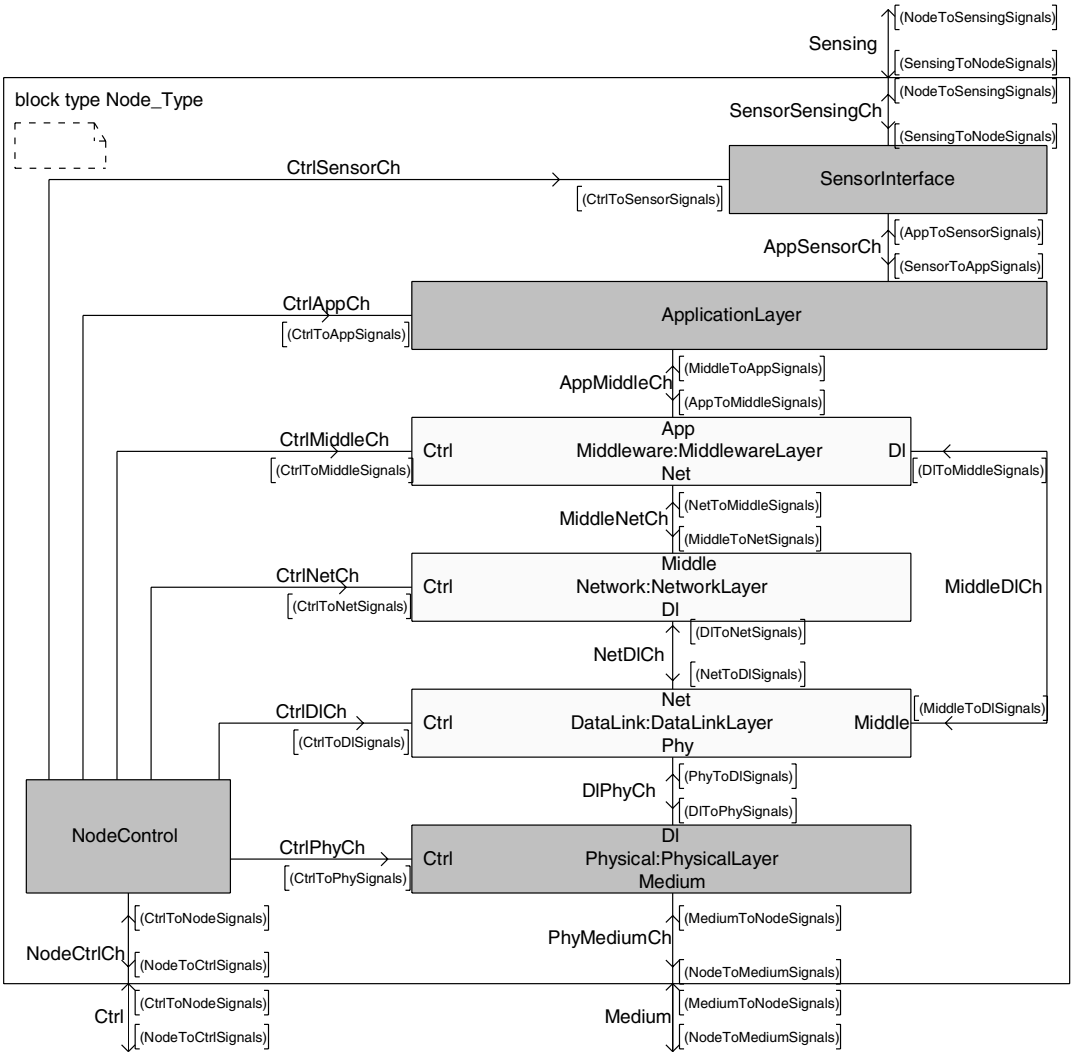


Fig. 6. The SDL block type for WISENES sensor node.

When a task is ready and scheduled for processing, the application layer process calls the procedure that implements the task. The event that moved the task to the ready state and the payload associated to the event are given in the parameters of the procedure. When the task completes its next transition it enters to the wait state. The waited event is returned to the application layer process. In occurrence of an event, all tasks waiting for it are set to the ready state.

4.6.4. Node control

The node control block consists of two processes, *Node control* and *Node simulation control*. The node control process implements OS routines in WISENES. The possible states of a sensor node and the actions triggering node state transitions are depicted in Fig. 7. When the node is in an *active* state, its Central Processing Unit (CPU) and transceiver unit are powered. Transceiver unit dependent states *receiving* and *transmit* are substates of the active state. The node enters to a *transceiver sleep* state, when its transceiver unit is not needed during a constant period. When there is nothing to process on CPU, the node is set to a sleep state that depends on the length of the inactive period. The periods and corresponding sleep states are defined in the input parameters. Tasks can be executed and sensing activated when the node is in the active state, one of its substates, or in the transceiver sleep state.

The remote procedures exported by the node control for implementing OS routines are presented

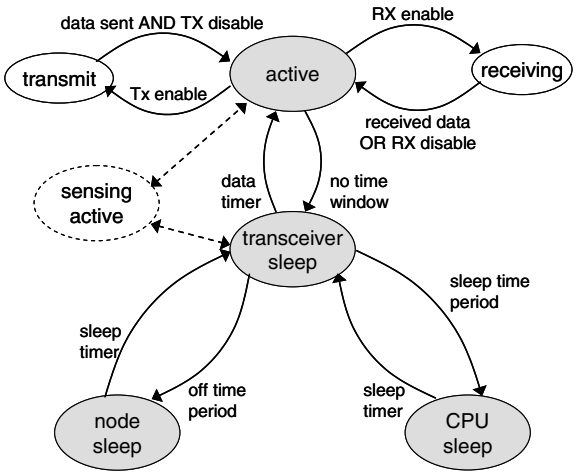


Fig. 7. Sensor node states and state transitions in WISENES.

in Table 5. Periodically activated protocols and application tasks call *SetNextActiveTime* procedure to indicate their next wakeup time. When all protocols and application tasks are inactive, the node control sets the node to a sleep state. The memory management for application tasks and protocols is implemented in *ReserveMemory* and *FreeMemory* procedures. Protocols and applications reserve CPU time slots by calling *Execute* procedure. The executed operations are given as a parameter. A protocol may check remaining energy resources of the node by calling *GetCurrentBatteryLevel* procedure.

Table 5
The remote procedures exported by the node control and node simulation control

Procedure	Parameters
Node control	
SetNextActiveTime	CallerId, ActivationTime, TransceiverUnitControl
ReserveMemory	CallerId, Type, Amount
FreeMemory	CallerId, Type, Amount
Execute	CallerId, OperationCount
GetCurrentBatteryLevel	*Returns* CurrentLevel
Node simulation control	
UpdateNodeRole	NodeRole
UpdateConnectivity	NearbyNodeIds, NearbyNodeSignalStrengths
ConsumePeripheralPower	PeripheralId, ActivationTime
ConsumeTransceiverSendPower	SendBytes
ConsumeTransceiverReceivePower	Type, ReceiverActivationTime
SetNodeState	State
MarkActiveExecution	CallerId, OperationCount
GetRemainingEnergy	*Returns* RemainingEnergy

The *Node simulation control* implements a per-node interface to the central simulation control and models the power consumption of node platform. In the initiation, the node simulation control relays node parameters in signal parameters to the node control and to the different layers. During an active simulation run, the node simulation control gathers GUI related information from the node and passes it to the central simulation control. Remote procedures that implement the gathering are presented in Table 5. *UpdateNodeRole* and *UpdateNodeConnectivity* must be called from protocols that possess the required information.

The node power consumption modeling is implemented in the node simulation control by a linear battery model, in which the component power consumption is independent of the battery discharge rate. The remote procedures related to the power consumption are called only from the SDL processes that are part of the WISENES framework. The sensor interface process indicates a peripheral activation by calling *ConsumePeripheralPower* procedure. Procedures *ConsumeTransceiverSendPower* and *ConsumeTransceiverReceivePower* are called by the transceiver unit process when a transmitter and a receiver are activated, respectively. The node control marks the node state transitions by calling *SetNodeState* and indicates the execution by calling *MarkActiveExecution*. *GetCurrentBatteryLevel* procedure in the node control calls *GetRemainingEnergy* to determine the battery level.

The power consumption in a sensor node can be divided into a very detailed level, as peripherals, protocols, and application tasks can be identified when they indicate their activation by calling a dedicated remote procedure. The power consumption by the transceiver unit can be split between the transmission and reception, while CPU power consumption can be assigned to the different states, and to the application tasks, protocols, and device drivers.

The harvesting of energy from the surroundings is modeled in the node simulation control, if such a peripheral is available at the node platform. The generated energy is randomized between 0 and the harvesting capacity limit specified in the input parameters. When a sensor node runs out of energy, the node simulation control removes the node from the transmission medium and indicates this to the central simulation control.

Please cite this article in press as: M. Kuorilehto et al., Rapid design and evaluation framework for ..., Ad Hoc Netw. (2007), doi:10.1016/j.adhoc.2007.08.003

4.7. Prototype mapping in WISENES

The accurate parameterization of sensor node platforms in the XML configuration files and the detailed capability modeling in the WISENES framework lead to the simulation results that correspond to those obtained from real node platforms. We refer the parameterization of hardware platforms and their modeling in the simulator to as *prototype mapping*.

The exact modeling of node and peripheral state changes, and the detailed specification of the node power characteristics in the input parameters result in realistic mapping of the power consumption. For a fine-grained power consumption mapping of protocol and application execution, the number of executed operations given as a parameter to *Execute* procedure must be estimated during the initial design phase. For the further evaluation, benchmarking information from the prototype measurements is utilized.

The memory and processing capacities of the node platforms are defined in the input parameters. The static instruction and data memory usage of protocols and applications is parameterized, whereas the dynamic data memory consumption depends on the data buffering and control constructs stored within each protocol layer. The processing capacity is controlled by *Execute* procedure in the node control.

The delays in the data transmissions are modeled in the transmission medium and in the transceiver unit. This approach considers the transfer delay and the internal processing delay of the transceiver unit, but omits the signal propagation delay in the transmission medium.

4.8. Simulation of node code implementations in WISENES

In addition to the simulation of node models that are composed of protocols implemented in SDL, WISENES contains a *prototype emulation environment*, which allows the simulation of low-level C implementations that are directly applicable for node prototypes. Similarly to sQualnet [15], the C code implementation is integrated above the data link layer by a dedicated network layer SDL block. This block consists of a process that redirects the per-node signals received from the data link layer to the emulation environment and vice versa, and implements the timer concept for the emulation.

When an SDL block in WISENES is created for a node, the prototype emulation environment creates a simulator host OS thread for the node and associates the thread to the node id. On the reception of a signal from the data link layer, the SDL block calls a signaling function from the prototype emulation environment. The emulation environment redirects the signal to the correct thread and converts the signal parameters to a function call or to a message for the final C implementation. After the subsequent processing is completed, the SDL block calls a query function at the emulation environment. The query function gathers the events caused by the received signal and passes them back to the SDL block. The indications related to timers are handled similarly.

The current version of the prototype emulation environment in WISENES supports TUTWSN C code implementations. The required effort for porting the emulation environment core to support other platforms is minimal. Only the interface provided by the emulation environment for the C code needs to be adapted.

5. WISENES use-cases: TUTWSN and ZigBee

WISENES is used for the design and evaluation of two use-cases: proprietary TUTWSN and a standard ZigBee network. In both, the design starts from scratch and ends up to extensive performance simulations. Prototype platforms for both cases are parameterized using the XML configuration files and the protocols are designed and implemented in SDL.

5.1. TUTWSN implementation

TUTWSN is a very energy efficient WSN targeted to low data rate applications, such as environmental and industrial monitoring [4]. TUTWSN consists of a configurable full feature protocol stack, a family of physical node platforms, and several GUIs for the network management and visualization.

5.1.1. TUTWSN protocol stack

The TUTWSN protocol stack in WISENES consists of a middleware, a multi-hop routing protocol at the network layer, and a MAC protocol at the data link layer. TUTWSN MAC is an energy efficient clustered protocol that minimizes the time spent in a receiving state per a node. Time Division

Please cite this article in press as: M. Kuorilehto et al., Rapid design and evaluation framework for ..., Ad Hoc Netw. (2007), doi:10.1016/j.adhoc.2007.08.003

Multiple Access (TDMA) is used for the intra-cluster communication, whereas neighboring clusters operate on different frequency channels. The control signaling, not the data transfers, for the inter-cluster communication is performed on a dedicated network signaling channel.

At the beginning of each access cycle a cluster headnode broadcasts an active network beacon to the network signaling channel. Other headnodes utilize the beacon for the multi-hop routing, whereas subnodes listen the channel only when they are searching for a cluster for the association. Neighbor discovery times are shortened by idle network beacons that are sent during the inactive period of the access cycle. An idle network beacon indicates the time of the next active network beacon.

The communication within a cluster is performed during a superframe consisting of cluster beacons, aloha slots for contention, and reservation data slots. The inter-cluster data transfers are made during the superframe of the recipient headnode. Cluster beacons start a superframe by informing the associated nodes of the allocated reservation data slots and access cycle timing. Subnodes listen the beacons but sleep the rest of the access cycle, unless they have data to process. In the contention slots, subnodes and neighbor headnodes send occasional data and slot reservation requests to the cluster headnode. The reservation data slots are allocated for periodical data transfers. Each slot consists of an uplink for data sending to the headnode and of a downlink for acknowledgements and data sending to the associated node.

The TUTWSN MAC protocol in WISENES implements a self-organizing cluster creation algorithm, and intra-cluster and inter-cluster communication. For the inter-cluster communication, the cluster access cycle timing can be adapted according to the routing protocol needs. In order to minimize the delay, the start time of the own access cycle is adjusted so that the superframe is completed right before the start of the access cycle of the next hop cluster. The length of a slot is 20 ms, uplink and downlink being both 10 ms. The access cycle length, and the number of contention and reservation data slots are varied.

The flooding routing protocol is implemented for the multi-hop path creation. Route requests are broadcast to the network until a path to the destination is found. In order to avoid unnecessary communication, the route requests are identified so that duplicates can be discarded. Further, the neighboring information from the MAC layer is utilized. If a node knows a valid route to the destination, it initiates a response. Each node stores only the address of the next hop and the total hop count for each destination.

The TUTWSN middleware layer controls the application task hosting in sensor nodes and abstracts the communication between tasks. The middleware keeps track of the neighbor nodes, which host tasks that are related those tasks hosted by the node itself. When a task sends data to another task, the middleware redirects the data to the correct node.

5.1.2. TUTWSN prototype platform

The TUTWSN prototype used in the simulations is depicted in Fig. 8. The main component on the prototype is a 2 MIPS Xemics XE88LC02 [43] MicroController Unit (MCU) consisting of a CoolRisc 816 processor core, a 16-bit ADC, 22 KB program memory, and 1 KB data memory. 2.4 GHz NordicVLSI nRF2401 transceiver unit [44] on the prototype supports 250 kbps and 1 Mbps data rates with transmit power adjustable between −20 and 0 dBm. A 16-bit CRC error detection is implemented in the transceiver unit. For the environmental monitoring, the prototype has an integrated MAX6607 temperature sensor. A 0.22 F capacitor is used as the energy storage for the prototype.

The mapping between the TUTWSN prototype and the WISENES sensor node model is depicted in Fig. 9. The TUTWSN protocol stack for the prototype is implemented in C and executed on Xemics MCU. In WISENES, it is implemented in SDL on the sensor node model. Application tasks run on top of the protocol stack. A lightweight OS that is implemented in C controls the scheduling and power management on the prototype. In WISENES, its functions are modeled by the node control.

The nRF2401 transceiver unit and its device driver are implemented by the transceiver unit process

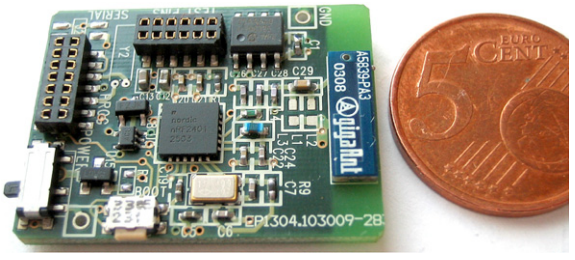


Fig. 8. TUTWSN Xemics prototype.

Please cite this article in press as: M. Kuorilehto et al., Rapid design and evaluation framework for ..., Ad Hoc Netw. (2007), doi:10.1016/j.adhoc.2007.08.003

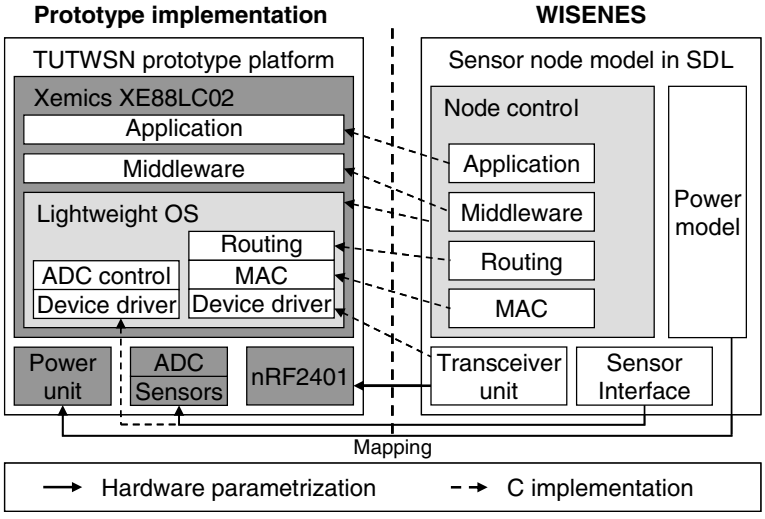


Fig. 9. The prototype mapping between the WISENES sensor node model and TUTWSN prototype.

in WISENES. The sensor interface process models the sensors, ADC, and their device drivers. The power model in the node simulation control simulates the power unit and the power consumption of the hardware components.

Fig. 10 gives an example of the XML configuration parameters that specify the presented TUTWSN prototype for WISENES. A dedicated parameter set for a sensor node is assembled from the node type parameters (a), transceiver unit parameters (b), peripheral parameters (c), and node parameters (d). The transceiver unit dependent signal attenuation curve constants for nRF2401 are following: $k = 0.2385$, $b = 2.7$, $v = 18.0$, and $L = 0.03$. These are obtained by measuring PER for different distances and deriving the values from the results.

5.2. ZigBee implementation

ZigBee [5] is a networking architecture targeted to low-cost and low-power monitoring and control applications. Although not directly designed for WSNs, many of its characteristics have encouraged its use also in WSN scenarios. The topology in a ZigBee network can be either star or mesh. In the star topology, a single *ZigBee coordinator* controls the whole network, whereas in the mesh topology, nodes communicate directly through peer-to-peer links. A special type of a mesh network is a cluster-tree topology, in which a coordinator starts the network but other coordinators (referred also to as *ZigBee routers*) can extend the network. In the following, we concentrate on the cluster-tree topology, because it is the only alternative for the low-power WSNs and the closest to the TUTWSN use-case.

5.2.1. ZigBee protocol stack

The ZigBee protocol stack defined in the specification consists of a MAC protocol, network layer, and application layer. WISENES implements a full featured version of the IEEE 802.15.4 Low-Rate Wireless Personal Area Network (LR-WPAN) MAC protocol [45] for ZigBee, and a simplified version of the ZigBee networking layer. The same middleware layer as in TUTWSN is used on top of the ZigBee protocols.

The ZigBee MAC in the cluster-tree topology networks uses a beacon-enabled channel access. A ZigBee network is started and its parameters defined by a ZigBee coordinator. When joining to a network, a ZigBee router starts transmitting beacons on the same communication channel with the parameters advertised by its parent. The beacon is followed by a Contention Access Period (CAP) for Carrier Sense Multiple Access (CSMA) data transfers. Periodic data transfers between the ZigBee coordinator and a child device can be made in guaranteed time slots during a Contention Free Period (CFP) that follows CAP in the superframe of the ZigBee coordinator. The length of the access cycle (beacon period) and the superframe (CAP period) depends on the constants *BeaconOrder* and *SuperframeOrder* that are defined by ZigBee coordinator.

Please cite this article in press as: M. Kuorilehto et al., Rapid design and evaluation framework for ..., Ad Hoc Netw. (2007), doi:10.1016/j.adhoc.2007.08.003

```
<node_type id="1">
  <name>TUTWSN Xemics Node</name>
  <type>FFU</type>
  <cpu_info>
    <capacity>2000000</capacity>
    <code_memory_inst>8000</code_memory_inst>
    <data_memory_byte>1024</data_memory_byte>
  </cpu_info>
  <battery>
    <voltage_V>3.0</voltage_V>
    <capacity_mAh>0.57</capacity_mAh>
    <efficiency>0.8</efficiency>
    <harvest_uW>0</harvest_uW>
  </battery>
  <state_info>
    <state name="active" ms="0" uW="1350"/>
    <state name="node sleep" ms ="3" uW="19"/>
  </state_info>
  <transceiver_unit id="1"/>
  <peripheral_info>
    <peripheral id="1" count="1"/>
    <peripheral id="2" count="2"/>
  </peripheral_info>
</node_type>
```

a

```
<transceiver_unit id="1">
  <name>nRF2401</name>
  <throughput_bps>1000000</throughput_bps>
  <rssi capability="NO"/>
  <data_load_info bps="230400" uW="1310"/>
  <receiver_info transient_ms="0.25" uW="43600"/>
  <transmitter_info>
    <transient_ms>0.25</transient_ms>
    <tx_power_levels>
      <tx_power dBm="-20" uW="18700"/>
      <tx_power dBm="-10" uW="23500"/>
      <tx_power dBm="-5" uW="26400"/>
      <tx_power dBm="0" uW="29300"/>
    </tx_power_levels>
  </transmitter_info>
  <carrier_sense_info capability="NO"/>
</transceiver_unit>
```

b

```
<peripheral id="1" phenomenon="NONE">
  <name>ADC</name>
  <dependency id="0"/>
  <active_period ms="0.5" uW="15"/>
</peripheral>
<peripheral id="2" phenomenon="TEMPERATURE">
  <name>Temperature sensor</name>
  <dependency id="1"/>
  <active_period ms="0.1" uW="10"/>
</peripheral>
```

c

```
<node_list count="5">
  <node id="1" type="2">
    <coordinates x="0.0" y="0.0" z="0.0"/>
  </node>
  <node id="2" type="1">
    <coordinates x="7.23" y="3.4" z="0.72"/>
  </node>
  <node id="3" type="1">
    <coordinates x="8.4" y="-5.21" z="1.03"/>
  </node>
  <node id="4" type="1">
    <coordinates x="15.4" y="-8.53" z="1.76"/>
  </node>
  <node id="5" type="1">
    <coordinates x="22.4" y="0.21" z="0.93"/>
  </node>
</node_list>
```

d

Fig. 10. TUTWSN prototype: (a) node type, (b) transceiver unit, (c) peripheral and (d) node parameters for WISENES.

Each child device communicates with its parent coordinator during CAP. All transactions must be completed before the end of CAP. Before a transmission, a random back-off period is waited. After the back-off, the state of the transmission medium is assessed. If the medium is busy, the back-off procedure continues. If a coordinator has data pending for its child devices, it indicates the identifiers of these child devices in the beacon. When a device receives a beacon listing its address, it sends a data requests to the coordinator, after which the transmission of data is made.

The ZigBee networking layer in WISENES implements a mechanism for joining and leaving the network, the algorithms for the cluster-tree topology formation, and data routing from devices to the ZigBee coordinator. The routes are created according to parent associations, i.e. each ZigBee router sends data targeted to the ZigBee coordinator to its parent. The continuous neighbor discovery and the support for other network topologies are not implemented. Further, the short address assignment and superframe scheduling algorithms benefit the simulator addressing and timing information.

5.2.2. ZigBee prototype platform

The prototype platform used in WISENES for the ZigBee evaluation is constructed from two separate components. The MCU module of the prototype is 2 MIPS PIC18LF4620 MCU [46] with 64 KB of program and 4 KB of data memory, an integrated 10-bit ADC, and an interface to MAX6607 temperature

Please cite this article in press as: M. Kuorilehto et al., Rapid design and evaluation framework for ..., Ad Hoc Netw. (2007), doi:10.1016/j.adhoc.2007.08.003

```
<node_type id="1">
  <name>ZigBee PIC node</name>
  <type>FFU</type>
  <cpu_info>
    <capacity>1000000</capacity>
    <code_memory_inst>32768</code_memory_inst>
    <data_memory_byte>3986</data_memory_byte>
  </cpu_info>
  <battery>
    <voltage_V>3.3</voltage_V>
    <capacity_mAh>2100</capacity_mAh>
    <efficiency>0.91</efficiency>
    <harvest_uW>0</harvest_uW>
  </battery>
  <state_info>
    <state name="active" ms="0" uW="3010"/>
    <state name="node sleep" ms ="2" uW="30"/>
  </state_info>
  <transceiver_unit id="2"/>
  <peripheral_info>
    <peripheral id="1" count="1"/>
    <peripheral id="2" count="2"/>
  </peripheral_info>
</node_type>
```

a

```
<transceiver_unit id="2">
  <name>CC2420</name>
  <throughput_bps>250000</throughput_bps>
  <rssi capability="YES"/>
  <data_load_info bps="460800" uW="1310"/>
  <receiver_info transient_ms="0.22" uW="54820"/>
  <transmitter_info>
    <transient_ms>0.22</transient_ms>
    <tx_power_levels>
      <tx_power dBm="-15" uW="28300"/>
      <tx_power dBm="-10" uW="31400"/>
      <tx_power dBm="-5" uW="37600"/>
      <tx_power dBm="0" uW="46400"/>
    </tx_power_levels>
  </transmitter_info>
  <carrier_sense_info capability="YES" uW="54200"/>
</transceiver_unit>
```

b

Fig. 11. ZigBee prototype: (a) node type and (b) transceiver unit parameters for WISENES.

sensor. The transceiver unit is 2.4 GHz Chipcon CC2420 transceiver [47], which is IEEE 802.15.4 standard compliant. The transceiver support 250 kbps data rate and the transmit power is adjustable between −24 dBm and 0 dBm. A CR123A lithium battery is used as an energy storage.

Although the prototype is assembled with two distinct components, the resulting platform is realistic. MCU has enough resources for the ZigBee implementation. The power consumptions of the components are measured separately, but they are combined in WISENES. The transceiver unit parameters are measured using a Chipcon SmartRF CC2420DK Development Kit [47]. The constants for the transmission medium are: $k = 0.08$, $b = 3.4$, $v = 6.0$, and $L = 0.05$. The WISENES XML configuration parameters describing the prototype node type (a) and transceiver unit (b) are depicted in Fig. 11.

6. WISENES evaluation

The design of a new protocol and the deployment of a complete WSN protocol stack in WISENES are straightforward for the designer due to the hierarchical structure of SDL and the modularity of the WISENES framework. The graphical design based on state machines is well-suited for the protocol modeling. The modeled protocols are internally independent of WISENES, but an external interface for the adaptation of each layer to the WISENES framework is required. Interface templates are provided for the designer.

For the usage evaluation, the full feature ZigBee MAC protocol was implemented according to the specification by a single designer within two working weeks. The designer was familiar with the WISENES interfaces, but still the development cycle was faster than expected. The SDL description attached to the IEEE 802.15.4 standard was too incomplete to be used in WISENES.

The full implementation of the ZigBee MAC protocol in WISENES consists of three processes having totally 31 different states. In addition, 75 SDL procedures are implemented in order to avoid redundant implementations and clarify the description. The number of state transitions in the implementation is 163. Among the transitions, there are in total 1817 divergent execution paths. The implementation of TUTWSN MAC protocol in WISENES consists of two processes that have totally 21 states and 32 procedures. The number of state transitions is 112, but there are still 2642 divergent execution paths.

Please cite this article in press as: M. Kuorilehto et al., Rapid design and evaluation framework for ..., Ad Hoc Netw. (2007), doi:10.1016/j.adhoc.2007.08.003

The simulation and evaluation capabilities of WISENES are assessed by simulating TUTWSN and ZigBee networks. The application used in the simulations is an environmental monitoring application. All nodes in WSN observe the temperature in their vicinity. Sensed data are aggregated in cluster headnodes (coordinator) and routed for a further processing to the sink node (ZigBee coordinator). The application is implemented by describing it as a task graph.

The evaluated aspects in WISENES are the performance of the simulator, applicability for the large scale simulations, and the accuracy of the prototype mapping. All simulations are executed on a workstation with a 2.8 GHz Pentium4 processor with 1 GB of memory and running Windows XP SP2. WISENES memory consumption is limited to 150 MB, meaning that gathered log data are written to files when the limit is exceeded.

6.1. WISENES performance

TUTWSN simulations are repeated five times with 10, 100, and 1000 of nodes. The node population is randomly generated for each simulation run. Monitored aspects are the correct functionality and performance. The initial energy capacity of a TUTWSN node is set twenty times larger than specified in Fig. 10 in order to obtain required lifetime. The TUTWSN access cycle in simulations is 10 s, and each node measures temperature once in an access cycle and sends the result towards the sink node. Headnodes aggregate the subnode readings to a single data packet.

The presented WISENES performance is the time elapsed for the simulation of a single node for a 24-h period. During that time, a node initiates 8640 data packets. The resulting time is obtained by dividing the overall network simulation time by the node count. The elapsed per-node simulation times with and without GUI are depicted in Fig. 12. In addition, the time consumed on writing the log data to files is presented separately.

The operations on a single node remain similar regardless of the node count. Hence, the simulation time per a node should be at the same level in all cases. However, as depicted, the time per node increases as the node count increases. This is caused by the increased time consumption in the processing of longer lists in both Telelogic TAU SDL Suite simulation engine and WISENES framework components. The communication with GUI is time consuming but the penalty is independent of the node count. The increase in the simulation time is caused by the environment function polling and the large amount of node information passed through the socket interface.

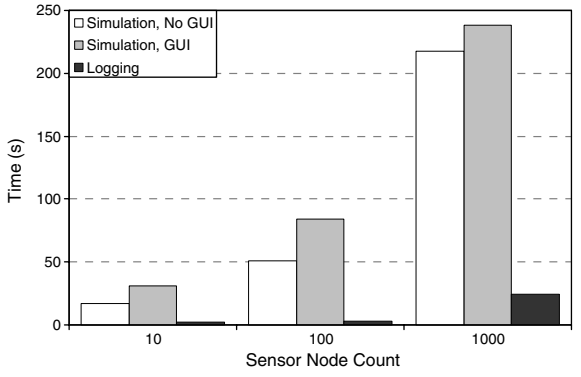


Fig. 12. Time required for the simulation of a single node in WISENES for a 24-h period with varying total number of nodes.

As depicted in Fig. 12, the time consumed on log writing with 1000 node simulations is ten times longer than in the other cases. A reason for this is not accurately known, but it may be due to the fragmentation of the storage disk, because the size of the logged data in this case is over 12 GB.

6.2. Prototype mapping results

For the evaluation of the prototype mapping accuracy, a similar configuration, illustrated in Fig. 13, is constructed for both WISENES and prototypes. Subnodes S1 and S2 perform sensing and send the data to headnode H1 once in every access cycle. Headnode H1 aggregates data and sends them to the sink node through headnode H2. The number of contention slots is four, reservation data slots are limited to eight, and idle network beacons are sent every 250 ms. The access cycle length is 1, 2, 5, and 10 s. Cluster scanning is avoided by using static access cycle timings.

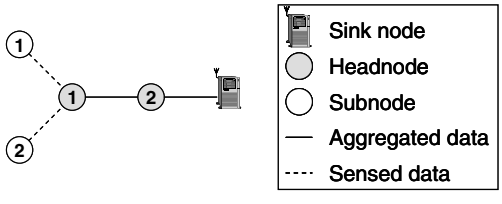


Fig. 13. Static topology for the prototype mapping test case.

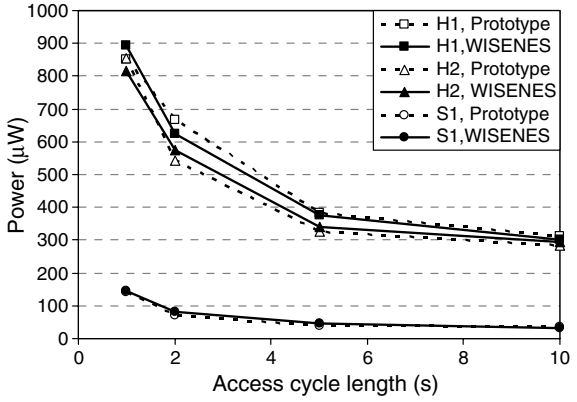


Fig. 14. Modeled WISENES and measured TUTWSN prototype power consumptions in the prototype mapping test case.

Fig. 14 presents the simulated power consumption from WISENES in contrasts to the measured prototype power consumption. Generally, the power consumption in subnodes is minimal compared to that of headnodes. The margin between the two headnodes is not considerable, because the only difference in their activity is one active reservation data slot. When compared to the prototype measurements, the WISENES results are very accurate. The overall average difference is 6.73%. The results are more accurate for headnodes, average difference being 4.0% for H1 and 4.8% for H2. Due to the very low activity, the modeling in subnodes is more inaccurate, as the average difference in case of S1 is 11.34%. The main reason to this is the differences in the timing models of WISENES and the prototypes.

Other aspects related to prototype mapping are delay and throughput. As mentioned, WISENES models delays in the transmission medium and transceiver units accurately. Only the delay of signal propagation is omitted, but it is negligible in short distances when compared to the other delays in transmissions. Moreover, the main causes for the delays on WSNs are higher layer protocol buffering and channel access. Thus, the verification of the delay mapping is omitted.

WSNs do not utilize the full bandwidth available on their transceiver units but only a small fraction of it. The throughput depends on the channel access method and PER. In WISENES, PER is derived from the transceiver unit dependent measurements and the implemented channel access methods are identical. Hence, the further verification of the throughput mapping is also omitted.

6.3. TUTWSN simulation results

A network of thousand nodes is simulated in order to evaluate the applicability of TUTWSN in large scale. The number of contention slots is set to four, reservation data slots to eight, and idle network beacons are sent every 250 ms. The access cycle length is 1, 2, 5, and 10 s. The environmental monitoring application is again activated once in an access cycle.

6.3.1. Power consumption

The average power consumptions for five arbitrary selected headnodes and subnodes are depicted in Fig. 15a and b respectively. The scale in Fig. 15a is approximately eight times larger than in Fig. 15b. The power consumptions of the transceiver unit, peripherals, power unit, and MCU in sleep and active states are presented separately.

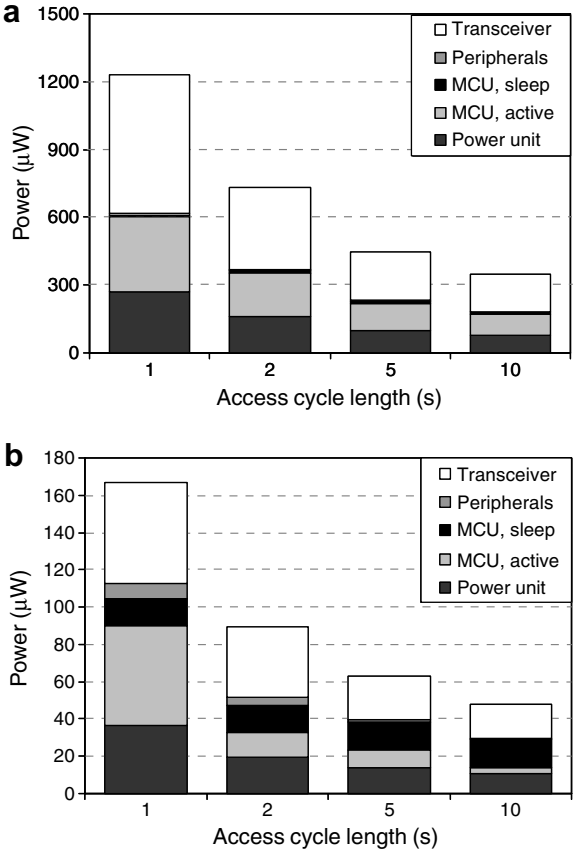


Fig. 15. The power consumptions of different components in TUTWSN: (a) headnode and (b) subnode in a thousand-node network.

Please cite this article in press as: M. Kuorilehto et al., Rapid design and evaluation framework for ..., Ad Hoc Netw. (2007), doi:10.1016/j.adhoc.2007.08.003

As shown, the transceiver unit is the dominating power consumer in both headnodes and subnodes. With shorter access cycles, the share of the transceiver unit in the power consumption is more dominant. The headnodes spend considerably more time with CPU active, due to added processing and active waiting while receiving data. The subnodes spend most of the time in the sleep states.

Compared to the power consumptions in the prototype mapping case, the presented results are averagely 28.7% larger for subnodes and 22.4% larger for headnodes. This is mainly due to the scanning required for the network topology creation and maintenance, which were omitted in the prototype mapping case. Further, headnodes have more active reservation data slots.

6.3.2. TUTWSN lifetime

The lifetimes of TUTWSNs with different access cycle lengths are presented in Fig. 16. The lifetimes are shown for both a case where a node acts as a headnode until it runs out of energy, and a case where a headnode deliberately releases its duty when its remaining energy level is first 50% and then 10%. The network lifetime is considered as the time until 50%r 20% of the nodes are left. In the first case, measurement data with reasonable accuracy can be obtained from WSN, while in the latter case the accuracy suffers but the network is still capable of providing routes to sink node.

The changing of the cluster headnode balances the load between the nodes in the network. This lengthens the time until the first node runs out of energy. Yet, the time between the first and the last node running out of energy is minimal. As shown in Fig. 16, the lifetime of the network until half of the nodes are remaining is considerably longer when the headnode duty is circulated. Instead, with lifetime consideration of 20% of nodes remaining, the network longevity is better if the headnode is not changed.

If a node operates as a subnode continuously, lifetimes are 5.2, 9.9, 21.2, and 34.2 h for 1, 2, 5, and 10 s access cycles respectively. The reason for the short lifetimes is the extremely limited capacity of the capacitor being the energy storage at the nodes. For comparison, with a 1 cm^3 non-rechargeable lithium battery, the obtained lifetimes for a subnode are 96, 182, 391, and 631 days for 1, 2, 5, and 10 s access cycles respectively [48].

6.3.3. Delay and throughput

Fig. 17a depicts the communication delays for different number of hops from a source to the sink node. The delays are measured after the cluster access cycle timings have been adapted and stabilized. For one and two second access cycles, the delays are acceptable. For an environmental moni-

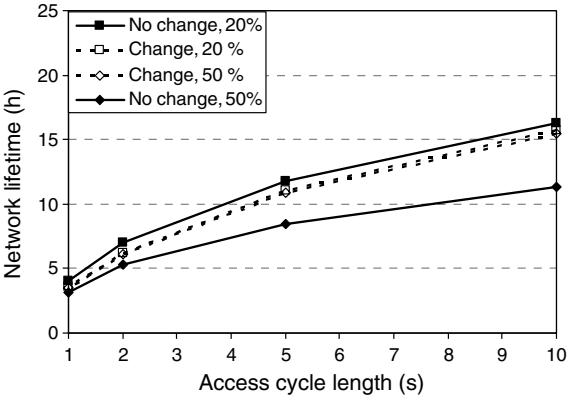


Fig. 16. TUTWSN lifetimes in different conditions.

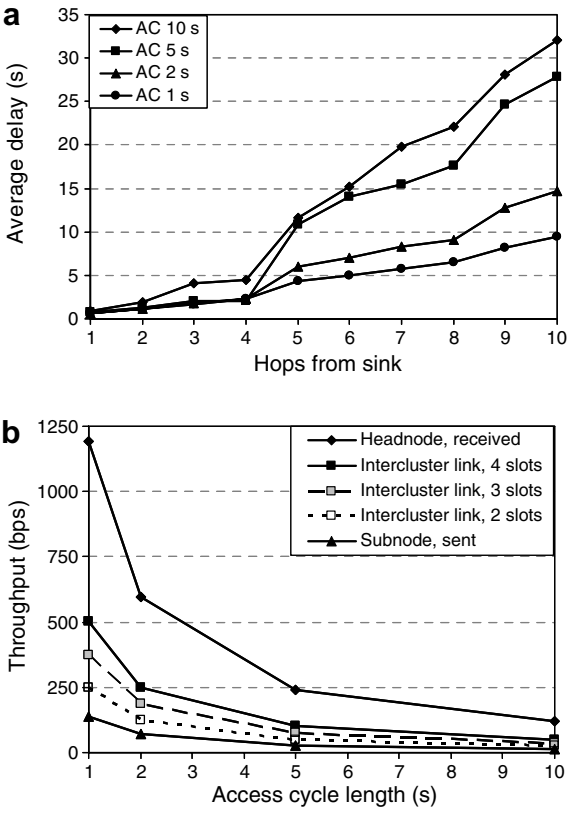


Fig. 17. TUTWSN: (a) packet delays to sink and (b) throughputs.

toring application, the delays are adequate even with the longer access cycle lengths. A per hop delay is quite independent of the distance from the sink node but the nodes in which several data flows converge cause congestion.

Throughputs for a subnode and a headnode, as well as for the active inter-cluster links with different number of reservation data slots are depicted in Fig. 17b. A subnode has one active uplink for the headnode. The headnode throughput is its incoming throughput. The number of the reservation data slots for an inter-cluster link is varied between two and four, depending on the available slots. Obviously, the throughput decreases as the access cycle length increases. This is acceptable, as the access cycle length is typically adapted according to the application requirements.

6.3.4. TUTWSN adaptability

The adaptability of TUTWSN is evaluated by simulating unexpected error situations. An unrecoverable error is simulated at the cluster headnode. The times elapsed until the network is reconfigured are depicted in Fig. 18 for the different access cycle lengths. The subnodes do not start the self-organizing cluster creation algorithm immediately, as a cluster beacon may be lost due to a packet error. Thus, the reconfiguration time depends on the limit of the missed cluster beacon back-off counter. A case, in which the cluster headnode is able to inform about its state, is given as a reference.

As can be seen, the reconfiguration time is almost directly proportional to the access cycle length. The reconfiguration delay is not considerable with the short access cycles, whereas the delay is over half a minute for the longer access cycles.

6.4. ZigBee simulation results

The applicability of ZigBee to WSN applications is evaluated by simulating the ZigBee protocols with the defined environmental monitoring application. In the simulations, the beacon order of ZigBee MAC protocol is varied from 6 to 10, which results to the access cycles of 0.98, 1.97, 3.93, 7.86, and 15.73 s. We refer these to as 1, 2, 4, 8, 16 s access cycles for clarity. The superframe order is set to 2, thus the length of the active period consisting of the beacon and CAP is 61.44 ms.

6.4.1. Small scale ZigBee network

The prototype for ZigBee is modeled from the components, the characteristics of which are measured individually. Therefore, we do not compare the simulated results to physical deployment measurements. For a fair comparison, we use the same statically defined network configuration for ZigBee, which is presented for TUTWSN in Section 6.2. The application in the ZigBee simulation is identical to that of TUTWSN simulations.

The power consumption results from the ZigBee simulations are depicted in Fig. 19. The difference between ZigBee device and coordinator power consumptions is considerably bigger than the same difference between the corresponding TUTWSN nodes. This is caused by the active listening of the complete CAP, which is also the reason for the identical results of both coordinators. Compared to TUTWSN, the power consumption of a device is averagely two times and that of a coordinator averagely 3.5 times larger than the power consumption of the corresponding TUTWSN node.

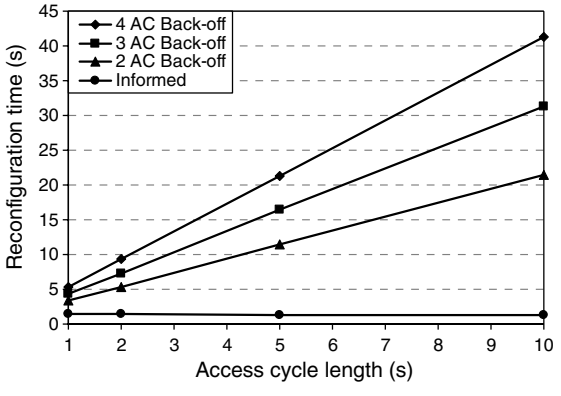


Fig. 18. The reconfiguration times of TUTWSN on an unexpected error situation.

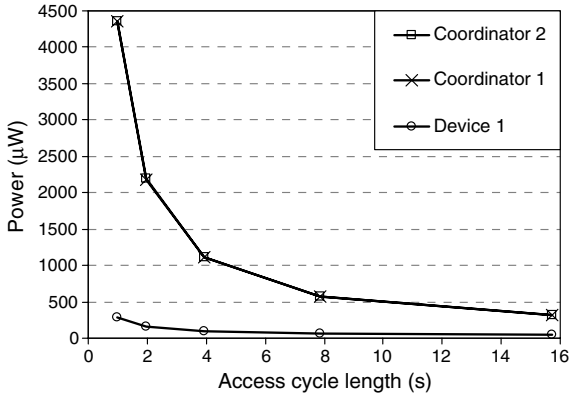


Fig. 19. The power consumption in a small scale ZigBee network.

Please cite this article in press as: M. Kuorilehto et al., Rapid design and evaluation framework for ..., Ad Hoc Netw. (2007), doi:10.1016/j.adhoc.2007.08.003

6.4.2. Large scale network

In the large scale ZigBee simulations, a network that consists of 100 nodes is simulated. The size of the simulated network is restricted by the usage of the same communication channel throughout the network. In dense networks, the distribution of coordinator superframes so that they do not overlap is problematic. In the simulations, 35 coordinator capable devices and 65 reduced function devices are distributed randomly to a 60×60 m area. Due to the congestion, the parameters of the environmental monitoring application are changed so that each coordinator stores the temperatures received from the devices over two access cycles. The results are then aggregated to a single data packet, which is routed towards the ZigBee coordinator.

The power consumptions of the different components in coordinators and devices are depicted in Fig. 20a and b respectively. The access cycle length is varied between 1 and 16 s and the results are averaged over five arbitrarily selected coordinators and devices. The scale in Fig. 20a is 18 times larger than in Fig. 20b. Compared to results presented in Fig. 19, the power consumption of a coordinator is quite similar, whereas in the case devices the power consumption depicted in Fig. 20b is considerable larger for longer access cycles. This is caused by the network scanning, which is required for topology creation and reformation in case of errors. A node scans for a complete access cycle when searching for a network.

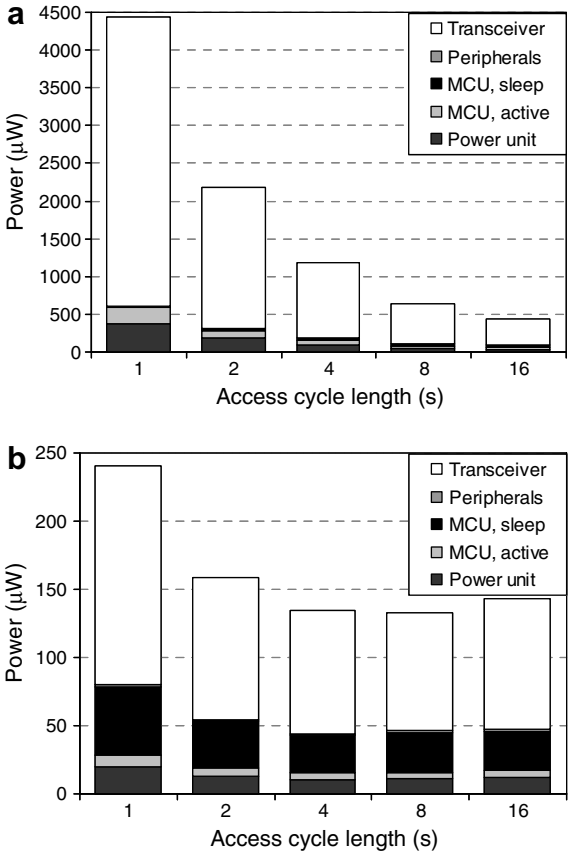


Fig. 20. The power consumptions of different components in ZigBee: (a) coordinator and (b) device in a hundred-node network.

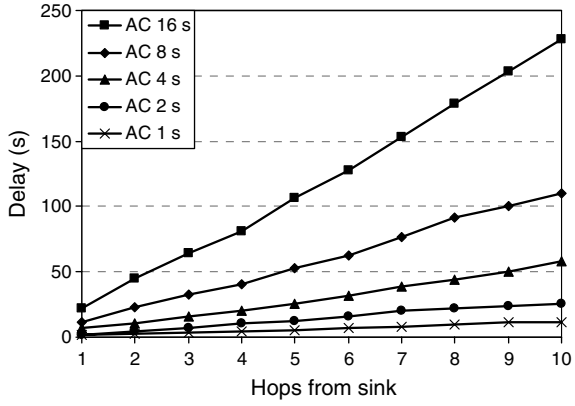


Fig. 21. ZigBee packet delays to the ZigBee coordinator.

In comparison to TUTWSN, the differences are similar to those presented in Section 6.4.1, except for the device vs. subnode power consumption with longer access cycles. While in TUTWSN the scanning times and the energy required for network maintenance diminishes as the access cycle length increases, the effect is opposite in ZigBee. Further, the proportional share of the transceiver unit on the power consumption is significantly larger in ZigBee than in TUTWSN.

The delays in data routing towards the ZigBee coordinator with the different access cycle lengths are depicted in Fig. 21. As in the case of TUTWSN, with 1 and 2 s access cycles the delays are moderate, but with the longer access cycles the delay becomes unacceptable. TUTWSN outperforms ZigBee in this case. The reason for this is that in ZigBee the start times of the access cycles are not delay optimized as in TUTWSN.

7. Conclusions

The large design space of WSNs cannot be managed without a complete tool for the WSN design, configuration, and evaluation. This paper presents

Please cite this article in press as: M. Kuorilehto et al., Rapid design and evaluation framework for ..., Ad Hoc Netw. (2007), doi:10.1016/j.adhoc.2007.08.003

WISENES, which is the first tool that supports the graphical design, simulation, final implementation, and evaluation of WSNs within a single framework. The WISENES framework enables a modular design of WSN protocols and applications. Different platforms and protocols are evaluated in order to obtain an optimal configuration for a specific application. The back-annotation of the measured performance information from physical node platforms improves the accuracy of the simulation results.

The implementation of TUTWSN and ZigBee networks shows that the design of protocols and their performance evaluation in WISENES is fast. The graphical state machine based notation is explicit and designer friendly. The TUTWSN and ZigBee network simulations prove the applicability and performance of WISENES for the simulations of large networks. Further, the simulated performance results correspond to those of real physical platforms.

Our main future work is projected on the development of a more accurate sensing channel model and mobility support. At the moment, the WISENES framework is not publicly available, but we are planning to open it as an online web-based WSN design service. We are also considering open source SDL tools for the design.

References

[1] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci, A survey on sensor networks, IEEE Communications Magazine 40 (8) (2002) 102–114.
[2] J.A. Stankovic, T.F. Abdelzaher, L. Chengyang, S. Lui, J.C. Hou, Real-time communication and coordination in embedded sensor networks, Proceedings of the IEEE 91 (2003) 1002–1022.
[3] SDL Forum Society Homepage, http://www.sdl-forum.org/, visited in November 2005.
[4] M. Kohvakka, M. Hännikäinen, T.D. Hämäläinen, Ultra low energy wireless temperature sensor network implementation, in Proc. 16th annual IEEE Int. Symp. on Personal Indoor and Mobile Radio Communications, 2005, pp. 187–200.
[5] ZigBee Specification, http://www.zigbee.org/, visited in October 2005.
[6] The Network Simulator – ns-2, http://www.isi.edu/nsnam/ns/, visited in November 2005.
[7] X. Zeng, R. Bagrodia, M. Gerla, GloMoSim: a library for parallel simulation of large-scale wireless networks, in: Proc. 12th Workshop on Parallel and Distributed Simulations, 1998, pp. 154–161.
[8] Qualnet Network Simulator, http://www.qualnet.com/, visited in November 2005.
[9] OPNET Modeler,http://www.opnet.com/products/modeler/home.html, visited in November 2005.
[10] OMNeT++ Community Site, http://www.omnetpp.org/index.php, visited in November 2005.
[11] Scalable Simulation Framework, http://www.ssfnet.org/, visited in November 2005.
[12] J-Sim Homepage, http://www.j-sim.org/, visited in November 2005.
[13] S. Park, A. Savvides, M.B. Srivastava, Simulating networks of wireless sensors, Proc. Winter Simulation Conference 2001 (2001) 1330–1338.
[14] I. Downard, Simulating sensor networks in ns-2, http://pf.itd.nrl.navy.mil/nrlsensorsim/, visited in November 2005.
[15] sQualnet: A Scalable Simulation Framework for Sensor Networks website, http://nesl.ee.ucla.edu/projects/squalnet/, visited in November 2005.
[16] J. Liu, L.F. Perrone, D.M. Nicol, M. Liljenstam, Simulation modeling of large-scale ad-hoc sensor networks, in: Proc. 2001 Simulation Interoperability Workshop, (2001).
[17] C. Mallanda, A. Suri, V. Kunchakarra, S.S. Iyengar, R. Kannan, A. Durresi, S. Sastry, Simulating wireless sensor networks with omnet++, http://csc.lsu.edu/sensor_web/simulator.html, visited in November 2005.
[18] S. Dulman, P. Havinga, A simulation template for wireless sensor networks, IEEE Int. Symp. on Autonomous Decentralized Systems, 2003, fast abstract.
[19] A. Sobeih, W.-P. Chen, J.C. Hou, L.-C. Kung, N. Li, H. Lim, H.-Y. Tyan, H. Zhang, J-Sim: A Simulation and Emulation Environment for Wireless Sensor Networks, http://www.j-sim.org/, visited in November 2005.
[20] P. Baldwin, S. Kohli, E.A. Lee, X. Liu, Y. Zhao Modeling of sensor nets in ptolemy II, in: Proc. 3rd Int. Symp. on Information Processing in Sensor Networks, 2004, pp. 359–368.
[21] Ptolemy II, http://ptolemy.eecs.berkeley.edu/ptolemyII/, visited in November 2005.
[22] G. Simon, P. Völgyesi, M. Maróti, Á. Lédeczi, Simulation-based optimization of communication protocols for large-scale wireless sensor networks, Proc. IEEE 2003 Aerospace Conference 3 (2003) 1339–1346.
[23] B.C. Mochocki, G.R. Madey, H-MAS: a heterogeneous, mobile, ad-hoc sensor network simulation environment, in: Proc. 7th Annual Swarm Users/Researchers Conference, 2003.
[24] G. Chen, J. Branch, M.J. Pflug, L. Zhu, B. Szymanski, SENSE: A sensor network simulator, http://www.cs.rpi.edu/~cheng3/sense/, visited in October 2005.
[25] A. Boulis, C.C. Han, M.B. Srivastava, Design and implementation of a framework for efficient and programmable sensor networks, in: Proc. 1st Int. Conf. on Mobile Systems, Applications, and Services, 2003, pp. 801–805.
[26] Crossbow Technology, http://www.xbow.com/, visited in November 2005.
[27] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, K. Pister, System architecture directions for networked sensors, Proc. 9th ACM Int. Conf. on Architectural Support for Programming Languages and Operating Systems (2000) 94–103.
[28] P. Levis, N. Lee, M. Welsh, D. Culler, TOSSIM: accurate and scalable simulation of entire TinyOS applications, Proc. 1st ACM Conf. on Embedded Networked Sensor Systems (2003) 126–137.
[29] M. Karir, J. Polley, D. Blazakis, J. McGee, D. Rusk, J.S. Baras, ATEMU: a fine-grained sensor network simulator, in

Please cite this article in press as: M. Kuorilehto et al., Rapid design and evaluation framework for ..., Ad Hoc Netw. (2007), doi:10.1016/j.adhoc.2007.08.003

Proc. 1st IEEE Int. Conf. on Sensor and Ad Hoc Communication Networks, 2004.
[30] L.F. Perrone, D.M. Nicol, A scalable simulator for TinyOS applications, Proc. Winter Simulation Conference 2002 (2002) 679–687.
[31] S. Sundresh, K. Wooyoung, A. Gul, SENS: a sensor, environment and network simulator, in: Proc. 37th Annual Simulation Symposium, 2004, pp. 221–228.
[32] L. Girod, J. Elson, A. Cerpa, T. Stathopoulos, N. Ramanathan, D. Estrin, Em*: a software environment for developing and deploying wireless sensor networks, CENS Technical Report 0034, 2003. http://research.cens.ucla.edu/, visited in November 2005.
[33] C. Kelly IV, V. Ekanayake, R. Manohar, SNAP: a sensor network asynchronous processor, Proc. 9th Int. Symp. on Asynchronous Circuits and Systems (2003) 24–35.
[34] M. Kuorilehto, M. Kohvakka, M. Hännikäinen, T.D. Hämäläinen, High level design and implementation framework for wireless sensor networks, in: Proc. Embedded Computer Systems: Architectures, Modeling, and Simulation, 2005, pp. 384–393.
[35] Evaluation of J-Sim, http://www.j-sim.org/comparison.html, visited in November 2005.
[36] VisualSense Homepage, http://ptolemy.eecs.berkeley.edu/visualsense/, visited in November 2005.
[37] Getting Started with the EmStar Simulator, http://cvs.cens.ucla.edu/emstar/tut/emsim.html, visited in November 2005.
[38] W. Stallings, Data and Computer Communications, Sixth ed., Prentice-Hall, 2001.
[39] The swing tutorial, http://java.sun.com/docs/books/tutorial/uiswing/, visited in August 2005.
[40] Telelogic Homepage, http://www.telelogic.com/corp/, visited in September 2005.
[41] Telelogic TAU SDL Suite, http://www.telelogic.com/corp/products/tau/sdl/index.cfm, visited in September 2005.
[42] Specification and description language (SDL), ITU-T Recommendation Z.100, http://www.itu.int/ITU-T/studygroups/com17/languages/, visited in June 2006.
[43] XE88LC02 Sensing Machine, Data Sheet, http://www.xemics.com/, visited in September. 2005.
[44] Nordic VLSI nRF2401, Data Sheet, ver 1.0, http://www.nvlsi.no/, visited in September 2004.
[45] Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs), IEEE Standard 802.15.4, 2003.
[46] PIC18 Data Sheet, http://www.microchip.com/, visited in October 2005.
[47] Chipcon CC2420 Data Sheet, http://www.chipcon.com/, visited in October 2005.
[48] S. Roundy, P.K. Wright, J. Rabaey, A study of low level vibrations as a power source for wireless sensor nodes, Elsevier Computer Communications 26 (11) (2003) 1131–1144.

Mauri Kuorilehto (M.Sc.'01, TUT) received the M.Sc. in Computer Science from Tampere University of Technology (TUT), Finland. He is currently pursuing his Ph.D. in the Institute of Digital and Computer Systems at TUT. His research interests include network simulation, operating systems, and application distribution in wireless sensor and ad hoc networks.

Marko Hännikäinen (M.Sc.'98, Ph.D.'02, TUT) acted as a research scientist and a project supervisor at the Institute of Digital and Computer Systems at TUT in 1998–2007, and was nominated to professor in 2007. He co-directs the DACI research group, concentrating on wireless sensor networks, high abstraction design tools, and novel web applications.

Timo D. Hämäläinen (M.Sc. '93, Ph.D.'97, TUT) acted as a senior research scientist and project manager at TUT in 1997–2001. He was nominated to full professor at TUT/Institute of Digital and Computer Systems in 2001. He heads the DACI research group that focuses on three main lines: wireless local area networking and wireless sensor networks, high-performance DSP/HW based video encoding, and interconnection networks with design flow tools for heterogeneous SoC platforms.

Please cite this article in press as: M. Kuorilehto et al., Rapid design and evaluation framework for ..., Ad Hoc Netw. (2007), doi:10.1016/j.adhoc.2007.08.003

PUBLICATION 2

M. Kuorilehto, M. Hännikäinen, T. D. Hämäläinen, "A Survey of Application Distribution in Wireless Sensor Networks," *EURASIP Journal on Wireless Communications and Networking, Special Issue on Ad Hoc Networks: Cross-Layer Issues*, vol. 2005, no. 5, pp. 774–788, December, 2005.

EURASIP Journal on Wireless Communications and Networking 2005:5, 774–788

A Survey of Application Distribution in Wireless Sensor Networks

Mauri Kuorilehto
Institute of Digital and Computer Systems, Tampere University of Technology, P.O. Box 553, 33101 Tampere, Finland
Email: mauri.kuorilehto@tut.fi

Marko Hännikäinen
Institute of Digital and Computer Systems, Tampere University of Technology, P.O. Box 553, 33101 Tampere, Finland
Email: marko.hannikainen@tut.fi

Timo D. Hämäläinen
Institute of Digital and Computer Systems, Tampere University of Technology, P.O. Box 553, 33101 Tampere, Finland
Email: timo.d.hamalainen@tut.fi

Received 14 June 2004; Revised 23 March 2005

Wireless sensor networks (WSNs) are deployed to an area of interest to sense phenomena, process sensed data, and take actions accordingly. Due to the limited WSN node resources, distributed processing is required for completing application tasks. Proposals implementing distribution services for WSNs are evolving on different levels of generality. In this paper, these solutions are reviewed in order to determine the current status. According to the review, existing distribution technologies for computer networks are not applicable for WSNs. Operating systems (OSs) and middleware architectures for WSNs implement separate services for distribution within the existing constraints but an approach providing a complete distributed environment for applications is absent. In order to implement an efficient and adaptive environment, a middleware should be tightly integrated in the underlying OS. We recommend a framework in which a middleware distributes the application processing to a WSN so that the application lifetime is maximized. OS implements services for application tasks and information gathering as well as control interfaces for the middleware.

Keywords and phrases: ad hoc networking, distribution, service discovery, task allocation, wireless sensor networks.

1. INTRODUCTION

Wireless sensor networks (WSNs) have gained much attention in both public and research communities because they are expected to bring the interaction between humans, environment, and machines to a new paradigm. Despite being a fascinating topic with a number of visions of a more intelligent world, there still exists a huge gap in the realizations of WSNs. In this paper, we define WSNs as networks consisting of independent, collaborating nodes that can sense, process, and exchange data as well as act upon the data content. Compared to traditional communication networks, there is no preexisting physical infrastructure that restricts topology.

WSNs are typically ad hoc networks [1] but there are major conceptual differences. First, WSNs are data-centric with an objective to deliver time sensitive data to different destinations. Second, a deployed WSN is application-oriented and performs a specific task. Third, messages should not be sent to individual nodes but to geographical locations or regions defined by data content [2].

In WSNs quantitative requirements in terms of latency and accuracy are strict due to the tight relation to the environment. In general, the capabilities of an individual sensor node are limited, but the feasibility of WSN lies on the joint effort of the nodes. Thus, WSNs are distributed systems and need distribution algorithms. Another motivation for distribution is the resource sharing. Further, to obtain results, WSN applications typically require collaborative processing of the nodes sensing different phenomena in diverse areas [2].

The main focus of WSN research, as well as wireless ad-hoc network research in general, has been on different protocol layers, reviewed in [2, 3, 4, 5, 6, 7, 8] and on energy efficiency [9, 10]. Recently, issues concerning security,

context-sensitivity, and self-organization have gained more attention [11]. Surveys concerning application layer issues and prototype implementations are fairly limited [4, 12, 13]. Furthermore, proposals implementing distribution are emerging as the complexity of applications increases. These are covered in [2] but the discussion of proposals supporting application distribution is limited to few solutions for distribution control.

In this paper, we focus on four essential distribution aspects in WSNs, namely, *service discovery*, *task allocation*, *remote task communication*, and *task migration*. The service discovery comprises of identifying and locating services and resources required by a client. In homogeneous WSNs, the service discovery is not important but when node platforms and the composition of tasks are heterogeneous, the service discovery is essential. The task allocation specifies a set of sensor nodes, on which the execution of an application task is activated. The remote task communication covers the means for communication between distributed tasks through a wireless communication link. The task migration means the methods for transferring a task executable from a sensor node to another. The algorithms defining the target nodes for migration are included in the task allocation.

Algorithms that are tightly bound to an application are not discussed. The presented distribution aspects are selected due to their generality for different types of WSNs and applications. We omit, for example, data fusion and data aggregation that are beneficial only for applications that gather data to a centralized storage.

In this paper we review the application distribution for WSNs focusing on distribution implemented in *systems software*. By systems software we mean software components providing application-independent services and managing node resources. The proposed solutions vary according to tools provided, requirements placed on the underlying platforms, and targeted applications and environments. However, the current proposals lack an integrated solution providing a distributed operating environment for WSN applications. This approach would lead to a more efficient usage of resources.

This paper is organized in two main parts as follows. The first part describes the basics of objectives, challenges, and systems software solutions of WSNs. In addition, a summary of WSN application proposals is presented in order to define requirements. The second part starting in Section 3 contains the survey of distribution proposals followed by their analysis in Section 4. Finally, conclusions are given in Section 5.

2. OVERVIEW OF WSNs

In order to give an overview of WSN applications, we review some examples and their characteristics. These are listed in Table 1. The selection is mainly based on prototype implementations and thus all the scopes of WSNs might not be represented.

The first column in Table 1 lists the applications and the second classifies them according to the main task. The third column presents the requirements set by the application. The networking requirements in terms of data amount and frequency are defined in the fourth column, while the last column gives the scale and density of the application.

Most of the applications gather, evaluate, or aggregate data from different types of sensors. Major differences are in networking requirements and complexity. Unfortunately, accurate values or limits to these properties are not often reported, which complicates a fair comparison.

The nature of applications listed in Table 1 varies, but at least four main tasks can be identified [28]. *Monitoring* is used to continually track a parameter value in a given location, and *event detection* recognizes occurrences of events. *Object classification* attempts to identify an object or its type and *object tracking* traces movements of an object.

For the presented applications, the "worst-case" WSN would comprise of an extensive number of nodes with varying density and a network topology that constantly changes due to the errors in communication, mobility of nodes, and inactive nodes [3]. To complete complex tasks in the scenario, the application requires distributed processing within the network.

In our view, WSN application quality of service (QoS) is constructed from network lifetime, network load, accuracy of data, and fault tolerance. Network load in this case comprises of the required data latency, throughput, and reliability. WSN protocols and their functions are adapted according to the QoS requirements. Currently, security is a QoS issue that is often omitted in WSNs. The natural reason is that security requires too much resources [2].

For the rest of the paper we define an *environmental monitoring application* that is used for the analysis of the proposed solutions. For clarification, we refer to the application as *EnvMonitor*. The main task of the application is the constant gathering of location-dependent information within a defined area. In addition to the passive monitoring involved in the environmental monitoring applications in Table 1, *EnvMonitor* consists of active monitoring tasks reacting to condition changes in WSN. The passive monitoring data are gathered to a central storage and aggregated during the routing. Active in-network monitoring tasks execute signal processing algorithms locally in order to determine threshold values for temperature and humidity. When a threshold is reached, a set of predefined actions modifying the application QoS and the communication topology taken. The modifications alter the requirements for data composition, accuracy, and latency. The priority of active monitoring tasks precedes passive monitoring.

2.1. Systems software for WSNs

A general-purpose operating system (OS) is an example of systems software. Early WSNs have not included systems software due to scarce resources and simplicity of applications. However, complex applications require systems software because it eases the control of resources and increases the predictability of execution. The heterogeneity of platforms can be hidden under common interfaces provided by the software. Still, the major disadvantages are heavy computation and memory usage.

Table 1: Examples of prototyped applications for WSNs.

Application	Type	Requirements	Data amount and frequency	Scale and density
Great Duck Island [14]	Environmental monitoring	Data archiving, Internet access, long lifetime	Minimal, every 5–10 min, 2–4 h per day	32 nodes in 1 km^2
PODS in Hawaii [15]	Environmental monitoring	Digital images, energy-efficiency	Large data amounts, infrequently	30–50 nodes in 5 hectares
CORIE (Columbia River) [16]	Environmental monitoring	Base stations, lifetime	Moderate data amounts, infrequently	18 nodes in Columbia River
Peek value evaluation [17]	Environmental monitoring	Collaborative processing, minimal network traffic	Moderate data amounts, periodically	Case dependent
Flood detection [18]	Environmental monitoring	Current condition evaluation	50 bytes every 30 s	200 nodes 50 m apart
SSIM (artificial retina) [19]	Health	Image identification, realtime, complex processing	Large data amounts, frequently every 200 ms	100 sensors per retina
Human monitoring [20]	Health	Quality of data, security, alerts	Moderate data amounts, depend on the human stress level	Several nodes per human
Mountain rescue [21]	Health	Communication intensive	Large data amounts in high frequency	One per rescuer in mountain area
WINS for military [22]	Military	Target identification, realtime, security, quality of data	Large data amounts, infrequently	Several distant nodes
Object tracking [23]	Military	Collaborative processing, realtime, location-awareness	Large data amounts with high frequency near an object	7 (prototype) nodes in proximity
Vehicle tracking [24]	Military	Identification and coordination, realtime	Large data amounts every 8 s near an object	1024 nodes in 40 km^2
Intelligent input/output [25]	Home entertainment	Communication intensive	Large data amounts with high frequency	One node per input device
WINS condition monitoring [22]	Machinery monitoring	Data aggregation, machinery lifetime projection	Depend on machinery complexity and its current status	Few nodes per machinery
Smart kindergarten [26]	Education	Video streaming, identification, location-awareness	Large data amounts in variable frequencies	Tens of sensors, indoor
Smart classrooms [27]	Education	Context-sensing, data exchange	Large data amounts in random frequency	Several nodes in classroom

The systems software for WSNs implements *single node control* and *network-level distribution control*. The single node control software implements the low-level routines in a node, whereas the network-level distribution control manages application execution within several nodes.

Single node control

The single node control operates on a physical node depicted in Figure 1. A processing unit consists of CPU, storage devices, and an optional memory controller for accessing the instruction memory of the main CPU. A sensing unit consists of sensors and an analog-to-digital converter (ADC). A transceiver unit enables the communication with other sensor nodes. A power unit can be extended by a power generator that harvests energy from environment. Other peripheral devices, like actuators for moving the node and location finding systems, are attached to the node depending on the application requirements [3].

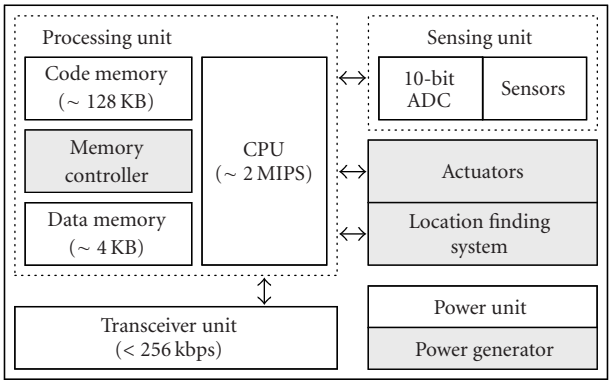


Figure 1: Reference hardware platform architecture of a sensor node.

The reference values in Figure 1 are the resources available in MICA2 mote [29]. The power consumption of a node is in order of mW when active and in order of μW when the node is in sleep. The power unit is typically an AA battery or similar energy source.

The single node control is accomplished by OS or virtual machine (VM). In the reference platform, OS is executed on the main CPU and it uses the same instruction and data memories as applications. Services implemented by OS include scheduling of tasks, interprocess communication (IPC) between tasks, memory control, and possible power control in terms of voltage scaling and component activation and inactivation. OS provides interfaces to access and control peripherals. The interfaces are typically associated with layered software components with more sophisticated functionality, for example a network protocol stack.

Network-level distribution control

Distribution control relies on networking. Figure 2 depicts an example protocol stack for WSN in comparison to two widely utilized stacks, the OSI model [1] and a distributed system in a wireless local area network (WLAN). In a WLAN computer, the TCP/IP stack is used through a sockets application programming interface (API). The WLAN adapter that contains the medium access control (MAC) protocol and the WLAN radio is accessed by a device driver.

There is no unified protocol stack for WSNs and most of the proposed stacks are just collections of known protocol functions. At the moment, the IEEE 1451.5 Wireless Sensor Working Group [30] is standardizing the physical layer for WSNs with an intention to adapt link layers from other wireless standards, for example, Bluetooth [31], IEEE 802.15.4 low-rate wireless personal area network (LR-WPAN) [32], or IEEE 802.11 WLAN [33]. Other types of networks posing common characteristics with WSNs are mobile ad hoc networks (MANETs) [34] targeted to address mobility.

In WSNs, the essential protocol layers are the MAC protocol on the data link layer and the routing protocol on the network layer. The MAC protocol creates a network topology and shares the transmission medium among sensor nodes. The topology in WSNs is either flat, in which all sensor nodes are equal, or clustered, in which communication is controlled by cluster headnodes. The routing protocol allows communication via multihop paths. A transport protocol that implements end-to-end flow control is rarely utilized in WSNs. The middleware layer is equivalent to the presentation layer in the OSI model [1].

For WSNs, the development of a distributed environment requires the consideration of all four distribution aspects. The control actions are taken according to the application QoS. The distribution aspects are typically implemented on the middleware layer on top of OS. Thus, the middleware component can reside in different types of platforms. In addition to OS routines, the middleware utilizes networking interface to implement communication between its own instances on different sensor nodes. Some distribution aspects can also be implemented directly by OS.

3. SURVEY OF DISTRIBUTION PROPOSALS

Numerous technologies for the service discovery and remote task communication are available for computer networks. The task migration is typically a transfer of a binary code image or a Java applet. In computer networks, the task allocation is often not the main concern as resources are sufficient. Even though not directly applicable for WSNs, the computer network technologies define the basic paradigms and algorithms for the application distribution.

Other types of wireless ad hoc networks, like MANETs and Bluetooth, have common characteristics with WSNs. First, communication in these networks is very similar to WSNs. Second, the resource constraints must be considered, even though the limits are looser than in WSNs. For this reason we include technologies proposed for MANETs and Bluetooth in our assessment of WSN proposals.

A distinct categorization of proposed solutions for WSNs cannot be made since a proposal typically present a more complete architecture addressing several distribution aspects. Therefore, we categorize the proposals according to their system architecture to OSs, VMs, middlewares, and stand-alone protocols.

3.1. Architectural paradigms

Figure 3 presents three architectural paradigms for distribution, which are *client-server*, *mobile code*, and *tuple space*. In computer networks, the client-server architecture is applied for the service discovery and remote task communication. It consists of one or multiple servers hosting a set of services and clients accessing these. A directory service is maintained at the server in the service discovery. In the remote task communication, a client outsources a task processing to a server. Two alternatives are available, remote procedure calls (RPCs) and object-oriented remote method invocations (RMIs). As the internal data and state of objects are accessed only through the object interface, RMI achieves better abstraction and fault tolerance. In addition, objects can be cached and moved [35].

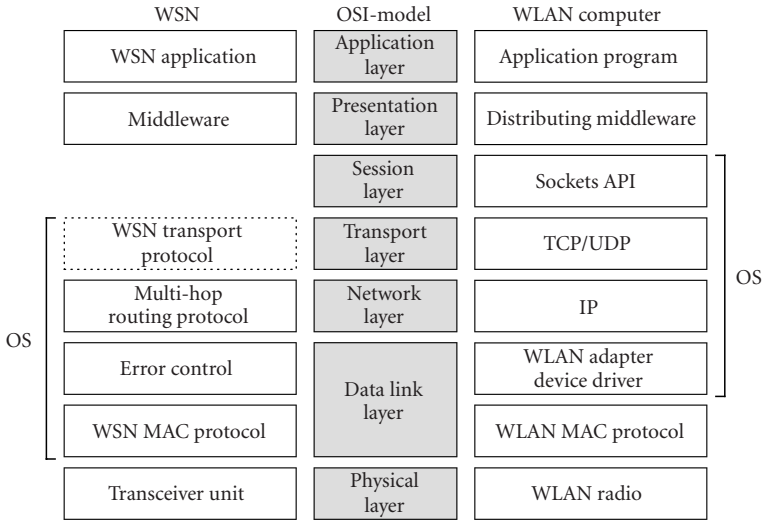


Figure 2: OSI model, WSN, and distributed system in WLAN protocol layers.

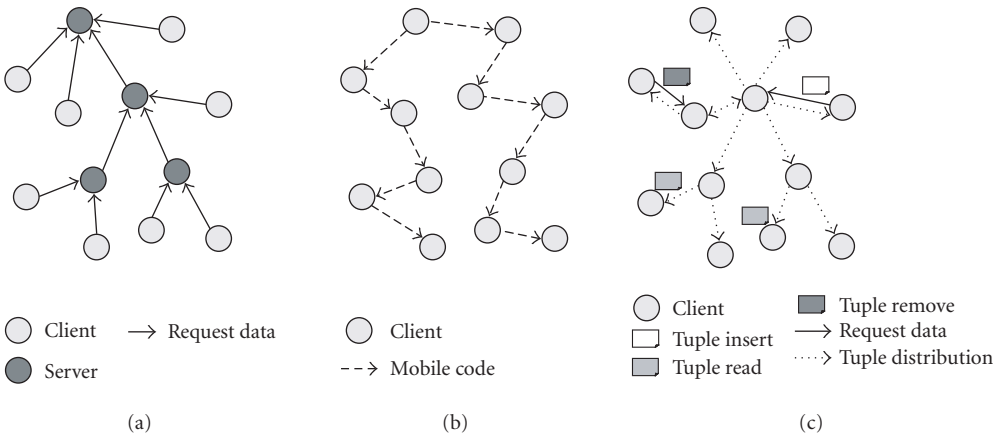


Figure 3: Three architectural paradigms for distribution: (a) client-server, (b) mobile code, and (c) tuple space.

Differences in programming languages and platforms must be hidden in the remote task communication. Stub procedures are generated for this from interface definitions. A stub procedure at the client marshals a procedure call to an external data presentation, which is then unmarshalled back to a primitive form at the server [35].

In the mobile code paradigm, instead of moving data from a client to a server for processing, the code is moved to the data origins, and data are then processed locally. A *mobile agent* is an object that in addition to the code carries its state and data. Furthermore, mobile agents make migration decisions autonomously. They are typically implemented on top of VMs for platform independency [36].

The concept of tuple space was proposed originally in Linda [46] for the remote task communication, but it is applicable also for the service discovery. Tuples are collections of passive data values. A tuple space is a pool of shared information, where tuples are inserted, removed, or read. Data are global and persistent in the tuple space and remain until explicitly removed. In the tuple space, a task does not need to know its peer task, tasks do not need to exist simultaneously, and they do not need to communicate directly.

3.2. Computer networks

Service location protocol (SLP) [47], Jini [48], universal plug and play (UPnP) [49], and secure service discovery service (SDS) [50] implement a client-server architecture service discovery in computer networks. The tuple space is utilized in JavaSpaces [51] on top of Jini and in TSpaces [52]. For the remote task communication, Sun RPC [53] and distributed computing environment (DCE) [54] are well-known RPC technologies. The best-known object-oriented technologies are common object request broker architecture (CORBA) [55], Java RMI [56], and Microsoft's distributed common object model (DCOM) [57]. The mobility of terminals is addressed in Mobile DCE [58], Mobile CORBA [59], and Rover Toolkit [60]. Schedulers for computer clusters implement task allocation within a cluster by allocating tasks to the most applicable resources [61].

Table 2: Implemented distribution aspects in single node proposals.

Proposal	Target network	Resource requirements (CPU/code memory/ data memory)	Service discovery	Task allocation	Remote task communication	Task migration
OS-based architectures						
EYES OS [37]	WSN	1 MHz / 60 KB / 2 KB	Resource requests	Not supported	RPC	Not supported
BTnodes [38]	WSN	8 MHz/ 128 KB/ 64 KB	Tuple space	Not supported	Callbacks	Smoblets
TinyOS [39]	WSN	8 MHz/ 128 KB/ 4 KB	Not supported	Not supported	Active messages	Not supported
BerthaOS [40]	WSN	22 MHz/ 32 KB/ 2,25 KB	Not supported	Not supported	BBS	Binary code
MOS [25]	WSN	8 MHz/ > 64 KB/ > 1 KB	Not supported	Not supported	Not supported	Binary code download
QNX [41]	LAN	33 MHz/ 100 KB/ N/A	Network manager	SMP scheduler	Message passing	Not supported
OSE [42]	LAN	N/A/ 100 KB/ N/A	Hunting service	Not supported	Phantom process	Not supported
VM-based architectures						
Sensorware [17]	WSN	N/A/ 1 MB/ 128 KB	Not supported	Script population specification	Not supported	TCL script migration
MagnetOS [43]	WSN	N/A / N/A / N/A	Not supported	Automatic object placement	DVM [44]	Mobile Java objects
Maté [45]	WSN	8 MHz/ 128 KB/KB	Not supported	Not supported	Not supported	Code capsule update

Distribution technologies designed for computer networks are typically both computation and communication intensive and cannot be implemented on sensor nodes. They are based on the client-server architecture and use detailed specifications for services and interfaces. These technologies do not consider the possible mobility or unavailability of sensor nodes. While mobility is addressed in Mobile DCE, Mobile CORBA, or Rover toolkit, these still rely on the client-server architecture from DCE and CORBA.

3.3. Distribution proposals for WSNs

From systems software proposals for WSNs, OSs and VMs implement the single node control and middleware architectures implement the network-level distribution control. These can be supported by stand-alone protocols that address only a single distribution aspect. We contribute the WSN proposals according to distribution aspects they implement.

OS-based architectures

The distribution aspects implemented in OSs are listed in Table 2. In addition, the second column defines the type of a network OS is targeted for, while the third one gives OS resource requirements. In WSNs, OSs implement a very limited set of services and they are fairly primitive in their nature. As shown in Table 2, the remote task communication is addressed typically by providing a simple method for RPC. The service discovery is rarely implemented in OS but on a higher system services layer that is associated to OS. Tasks migrate as binary code, because OSs do not support code interpreting.

The service discovery is implemented in EYES OS [37] on a distributed services layer above the OS by utilizing resource requests to neighbor nodes. Also Bluetooth smart nodes (BTnodes) [38] implement distribution in system services above a lightweight OS. BTnodes use the tuple space to implement the service discovery. The task allocation is not implemented in any of the proposals.

A client-server type RPC is applied to the remote task communication in TinyOS [39], BerthaOS (for Pushpin nodes) [40], and in EYES OS. In the component-based TinyOS, the handler name of the remote component and required parameters are encapsulated in a TinyOS active message. BerthaOS uses bulletin board system (BBS) for IPC and nodes can post messages also to BBS of a neighbor node. In EYES OS, the basic RPC between neighbor nodes is applied. BTnodes use the tuple space also for information sharing and for sending notifications to callbacks routines.

The task migration as binary code is possible in BetrhaOS and in MultimodAI NeTworks of In-situ Sensors (MANTIS) OS (MOS) [25]. BerthaOS allows the in-network initiation of transfers and checks the code integrity using a simple checksum, but neither it nor MOS considers the vulnerability of the system to malicious code. In BTnodes, precompiled Java classes, smoblets, are able to migrate but they must be executed on more powerful platforms.

Embedded OSs and RealTime OSs (RTOS), like QNX [41] and OSE [42], support service discovery and remote task communication in OS services. In QNX, the network of computers is abstracted to a single homogenous set of resources. QNX uses message passing to implement IPC and hides remote locations in process and resource managers.

The local managers interact with a network manager that handles name resolution. OSE uses stub procedures, referred to as phantom processes, for the remote task communication. A phantom process uses a link handler to communicate with the peer phantom process on the remote node. The remote node is discovered by a hunting system service that broadcasts service requests to the network.

From these proposals, QNX and OSE offer a distributed environment for applications, but they require more efficient sensor node platforms. Their resource requirements shown in Table 2 do not contain all the components required for the implementation of the distributed environment. The resource requirements set by other OSs are in the same order of magnitude. All the proposed OS architectures implement the single node control over the application tasks of *EnvMonitor*. The most applicable environment for *EnvMonitor* is available in BTnodes, where the tuple space implements service discovery and callbacks and smoblets support in-network distributed processing.

VM-based architectures

Compared to OSs, VMs offer hardware platform independency and substitute the lack of hardware protection by the protection implemented in code interpreters. The distribution aspects, target network, and required resources of VM architectures are categorized in Table 2. As shown, the mobile code is a common approach to distribution, whereas service discovery is not supported.

The task allocation is supported by Sensorware [17] and MagnetOS [43]. The population of tool command language (TCL) scripts in Sensorware is specified in the scripts themselves. MagnetOS utilizes automatic object placements algorithms that adaptively attempt to minimize communication by moving Java objects nearer to the data source. The remote task communication is addressed only in MagnetOS that relies on distributed VM (DVM) [44]. DVM abstracts network of computers to a single Java VM (JVM).

As depicted in Table 2, the mobile code is a TCL script in Sensorware, a custom bytecode capsule in Maté [45], and a Java object in MagnetOS. The size of the TCL scripts and especially the Maté code capsules is small compared to the size of Java objects. In Maté that operates on top of TinyOS a new code capsule is sent in TinyOS active messages to all nodes.

From the proposed solutions, Sensorware and MagnetOS implement task migration and task allocation, whereas in Maté only the latest code version is updated to all nodes. Implementation of MagnetOS on sensor nodes is not possible, Sensorware sets considerable requirements for underlying platforms, and Maté is implemented to very resource constrained nodes.

Like OSs, these proposals implement the single node control for *EnvMonitor*. From these proposals, Sensorware is the most suitable for *EnvMonitor* due to its migration, allocation, and task coprocessing capabilities. However, the control for these actions must be implemented by the application scripts.

Middleware architectures

Middleware architectures implement a higher abstraction level environment for applications. Generally, three different approaches in WSN middlewares can be identified. First, a middleware coordinates the task allocation based on the application QoS. Second, WSN is abstracted to a database that supports query processing. Third, a middleware controls application processing in the network based on the current context of surrounding environment. The context depends on the location, nearby people, hosts, and devices, and the changes in these over time [62]. The target network and distribution aspects for proposals are listed in Table 3.

Application QoS is applied for controlling the task allocation in the configuration adaptation of the middleware linking applications and networks (MiLAN) [20], in the resource management of the cluster-based middleware architecture for WSNs [63], and in QoSProxies of the QoS-aware middleware for ubiquitous and heterogeneous environments [64]. The cluster-based middleware and MiLAN adapt also the network topology. The QoSProxy selects an application configuration matching available resources and makes resources reservations to guarantee the specified QoS for that configuration. Both MiLAN and QoS-aware middleware adopt service discovery protocols from computer network solutions. QoS-aware middleware requires a more powerful platform than the other two.

A database approach is taken in sensor information and networking architecture (SINA) [24], in TinyDB [65] on top of TinyOS, and in Cougar [66]. In SINA, database queries are injected to network as sensor querying and tasking language (SQTL) [71] scripts. These scripts migrate from node to node depending on their parameters. The task allocation in SINA is implemented by a sensor execution environment (SEE), which compares SQTL script parameters to node attributes and executes script only if these match. In TinyDB and Cougar, the task allocation is implemented by a query optimizer that determines energy-efficient query routes. The query plans generated by the query optimizer are parsed in the nodes and then executed accordingly. TinyDB supports also event-based queries that are initiated in-network on the occurrence of an event.

Application adaptation based on the current context is performed by Linda in a mobile environment (LIME) [67], mobile agent runtime environment (MARE) [21], and reconfigurable context-sensitive middleware (RCSM) [27]. Service discovery is implemented by the tuple space in LIME and MARE. RCSM uses a custom RKS [68] protocol that reduces communication by advertising services only if they can be activated in the current context and potential clients are in the vicinity. LIME implements task allocation by reactions added to tuples. The MARE control manages nearby mobile agents and allocates tasks to the agents. RCSM ADaptive object containers (ADC) activate tasks in an appropriate context.

The tuple space in LIME and MARE is used also for the remote task communication. LIME supports also location-dependent recipient identification. RCSM utilizes RCSM

Table 3: Implemented distribution aspects in middleware and stand-alone protocol proposals.

Proposal	Target network	Service discovery	Task allocation	Remote task communication	Task migration
Middleware architectures					
MiLAN [20]	WSN	SLP, Bluetooth SDP	Configuration adaptation	Not supported	Not supported
Cluster-based middleware [63]	WSN	Not supported	Resource management	Not supported	Not supported
QoS-aware middleware [64]	MANET	SLP/Jini/SDS	QoSProxy	Not supported	Not supported
SINA [24]	WSN	Not supported	Attribute matching in SEE	Not supported	SQTL scripts
TinyDB [69]	WSN	Not supported	Query optimizer, event-based queries	Not supported	Not supported
Cougar [66]	WSN	Not supported	Query optimizer	Not supported	Not supported
LIME [67]	MANET	Tuple space	Context reaction	Tuple space	Mobile Java objects
MARE [21]	MANET	Tuple space	MARE control	Tuple space	Mobile Java objects
RCSM [27]	MANET	RKS [68]	Adaptive object containers	R-ORB	Not supported
Stand-alone protocols					
GSD [69]	MANET	Service groups	Not supported	Not supported	Not supported
Bluetooth SDP [31]	Bluetooth	Clients and servers	Not supported	Not supported	Not supported
Task migration in [70]	WSN	Not supported	Not supported	Not supported	Edit scripts

context-sensitive object request broker (R-ORB) that adapts basics from CORBA ORB. Both LIME and MARE utilize mobile agents implemented as Java objects for the task migration.

Unlike OSs and VMs, most of the middleware architectures implement the network-level distribution control but do not address the single node control. Middlewares relying on the application QoS specification address mainly task allocation, but leave other aspects to external components. The database abstraction is applicable to a certain type of applications, like *EnvMonitor*, but the expressivity of the SQTL scripts in SINA, the event-based queries in TinyDB, and especially the query processing capabilities in Cougar do not support complex in-network processing. As can be seen from Table 3, context-aware proposals cover distribution aspects extensively. They implement extensive environment for *EnvMonitor* but their resource requirements are too high for sensor nodes.

Stand-alone protocols

The environment provided by OSs, VMs, or middleware architectures can be supported by stand-alone protocols implementing dedicated functions. We do not cover WSN MAC and routing protocols but focus on protocols that implement any of the four distribution aspects. The protocols and their target networks are listed in Table 3.

The group-based service discovery protocol (GSD) for MANETs [69] and the Bluetooth service discovery protocol (SDP) [31] implement the service discovery. In GSD, terminals advertise their services and nearby service groups within the distance of n hops. Service requests are forwarded towards the service provider based on group advertisements. A Bluetooth terminal maintains information about its services in an SDP server. Searching and querying for existing services are performed by an SDP client that queries one server at a time.

An approach for minimizing the transferred binary code size on the task migration is proposed in [70]. The proposal transmits only the differences between the existing and the new code. The algorithm is adopted from the *diff* command of UNIX.

These protocols can be used as separate components for *EnvMonitor*, but none of them provides a complete environment. GSD is communication intensive due to the multi-hop advertisements. Bluetooth SDP does not support broadcast queries, which restricts its applicability in large WSNs. The task migration proposed in [70] cannot be initiated in WSNs due to the complexity of the algorithm and the lack of integrity checking.

4. ANALYSIS OF PROPOSALS

A comprehensive comparison of the proposals is problematic due to the diversity of platforms, applications, and implementations. However, the requirements for each distribution aspect are similar, which makes their assessment possible. In the analysis, we concentrate on the proposals targeted for WSNs.

Table 4: System testing and validation environments for distribution proposals.

Proposal	Test environment	Simulation and testing tools	Prototype platform	Result accuracy	Published results
OS-based architectures					
TinyOS [39]	Prototype	TOSSIM [72]	Motes	Accurate	Component sizes, OS routine delays, computation costs
BerthaOS [40]	Prototype	None	Pushpin	None	Functionality mentioned
EYES OS [37]	None	None	None	None	None
MOS [25]	Prototype	PC emulator XMOS [25]	Nymph	Moderate	Memory and power consumption, test application performance results
BTnodes [38]	Prototype	None	Micro-size BTnodes	Moderate	Component sizes, energy consumption
VM-based architectures					
Sensorware [17]	Prototype	SensorSim [73]	Linux IPAQ	Accurate	Framework size, execution delays, energy consumption
MagnetOS [43]	Windows/Linux JVM	Custom packet-level simulator	PC	None	Internal algorithm comparison in simulator
Maté [45]	Prototype	TOSSIM [72]	TinyOS mote	Accurate	Bytecode overhead, installation costs, code infection performance
Middleware architectures					
MiLAN [20]	None	None	None	None	None
Cluster-based middleware in [63]	Algorithm simulation	Custom simulator	None	None	Heuristic resource allocation, algorithm performance
Qos-aware middleware in [64]	None	None	None	None	None
SINA [24]	Simulations	GloMoSim [74]	None	Poor	SINA networking overhead, application performance
TinyDB [65]	Simulations, prototype	Custom environment	TinyOS mote	Accurate	Query routing performance in simulations, sample accuracy and sampling frequency in prototypes
Cougar [66]	None	None	None	None	None
LIME [67]	JVM	None	PC	Poor	Approximations about Java code size
MARE [21]	JVM	None	PDA	Poor	Service discovery performance
RCSM [27]	Prototype	None	PDA with custom hardware	RCSM poor, RKS accurate	RCSM memory consumption, RKS size, communication, energy consumption
Stand-alone protocols					
GSD [69]	Simulations	GloMoSim [74]	None	Poor	Influence of internal parameters on service discoverability
Task migration in [70]	PC	None	Tested in EYES nodes	Accurate	Algorithm performance, influence of internal parameters

4.1. Testing and validation of WSN proposals

Discussed WSN architectures vary in their complexity and requirements. In order to provide a scope for the assessment of proposals, their testing and validation environments are presented in Table 4. The test environment is presented in the second column. The simulation and testing tools and prototype platforms identify the proposal validation tools and test platform. The published results and their accuracies are listed in the last two columns.

Generally, prototypes exist for the single node architectures and their results are accurate including information required for comparison. Instead, on the middleware layer, proposals are evaluated by simulations or not at all. The simulation results are inaccurate as they compare only the internal algorithms and do not give any information for a general comparison. Of course, exceptions exist in both cases.

Even though some of the presented results in Table 4 are accurate and their scope is adequate, the direct comparison

Table 5: Characteristics of technologies implementing service discovery.

Technology	Communication	Scalability	Fault tolerance	Requirements	Benefits (pros)	Problems (cons)
Resource requests	Requests to neighbors	Restricted to neighbors	Broadcasted to all neighbors	Resource declaration	One-hop communication	Scalability
Tuple space	Tuple operations	Balancing between memory and scale	Redundant information	Memory pool in each node	Source and target independency	Communication/ memory load
Network manager	Name resolution requests to manager	Local manager area, but extensible	Possibly redundant network managers	Resource managers, register to manager	Scalability due to naming	Name resolution, communication load
Hunting service	Broadcast hunt service requests	Not restricted	Lost services can be rehunted	Remote service identification	Lightweight after initiation	First hunt latency and communication load
Bluetooth SDP	Peer-to-peer link	Only nearby nodes one at a time	Service information only in the host	Bluetooth protocol stack	Querying for available services	Scalability, no broadcast
RKS	Advertises for potential clients	Only to nearby clients	Advertisements when context and clients applicable	Context definitions for services	Advertisements	Scalability
GSD service groups	Service and group advertisements	n-hop diameter, but groups span wider	Redundant information	Service registration	Request routing based on group advertisements	Communication load (both advertisements and requests used)

of distribution performance is not possible. The prototype platforms vary in their efficiency, the simulators in their accuracy, and the test applications in their requirements and functionality. As the area is evolving rapidly, generally accepted benchmarks would ease the comparison of the proposals. However, the definition of general-enough benchmarks for WSNs is difficult due to their application-specific nature.

4.2. Comparison of technologies

We classify the technologies for each distribution aspect separately. The classification dimensions for a technology are *communication mechanism*, *scalability* to large WSNs, *fault tolerance*, and *requirements* that must be met before the technology can be used. For each technology, we also assess its pros and cons in general. These dimensions offer tools for the evaluation of the robustness and applicability of a technology for different kinds of WSNs and applications.

Service discovery

The classification of the service discovery technologies in the proposals according to the defined dimensions is presented in Table 5. From the presented solutions, all but the tuple space and GSD rely on client-server architecture. Still, the network manager is the only centralized server. In general, two problems can be identified from the proposals. They either have a restricted scalability or require intensive communication.

The client-server technologies that are limited to nearby nodes do not scale to large WSNs. GSD and the tuple space both scale to large networks but they require more communication for locating a service. However, in both technologies the communication load can be decreased by increasing the number of hops, to which the service information is distributed. This increases the communication during the initiation but reduces it during the discovery, with the cost of increased memory consumption.

Task allocation

The technologies that implement a mechanism for the task allocation and the characteristics of each technology are listed in Table 6. As peer-to-peer communication is not needed in all the technologies, the communication mechanism is replaced by a more general outlining of the taken approach. As shown in Table 6, the variance of technologies is greater than in the service discovery. As most of the technologies are middleware layer implementations, the main reason for the variance is the three different approaches taken at that layer.

The most promising approach is the task allocation based on application QoS. It does not restrict the implementation of tasks nor rely on the surrounding context. Instead, it enables the adaptation of application operations depending on the current application requirements. The application requirements can be adjusted depending on the output of the application itself, which makes the technologies adaptive to changing conditions. Generally, application-QoS-based technologies require a central control for the task allocation, but a distributed control lacks similar adaptability.

Remote task communication

From the remote task communication technologies classified in Table 7, most utilize traditional RPC or RMI that are tailored for resource constrained environments. The tuple space and callbacks, which also utilize tuple space, are the only exceptions.

In general, the technologies either are restricted in their scalability or burden memory and communication resources. The problem in RPC and RMI technologies is the requirement for a client to know the server. In the tuple space and callbacks this is not required. In the callbacks, the message

Table 6: Characteristics of technologies implementing task allocation.

Technology	Approach	Scalability	Fault tolerance	Requirements	Benefits (pros)	Problems (cons)
SMP scheduler	Scheduling of tasks to free resources	Not restricted	Redundant	High-speed bus, shared memory	Efficiency and transparency	Inapplicable requirements
Script population specification	Specification in migrating scripts	Not restricted	Multiple copies in network	Control in application scripts	No control required	Expressivity of specification
Automatic object placement	Activating and moving objects near to source	Not restricted	Multiple agents available	Object placement algorithms	Reduced data communication	Complexity
Configuration adaptation	Mapping tasks to available resources	Not restricted	Changes active nodes adaptively	Feasibility analysis, state updates	Application QoS consideration	Control communication
Resource management	Heuristic algorithm balancing load [75]	Restricted to a cluster	Continuous allocation	Control messages	Network lifetime maximizing	Algorithm complexity
QoSProxy	Component and service adaptation for resources and application QoS	Network-wide in small networks	Adaptation according to conditions	Application QoS specification	QoS adaptation dynamically to available resources	Server required, complexity and communication
Attribute matching in SEE	Matching script attributes to node parameters locally	Not restricted	Multiple copies in network	Accurate attribute specifications	Local late binding	Restricted expressivity
Query optimizer	Optimizing query routing to network	Optimization in gateway node	Redundancy in queries	Disseminated query plans	Only required set of nodes activated	Networking load of query plans
Event-based queries	Initiate query on occurrence of event	Not restricted	Possibly several event detectors	Event identification capability	In-network reaction	Loading of event source node
Context reaction	Reactions on tuples and executed on matching context	Reaction restricted to a location	Redundancy in tuple space	Location identifying	Task executed only when its context is applicable	Scalability
MARE control	Nearby agents form an execution environment	Restricted to nearby agents	Possible redundancy	Agent managers controlling agents	Agent cooperation in complex tasks	Scalability
Adaptive object containers	ADC activates tasks in correct context	Not restricted	Possible redundancy	Context interface specifications	Only applicable tasks activated	Complex context specifications

is sent to a registered callback function whenever the value of a tuple changes. The tuple space does not support such interests on tuples. Like in the service discovery, the communication and memory load of the tuple space are adjustable.

Task migration

The technologies for the task migration are summarized in Table 8. Most of the technologies rely on the mobile agents due to their fault tolerance and smaller physical size. Three technologies rely on binary code in order to lessen the computation load caused by the agent interpreting.

In order to use binary code in the task migration, the possible errors during transfers and malicious attacks must be managed. The edit script generation algorithm is too complex to be executed in nodes, thus making it inapplicable for dynamic WSNs. From the VM approaches, the TCL and SQTL scripts and Maté bytecode capsules are more lightweight than Java objects because of the complexity and memory requirements of JVM.

4.3. Suitability assessment

Generally, the OS and VM proposals support the remote task communication and the task migration but leave the task allocation and the service discovery to an application or other external components. On the contrary, middleware approaches concentrate on implementing the task allocation, leaving other aspects for the tuple spaces or some legacy protocols. MARE and LIME are the only proposals that cover all distribution aspects. However, the utilization of JVM and the distributed tuple space requires resources that are not generally available in current WSN platforms.

We assess the applicability of the proposals for *EnvMonitor*. For a fair comparison, we separately compare the approaches for node platforms with enough resources, and then for platforms with limited resources defined in Figure 1. The main aspect considered in the assessment is the completeness of the operating environment provided for the application. In a complete environment, the application does not need to consider its distributed nature but the distribution is handled by the systems software. Further, the adap-

Table 7: Characteristics of technologies implementing remote task communication.

Technology	Communication implementation	Scalability	Fault tolerance	Requirements	Benefits (pros)	Problems (cons)
Active messages	Remote handler, data encapsulation	Not restricted	N/A	Awareness of remote handler	Mapping to TinyOS event model	Handler name in ASCII
BBS	Message posting to neighbor BBS	Restricted to neighbor nodes	Message posted to all neighbors	Neighbor posting enabled by sender	One-hop communication	Scalability, memory load
EYES OS RPC	N/A	Restricted to neighbors	N/A	N/A	One-hop communication	Scalability
Callbacks	Callback registered to a tuple	Restricted to nodes sharing tuple space	Callback registered only in one node	Shared tuple space between nodes	Callback fired only on an event	Fault tolerance
Message passing	Custom networking (QNet) operations	Not restricted	Possibility for redundant messages	Name resolution	Mapping to local IPC	Network naming overhead
Phantom process	Messages sent by link handler	Not restricted	Possible secure channels	Created channel for communication	Mapping to local IPC	Required handshaking, communication load
DVM	Invocation redirection	Not restricted	N/A	Compile time script modification	Seamless IPC between objects	Communication and processing load
Tuple space	Tuple operations	Not restricted	Redundant	Shared tuple space between nodes	Distributed in space and time	Communication/memory load
R-ORB	Message-oriented communication	Requires nearby recipient	Activated when link available	Context sensing	Activated only in applicable context	Scalability

Table 8: Characteristics of technologies implementing task migration.

Technology	Communication	Scalability	Fault tolerance	Requirements	Benefits (pros)	Problems (cons)
Binary code	Binary code after negotiation	Only to one neighbor at a time	Simple checksum	Initiated by the binary code itself	Runtime initiation	Scalability, bit errors, binary size
Binary code download	Binary code from workstation	No in-network initiation	No protection	User initiates downloads	Possibility to update OS components	Errors, binary size, user interaction
Smoblets	Java applet modules	Execution only in laptops/PDAs	Java interpreter protection	Efficient platforms	Complex processing outsourcing	Executed only in efficient nodes
TCL script migration	TCL scripts	The scale specified in scripts	TCL interpreter protection	Injected to network by a user	Dynamic migration, small size of scripts	Complex population specifications
Mobile Java objects	Objects on top of JVM	Not restricted	Interpreter protection	Event initiating mobilization	Scalability	Communication and processing load
Code capsule updates	Small capsules in one active message	Script populated to all nodes in network	Maté interpreter protection	Injected to network by a user	Small size of scripts	No controlled migration
SQTL scripts	Custom query scripts	The scale specified in scripts	SEE interpreter protection	Injected to network by a user	Small size of scripts	Communication cost in broadcast
Edit scripts	Scripts containing changes to old code	No in-network initiation	Erroneous/missing scripts requested from neighbors	Generation of edit scripts in workstation	Small size of scripts	Complexity, no in-network operation

tivity of the proposals to changing conditions and the task allocation for extending network lifetime are emphasized.

For resource rich environments, MARE is the most suitable environment. The sensing and aggregation tasks in *EnvMonitor* can be allocated by the MARE control, and the active monitoring tasks can be implemented as mobile agents that are activated on demand.

For typical WSN platforms, MARE is not applicable due to its resource requirements. On the other hand, BTnodes fit to the restricted resources. The callbacks can be used to implement active monitoring tasks in *EnvMonitor*. The only aspect that is not supported by BTnodes is the task allocation so that the load is balanced between nodes.

4.4. Recommendations

From the systems software proposals for WSNs, OS and VM technologies implement the single node control and separate solutions for application distribution. The middleware proposals are applicable to the network-level distribution control. However, we argue that in WSNs, OS and middleware layers must be integrated to provide sufficient services within the constraints set by applications and platform resources.

In this kind of an approach, OS and middleware are inside the same framework so that information about OS internals and network topology is applicable to the middleware layer. Thus, this approach minimizes extra computation required for interfacing OS routines and communication due to the control signaling. Further, the middleware layer is aware of the influences of its actions at both the single node and network level. This awareness can be beneficial in the network-level power management and in the balancing of node loading.

For a sufficient environment for *EnvMonitor*, OS must implement a preemptive scheduling of tasks, a memory and power management, and a local IPC. The memory control should support static and dynamic memory and maintain information about available memory. We recommend the usage of a message-passing IPC because it is easily extended to the remote task communication. This kind of a general-purpose OS can be implemented on limited resources as shown in [25].

In addition to the local services, OS informs the middleware about the node energy and storage consumption, network role, associations, and nearby nodes and routes. An internal interface for the middleware to control tasks, power states, and network is implemented in OS. When all distribution aspects are implemented on the middleware layer, the components are able to utilize the information from each other more efficiently.

For service discovery we recommend the tuple space, since the pure client-server architecture is too static for WSNs. The resource and communication load of the tuple space can be diminished by selectively distributing tuple storing to nodes that use the tuple data and by dividing tuples to two-level hierarchies similar to GSD. The nodes that need a tuple for their operation can be identified with the support of task allocation. By sending only service group tuples to the distant nodes, less memory is needed but requests for tuples can still be routed accurately.

For the task allocation, the current application-QoS-based middleware proposals implement sufficient technologies. However, simpler algorithms that require less control communication should be used, even with the cost of accuracy.

For the remote task communication we recommend a simple approach that marshals the local message passing IPC to network packets. The remote nodes are identified by the service discovery. To make the delivery of a packet reliable, acknowledgements must be used. This is more lightweight than the tuple space, and the fault tolerance does not depend on the available recipients.

From our perspective, the task migration is required only in very dynamic applications, like object tracking. These applications require a VM-based environment. In OSs, the communication cost of the large binary transfers is extensive. Thus, the task migration should only be used when extremely necessary. The transfers must be protected with checksums and digital signatures, even though these are resource consuming.

We recommend also the usage of *virtual clusters*. A virtual cluster may follow the physical topology or it can be a set of adjacent nodes that have elected a single control entity. By storing detailed tuple information and performing task allocation within the boundaries of a virtual cluster, the communication and memory load can be diminished.

5. CONCLUSIONS

Our survey of WSN applications and their distribution shows that, despite many proposals, no common benchmarks nor detailed, large-scaled experiments have been published. The research seems to focus either on node implementations or theoretical work on distinct aspects, such as routing algorithms, without a realistic relation to physical platforms.

The systems software proposals are still evolving. Currently, they implement technologies and algorithms for application distribution but lack an approach combining a distributing middleware layer to OS providing a single node control. This kind of an approach is needed in order to implement a distributed operating environment, which supports application QoS and extends network lifetime, for resource scarce sensor nodes.

REFERENCES

[1] W. Stallings, *Data & Computer Communications*, Prentice-Hall, Englewood Cliffs, NJ, USA, 6th edition, 2001.

[2] J. A. Stankovic, T. E. Abdelzaher, C. Lu, L. Sha, and J. C. Hou, "Real-time communication and coordination in embedded sensor networks," *Proc. IEEE*, vol. 91, no. 7, pp. 1002–1022, 2003.

[3] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "A survey on sensor networks," *IEEE Commun. Mag.*, vol. 40, no. 8, pp. 102–114, 2002.

[4] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Computer Networks*, vol. 38, no. 4, pp. 393–422, 2002.

[5] K. Akkaya and M. Younis, "A survey on routing protocols for wireless sensor networks," *Elsevier Ad Hoc Network Journal*, vol. 3, no. 3, pp. 325–349, 2005.

[6] H. Karl and A. Willig, "A short survey of wireless sensor networks," Tech. Rep. TKN-03-018, Technical University Berlin, Berlin, Germany, 2003, available: http://www.tkn.tu-berlin.de/publications.

[7] D. Chen and P. K. Varshney, "QoS support in wireless sensor networks: a survey," in *Proc. International Conference on Wireless Networks (ICWN '04)*, pp. 227–233, Las Vegas, Nev, USA, June 2004.

[8] S. Tilak, N. B. Abu-Ghazaleh, and W. B. Heinzelman, "A taxonomy of wireless micro-sensor network models," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 6, no. 2, pp. 28–36, 2002.

[9] V. Raghunathan, C. Schurgers, S. Park, and M. B. Srivastava, "Energy-aware wireless microsensor networks," *IEEE Signal Processing Mag.*, vol. 19, no. 2, pp. 40–50, 2002.
[10] A. J. Goldsmith and S. B. Wicker, "Design challenges for energy-constrained ad hoc wireless networks," *IEEE Wireless Communications*, vol. 9, no. 4, pp. 8–27, 2002.
[11] D. Remondo and I. G. Niemegeers, "Ad hoc networking in future wireless communications," *Computer Communications*, vol. 26, no. 1, pp. 36–40, 2003.
[12] N. Xu, "A survey of sensor network applications," available: http://enl.usc.edu/~ningxu/papers/survey.pdf.
[13] C.-Y. Chong and S. P. Kumar, "Sensor networks: evolution, opportunities, and challenges," *Proc. IEEE*, vol. 91, no. 8, pp. 1247–1256, 2003.
[14] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson, "Wireless sensor networks for habitat monitoring," in *Proc. International Workshop on Wireless Sensor Networks and Applications (WSNA '02)*, pp. 88–97, Atlanta, Ga, USA, September 2002.
[15] PODS, A Remote Ecological Micro-sensor Network Project website, http://www.pods.hawaii.edu.
[16] CORIE website, http://www.ccalmr.ogi.edu/CORIE.
[17] A. Boulis, C.-C. Han, and M. B. Srivastava, "Design and implementation of a framework for efficient and programmable sensor networks," in *Proc. 1st International Conference on Mobile Systems, Applications, and Services (MobiSys '03)*, San Francisco, Calif, USA, May 2003.
[18] P. Bonnet, J. Gehrke, and P. Seshadri, "Querying the physical world," *IEEE Pers. Commun.*, vol. 7, no. 5, pp. 10–15, 2000.
[19] L. Schwiebert, S. K. S. Gupta, and J. Weinmann, "Research challenges in wireless networks of biomedical sensors," in *Proc. 7th ACM International Conference on Mobile Computing and Networking (MobiCom '01)*, pp. 151–165, Rome, Italy, July 2001.
[20] W. B. Heinzelman, A. L. Murphy, H. S. Carvalho, and M. A. Perillo, "Middleware to support sensor network applications," *IEEE Network*, vol. 18, no. 1, pp. 6–14, 2004.
[21] M. Storey, G. S. Blair, and A. Friday, "MARE: resource discovery and configuration in Ad hoc networks," *J. Mobile Networks and Applications*, vol. 7, no. 5, pp. 377–387, 2002.
[22] H. O. Marcy, J. R. Agre, C. Chien, L. P. Clare, N. Romanov, and A. Twarowski, "Wireless sensor networks for area monitoring and integrated vehicle health management applications," in *Proc. AIAA Guidance, Navigation, and Control Conference and Exhibit*, Portland, Ore, USA, 1999, Collection of Technical Papers. Vol. 1 (A99-36576 09-63).
[23] K. Römer , "Tracking real-world phenomena with smart dust," in *Proc. 1st European Workshop on Wireless Sensor Networks (EWSN '04)*, pp. 28–43, Berlin, Germany, January 2004.
[24] C.-C. Shen, C. Srisathapornphat, and C. Jaikaeo, "Sensor information networking architecture and applications," *IEEE Pers. Commun.*, vol. 8, no. 4, pp. 52–59, 2001.
[25] H. Abrach, S. Bhatti, J. Carlson, et al., "MANTIS: system support for MultimodAl NeTworks of In-situ Sensors," in *Proc. 2nd ACM International Workshop on Wireless Sensor Networks and Applications (WSNA '03)*, pp. 50–59, San Diego, Calif, USA, September 2003.
[26] M. B. Srivastava, R. R. Muntz, and M. Potkonjak, "Smart kindergarten: sensor-based wireless networks for smart developmental problem-solving enviroments," in *Proc. 7th ACM International Conference on Mobile Computing and Networking (MobiCom '01)*, pp. 132–138, Rome, Italy, July 2001.
[27] S. S. Yau, F. Karim, Y. Wang, B. Wang, and S. K. S. Gupta, "Reconfigurable context-sensitive middleware for pervasive computing," *IEEE Pervasive Computing*, vol. 1, no. 3, pp. 33–40, 2002.
[28] NIST Wireless Ad hoc Networks Project website, http://www.antd.nist.gov/wahn_ssn.shtml.
[29] MICA2 data sheet, available: http://www.xbow.com.
[30] IEEE P1451.5 Wireless Sensor Working Group website, http://grouper.ieee.org/groups/1451/5.
[31] Bluetooth Special Interest Group, "Bluetooth specification, version 1.1," February 2001.
[32] *Wireless Medium Control (MAC) and Physical Layer (PHY) Specifications for Low Rate Wireless Personal Area Networks (LR-WPAN)*, IEEE Standard 802.15.4, 2003.
[33] *Wireless LAN Medium Control (MAC) and Physical Layer (PHY) Specifications*, IEEE Standard 802.11, 1999.
[34] IETF Mobile Ad-hoc Networks Working Group website, http://www.ietf.org/html.charters/manet-charter.html.
[35] G. Coulouris, J. Dollimore, and T. Kindberg, *Distributed Systems: Concepts and Design*, Addison-Wesley, Boston, Mass, USA, 3rd edition, 2001.
[36] A. Fuggetta, G. P. Picco, and G. Vigna, "Understanding code mobility," *IEEE Trans. Software Eng.*, vol. 24, no. 5, pp. 342–361, 1998.
[37] P. J. M. Havinga, "System architecture specification," EYES Project Deliverable 1.1, available: http://www.eyes.eu.org/dissem.htm.
[38] J. Beutel, O. Kasten, F. Mattern, K. Roemer, F. Siegemund, and L. Thiele, "Prototyping wireless sensor networks with BTnodes," in *Proc. 1st European Workshop on Wireless Sensor Networks (EWSN '04)*, pp. 323–338, Berlin, Germany, January 2004.
[39] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, "System architecture directions for networked sensors," in *Proc. 9th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '00)*, pp. 94–103, Cambridge, Mass, USA, November 2000.
[40] J. Lifton, D. Seetharam, M. Broxton, and J. Paradiso, "Pushpin computing system overview: a platform for distributed, embedded, ubiquitous sensor networks," in *Proc. 1st International Conference on Pervasive Computing (Pervasive '02)*, pp. 139–151, Zurich, Switzerland, August 2002.
[41] QNX website, http://www.qnx.com.
[42] OSE website, http://www.ose.com.
[43] R. Barr, J. C. Bicket, D. S. Dantas, et al., "On the need for system-level support for ad hoc and sensor networks," *ACM SIGOPS Newsletter on Operating Systems Review*, vol. 36, no. 2, pp. 1–5, 2002.
[44] E. G. Sirer, R. Grimm, A. J. Gregory, and B. N. Bershad, "Design and implementation of a distributed virtual machine for networked computers," in *Proc. 17th ACM Symposium on Operating Systems Principles (SOSP '99)*, pp. 202–216, Kiawah Island, SC, USA, December 1999.
[45] P. Levis and D. Culler, "Maté: a tiny virtual machine for sensor networks," in *Proc. 10th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '02)*, pp. 85–95, San Jose, Calif, USA, October 2002.
[46] D. Gelernter, "Generative communication in linda," *ACM Trans. Programming Languages and Systems*, vol. 7, no. 1, pp. 80–112, 1985.
[47] E. Guttman, C. Perkins, J. Veizades, and M. Day, "Service location protocol, version 2," IETF, RFC 2608, June 1999.
[48] *Jini Architecture Specification*, version 2.0, Sun Microsystems, June 2003, available: http://www.sun.com/software/jini/specs.
[49] *UPnP Device Architecture*, Microsoft Corporation, June 2000, available: http://www.upnp.org/resources/documents.asp.
[50] S. E. Czerwinski, B. Y. Zhao, T. D. Hodes, A. D. Joseph, and R. H. Katz, "An architecture for a secure service discovery

service," in *Proc. 5th International Conference on Mobile Computing and Networking (MobiCom '99)*, pp. 24–35, Seattle, Wash, USA, August 1999.
[51] *JavaSpaces Service Specification*, version 2.0, Sun Microsystems, June 2003, available: http://www.sun.com/software/jini/specs.
[52] T. J. Lehman, A. Cozzi, Y. Xiong, et al., "Hitting the distributed computing sweet spot with TSpaces," *Computer Networks*, vol. 35, no. 4, pp. 457–472, 2001.
[53] R. Srinivasan, "RPC: remote procedure call protocol specification version 2," IETF, RFC 1831, August 1995.
[54] The Open Group Portal to World of DCE, website, http://www.opengroup.org/dce.
[55] Object Management Group website, http://www.omg.org.
[56] *Java RMI specification*, Sun Microsystems, 1997–2003, available: http://java.sun.com/products/jdk/rmi/reference/docs.
[57] M. Horstmann and M. Kirtland, "DCOM architecture," Microsoft Corporation, July 1997, available: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnanchor/html/dcom.asp.
[58] A. Schill and S. Kümmel, "Design and implementation of a support platform for distributed mobile computing," *Distributed Systems Engineering*, vol. 2, no. 3, pp. 128–141, 1995.
[59] S. Adwankar, "Mobile CORBA," in *Proc. 3rd International Symposium on Distributed Objects and Applications (DOA '01)*, pp. 52–63, Rome, Italy, September 2001.
[60] A. D. Joseph, J. A. Tauber, and M. F. Kaashoek, "Mobile computing with the rover toolkit," *IEEE Trans. Comput.*, vol. 46, no. 3, pp. 337–352, 1997.
[61] Supercluster.org website, http://www.supercluster.org.
[62] B. Schilit, N. Adams, and R. Want, "Context-aware computing applications," in *Proc. Workshop on Mobile Computing Systems and Applications (WMCSA '94)*, pp. 85–90, Santa Cruz, Calif, USA, December 1994.
[63] Y. Yu, B. Krishnamachari, and V. K. Prasanna, "Issues in designing middleware for wireless sensor networks," *IEEE Network*, vol. 18, no. 1, pp. 15–21, 2004.
[64] K. Nahrstedt, X. Dongyan, D. Wichadakul, and L. Baochun, "QoS-aware middleware for ubiquitous and heterogeneous environments," *IEEE Commun. Mag.*, vol. 39, no. 11, pp. 140–148, 2001.
[65] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "The design of an acquisitional query processor for sensor networks," in *Proc. 22nd ACM SIGMOD International Conference on Management of Data (SIGMOD '03)*, pp. 491–502, San Diego, Calif, USA, June 2003.
[66] Y. Yao and J. Gehrke, "The Cougar approach to in-network query processing in sensor networks," *ACM SIGMOD Record*, vol. 31, no. 3, pp. 9–18, 2002.
[67] A. L. Murphy, G. P. Picco, and G.-C. Roman, "LIME: a middleware for physical and logical mobility," in *Proc. 21st International Conference on Distributed Computing Systems (ICDCS '01)*, pp. 524–533, Phoenix, Ariz, USA, April 2001.
[68] S. S. Yau and F. Karim, "An energy-efficient object discovery protocol for context-sensitive middleware for ubiquitous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 14, no. 11, pp. 1074–1085, 2003.
[69] D. Chakraborty, A. Joshi, Y. Yesha, and T. Finin, "GSD: a novel group-based service discovery protocol for MANETS," in *Proc. 4th International Workshop on Mobile and Wireless Communications Network (MWCN '02)*, pp. 140–144, Stockholm, Sweden, September 2002.
[70] N. Reijers and K. Langendoen, "Efficient code distribution in wireless sensor networks," in *Proc. 2nd ACM International Workshop on Wireless Sensor Networks and Applications*, pp. 60–67, San Diego, Calif, USA, September 2003.
[71] C. Jaikaeo, C. Srisathapornphat, and C.-C. Shen, "Querying and tasking in sensor networks," in *Proc. 14th International Symposium on Aerospace/Defense Sensing, Simulation, and Control*, vol. 4037 of *Proc. SPIE's*, pp. 184–197, Orlando, Fla, USA, April 2000.
[72] P. Levis, N. Lee, M. Welsh, and D. Culler, "TOSSIM: accurate and scalable simulation of entire TinyOS applications," in *Proc. 1st ACM Conference on Embedded Networked Sensor Systems (SenSys '03)*, pp. 126–137, Los Angeles, Calif, USA, November 2003.
[73] S. Park, A. Savvides, and M. B. Srivastava, "Simulating networks of wireless sensors," in *Proc. Winter Simulation Conference (WSC '01)*, pp. 1330–1338, Arlington, Va, USA, December 2001.
[74] X. Zeng, R. Bagrodia, and M. Gerla, "GloMoSim: a library for parallel simulation of large-scale wireless networks," in *Proc. 12th Workshop on Parallel and Distributed Simulations (PADS '98)*, pp. 154–161, Banff, Alberta, Canada, May 1998.
[75] Y. Yu and V. K. Prasanna, "Energy-balanced task allocation for collaborative processing in networked embedded systems," in *Proc. ACM SIGPLAN Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES '03)*, pp. 265–274, San Diego, Calif, USA, June 2003.

Mauri Kuorilehto received the M.S. degree in 2001 from Tampere University of Technology (TUT), Finland. He is currently pursuing his Ph.D. degree and acting as a Research Scientist in the DACI Research Group, the Institute of Digital and Computer Systems at TUT. His research interests include wireless sensor and ad hoc networks concentrating on distributed processing, operating systems, and network simulation.

Marko Hännikäinen received the M.S. degree in 1998 and the Ph.D. degree in 2002 both from Tampere University of Technology (TUT). Currently he acts as a Senior Research Scientist in the Institute of Digital and Computer Systems at TUT, and a Project Manager in the DACI Research Group. His research interests include wireless local and personal area networking, wireless sensor and ad hoc networks, and novel web services.

Timo D. Hämäläinen received the M.S. degree in 1993 and the Ph.D. degree in 1997 both from Tampere University of Technology (TUT). He acted as a Senior Research Scientist and Project Manager at TUT during 1997–2001. He was nominated to be Full Professor at TUT, Institute of Digital and Computer Systems in 2001. He heads the DACI Research Group that focuses on three main lines: wireless local area networking and wireless sensor networks, high-performance DSP/HW-based video encoding, and interconnection networks with design flow tools for heterogeneous SoC platforms.

PUBLICATION 3

M. Kuorilehto, T. Alho, M. Hännikäinen, T. D. Hämäläinen, "SensorOS: a New Operating System for Time Critical WSN Applications," in *Embedded Computer Systems: Architectures, Modeling, and Simulation*, Lecture Notes in Computer Science, vol. 4599, S. Vassiliadis, M. Bereković, T. D. Hämäläinen (Eds.), Springer-Verlag, Heidelberg, Germany, 2007, pp. 431–442.

SensorOS: A New Operating System for Time Critical WSN Applications

Mauri Kuorilehto[1], Timo Alho [2], Marko Hännikäinen[1], and Timo D. Hämäläinen[1]

[1] Tampere University of Technology, Institute of Digital and Computer Systems
P.O. Box 553, FI-33101 Tampere, Finland
{mauri.kuorilehto, marko.hannikainen, timo.d.hamalainen}@tut.fi
[2] Nokia Technology Platforms, Tampere, Finland
timo.a.alho@nokia.com

Abstract. This paper presents design and implementation of a multi-threading Operating System (OS), SensorOS, for resource constrained Wireless Sensor Network (WSN) nodes. Compared to event-handler kernels, such as TinyOS, SensorOS enables coexistence of multiple time critical application tasks. SensorOS supports preemptive priority-based scheduling, very fine-granularity timing, and message passing inter-process communication. SensorOS has been implemented for resource constrained Tampere University of Technology WSN (TUTWSN) nodes. In TUTWSN node platform with 2 MIPS PIC micro-controller unit, SensorOS kernel uses 6964 B code and 115 B data memory. The context swap time is 92 μs and the variance of timing accuracy for a high priority thread less than 5 μs. The results show that the realtime coordination of WSN applications and protocols can be managed by a versatile OS even on resource constrained nodes.

1 Introduction

Wireless Sensor Networks (WSN) consists of a large number of randomly deployed nodes that self-organize and operate autonomously. A WSN node is characterized by restricted resources in terms of memory, energy, and processing capacity, and by unreliable wireless link with limited bandwidth. While advances in manufacturing technologies have resulted in smaller and cheaper platforms suitable for WSN realizations, the resource constraints persist as the environments become more demanding. Simultaneously, the complexity and number of tasks of WSN applications increases [1].

The key functionalities for the layered WSN protocol stack are the controlling of channel access, network topology creation and maintenance, and route formation. The protocols together with multiple applications comprise an extremely complex system that must be fitted to resource constrained WSN nodes. Further, due to the tight interaction with the real world, realtime requirements are strict. Therefore, realtime communication and coordination are required in both single node and network level [2]. At a single node level, resource usage, timeliness, and peripheral access are managed by an Operating System (OS) [3]. The network level control in WSNs is handled by middleware architectures that perform task allocation and network control [2,4]

This paper presents the design and implementation of SensorOS, a preemptive multi-threading kernel for resources constrained TUTWSN (Tampere University of Technology WSN) nodes [5]. The time sliced Medium Access Control (MAC) protocol of

S. Vassiliadis et al. (Eds.): SAMOS 2007, LNCS 4599, pp. 431–442, 2007.

TUTWSN requires timing accuracy and efficient use of power saving modes. SensorOS guarantees timing with a priority-based realtime scheduler. The evaluation proves SensorOS suitability for WSNs, and shows the feasibility of the simple POSIX-like Application Programming Interface (API). A network level coordination can be incorporated into SensorOS by a distributing middleware for task allocation [6].

1.1 Related Work

Embedded Realtime OSs (RTOS), such as OSE, QNX Neutrino, and VxWorks are widely used in industrial and telecommunication systems. However, their memory consumption even in the smallest configurations is too large for resource constrained WSN nodes. Small memory footprint RTOSs, such as FreeRTOS, have a general purpose approach and do not meet the strict timing and energy saving requirements of WSNs.

The most widely known OS for WSNs is TinyOS [7] that uses a component-based event-driven approach for task scheduling. Each component has a separate command handler for upper layer requests and an event handler for lower layer events. The processing is done in an atomic task. SOS [8] adopts the component model from TinyOS but allows dynamic runtime loading and unloading of components. Similar approach without relation to TinyOS is taken in BerthaOS [9]. Event handler kernel of Contiki [10] supports dynamic loading and can be complemented with a library for preemptive multi-threading. In CORMOS [11], all system and application modules consist of handlers that communicate seamlessly with local and remote modules using events.

Preemptive multi-threading for sensor nodes with POSIX API is implemented in MOS [12] and nano-RK [13]. Both support priority-based scheduling and have integrated networking stack and power management features.

Due to run to completion semantics, event handler OSs, such as TinyOS, are poorly suitable for applications with lengthy computation, e.g. cryptographical algorithms. Further, compared to traditional preemptive kernels, their programming paradigm can be difficult to understand. The drawback of preemptive schedulers is the increased data memory consumption as a separate stack is needed for each thread. While Contiki partially solves this, it faces the problems of event-driven OS if multi-threading is not used.

The approach in SensorOS is similar to MOS and nano-RK. Features that put SensorOS apart from these two are very accurate time modeling and energy efficiency. The energy efficiency results from the sophisticated use of advanced power saving modes.

1.2 Contents of the Paper

The architecture and design of SensorOS are discussed in Section 2. Section 3 presents TUTWSN platform and environment. The implementation of SensorOS on target platform is presented in Section 4 and evaluation results in Section 5. Finally, conclusions are given in Section 6.

2 SensorOS Design

SensorOS design objective is a realtime kernel that supports features required by WSN protocols and applications. WSN protocol and application tasks are executed as separate

threads communicating with SensorOS Inter-Process Communication (IPC) methods. The composition of threads implementing protocols and applications is not restricted.

In this paper, a *task* is a functional entity, whereas a *thread* is the OS context, in which a task is executed. A task can be divided into multiple threads, but on the other hand several tasks can be executed within a single thread.

2.1 Design Requirements

A WSN protocol stack consists of several functional entities that require cross layer interaction for controlling network operation. Typically, the energy efficiency of a WSN results from the accuracy of MAC protocol timing. Accurate timing allows longer sleep periods since the wake-up can be done just before the active period. In addition, a tight relation to the real world requires reactiveness from applications. As a result, the programming of complex protocols and applications, and the managing of their communication and synchronization are extremely challenging and tedious without OS control.

The requirements for SensorOS derive from the characteristics of WSNs. The main functional requirements are seamless coexistence of multiple tasks, realtime capability, and timing accuracy. Due to limited WSN node capabilities, efficient usage of resources is essential. Portability is required to deal with heterogeneous WSN node platforms. Memory management is needed to allow as many tasks as possible to be located in a node and power management to maximize the lifetime of battery-powered nodes.

More abstract requirements for SensorOS relate to the ease of use and the integration of a distributing middleware. A simple API facilitates application development. The middleware integration is alleviated by using a message passing IPC that can be easily abstracted to network packets.

2.2 SensorOS Architecture

The architecture of SensorOS is divided into components as depicted in Fig. 1. Tasks access OS services through API. The main components in the kernel are *scheduler*, message passing *IPC*, *timer*, *synchronization*, *memory* and *power management*. Interrupt-driven device drivers (UART and Analog-to-Digital Converter (ADC) in Fig. 1) are integrated into the kernel, whereas context-related drivers (I^2C, radio) are executed in the context of a calling thread without a relation to the OS kernel. In general, devices accessed by a single thread are context-related, while shared devices are included in the kernel. Hardware resources are accessed through a Hardware Abstraction Layer (HAL).

Each thread in SensorOS has a Thread Control Block (TCB) for per threadL information. A thread can be in three different states. When a thread is executed on MCU it is *running*. The state of a thread is *ready* when it is ready for execution but another thread is running, and *wait* when it needs an event to occur before execution.

A thread can be waiting for multiple different type of events in SensorOS. The relation between a running thread, a ready queue, and different wait queues are depicted in Fig. 2. When a thread is created it is put to the ready queue, and it can explicitly exit when running. Threads waiting for a timeout are in timer queues and those waiting for IPC in message set. Synchronization is waited in per mutex queues and a completion of peripheral operation in a peripheral specific item.

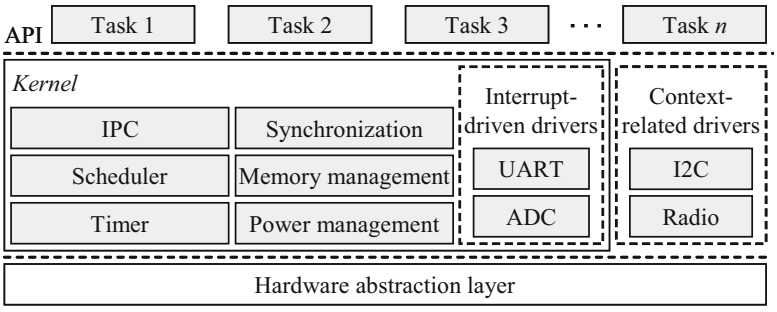


Fig. 1. Overview of SensorOS architecture

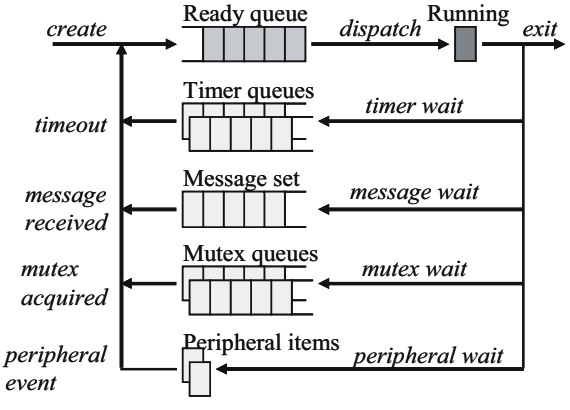


Fig. 2. Thread queues and events moving a thread from a queue to another

2.3 SensorOS Components

SensorOS components maintain TCBs of threads accessing their services. The interrelations between components are kept in minimum, but clearly scheduler is dependent on other components.

***Scheduler* –** SensorOS incorporates a priority-based preemptive scheduling algorithm. Thus, the highest priority thread ready for execution is always running. Threads at the same priority level are scheduled by a round robin algorithm without a support for time-slicing.

When an event changes a thread to the ready state, the scheduler checks whether it should be dispatched. If it has a higher priority than the running thread, contexts are switched. When the running thread enters to a wait state, the highest priority thread from the ready queue is dispatched. If the ready queue is empty, power saving is activated.

***Timer* –** Timer component implements timeout functionality. The local time in SensorOS is microseconds since system reset. Timing is divided into two separate approaches that have their own timer queues. A fine granularity timing provides microsecond accuracy for applications and communication protocols that need exact

timestamps. The coarse timing is for tasks that tolerate timeout variations in order of millisecond.

IPC – The method for communication between tasks in SensorOS is message-passing IPC. A thread allocates a message envelope and fills it, after which it is sent to the recipient. The message must always be assigned to a certain thread. Broadcast messages can be implemented using multiple unicast messages.

Synchronization – Synchronization controls the flow of execution between tasks and access to peripheral devices and other hardware resources. The synchronization is implemented with binary mutexes. A mutex can be waited by several threads, of which the highest priority thread acquires it when released. Each mutex has its own wait queue. Avoiding of priority inversion is not considered but it is left to programmers [3].

Memory Management – In SensorOS, dynamic memory management is incorporated for message envelopes and for temporary buffers occasionally needed by tasks. A thread allocates and frees previously reserved blocks from a memory pool.

Power Management – Since the activity of WSN nodes is in order of few per cents, power management is crucial. In SensorOS, the power management of peripherals is implemented in the device drivers. The power saving activation of context-related devices is left to the task that controls the device, because the task is aware of the device activation patterns. Instead, the power modes of MCU and integrated peripherals are managed by OS. When there are no threads to schedule, MCU is set to platform dependent power saving mode, of which it is woken up by an external event.

Peripherals – The interrupt-driven device drivers integrate peripherals, such as ADC and UART, tightly to SensorOS kernel. They have separate functions for open, close, control, read, and write operations. The read and write operations are controlled by interrupts. A thread can block its execution on such peripheral until the specified number of bytes has been transferred. The context-related device drivers are either non-blocking or can use an external interrupt source for controlling read and write operations.

3 TUTWSN Platforms and Protocols

SensorOS is primarily targeted to TUTWSN node platforms and protocols. TUTWSN is an energy efficient WSN framework targeted mainly for monitoring applications. TUTWSN incorporates several different types of node platforms, a configurable protocol stack, and user interfaces for network monitoring and application visualization.

3.1 TUTWSN Node Platform

Several different types of node platforms are used in TUTWSN. An outdoor temperature sensing node platform illustrated in Fig. 3 is built from off-the-shelf components. The main component is PIC18LF4620 MCU, which contains a 10-bit integrated ADC and 1 KB of EEPROM as a non-volatile data storage. The power unit consists of a MAX1725 regulator with 2.4 V output voltage and a 3 V CR123A lithium battery. In addition, a DS620 digital thermometer is integrated to the platform. The radio interface

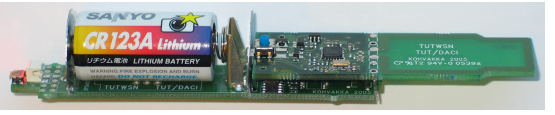


Fig. 3. TUTWSN PIC node platform

Table 1. TUTWSN PIC node power consumption in different states

MCU	Radio	Power (mW)
active	receive	60.39
active	transmit (0 dBm)	39.90
active	transmit (-20 dBm)	26.73
active	active	3.68
active	off	3.29
idle	off	1.27
sleep	off	0.031

on the platform is a 2.4 GHz nRF2401A transceiver unit that supports 1 Mbit/s data rate and transmit power between -20. . .0 dBm.

MCU has a 64 KB Flash as code memory, each instruction word taking two bytes. Internal SRAM data memory is limited to 3986 B. With internal oscillator the MCU frequency can be either 4 MHz or 8 MHz resulting in 1 MIPS and 2 MIPS, respectively. For power saving, PIC supports idle and sleep modes. The measured power consumptions of TUTWSN node platform in PIC MCU power modes with 4 MHz frequency and different radio activation states are depicted in Table 1. The power consumptions of other, application dependent, peripherals are typically in order of hundreds of μWs. The radio power consumption on receive and transmit is dominant.

3.2 TUTWSN Protocols

The main protocols in TUTWSN stack are Time Division Multiple Access (TDMA) MAC and gradient-based routing. The MAC protocol creates a clustered network topology and controls wireless channel access. The coordination between clusters is done on a dedicated signaling channel, while each cluster operates on its own frequency channel. The routing protocol creates routes from cluster *headnodes* to a *sink* based on the cost gradient of the route.

The cluster headnode maintains its access cycle by periodic beacons. Neighbor headnodes and *subnodes* associate to the cluster for data communication. The objective of TDMA MAC is to minimize power-hungry radio idle listening, which requires accurate time synchronization among nodes.

4 SensorOS Implementation

SensorOS is implemented on TUTWSN PIC nodes. The implementation follows the architecture presented in Section 2. Common functionality is implemented separately,

whereas hardware dependent parts are included in HAL in order to facilitate portability. The common functionalities and most of HAL are implemented in ANSI C. Only a small portion of the lowest level HAL, e.g. context switch, is implemented in assembly.

4.1 Implementation of Hardware Abstraction Layer

Lowest level context switching, power saving, timer, and peripheral access are detached from SensorOS kernel to the HAL implementation. Internal registers that need to be saved at context switch are MCU dependent. Also power saving modes need low level register access. Each peripheral has a HAL component that implements interface to dedicated I/O ports and interrupt handlers.

Each MCU has an own set of hardware timers and their control registers. HAL timer implementation consists of time concept, interrupt handlers, and time management routines. SensorOS utilizes two different time concepts implemented by HAL; a microsecond resolution timer for accurate timing and a millisecond resolution timer for timeouts. The interrupt handlers update internal time and when a time limit expires indicate this to the OS timer through a callback function. The time management routines are for getting and manipulating internal time, setting of timeout triggers, and atomic spinwait for meeting an exact timeline.

4.2 Implementation of SensorOS Components

SensorOS API consists of system calls listed in Table 2. Peripheral system calls are for character devices (e.g. UART) while context-related devices have dedicated interfaces. SensorOS is initialized in *main* -function, which issues *user_main* -function after OS components have been initialized. In user_main, threads for application tasks and required mutexes are initialized. After the user_main returns, scheduling is started.

Scheduler – A thread is created with **os_thread_create** that takes the stack and Process IDentifier (PID) as parameters. This simplifies the implementation but prevents runtime creation and deletion of threads. The modification of the kernel for such a support is straightforward. When a thread is created it is inserted to the ready queue but the scheduler is not invoked until the running thread releases processor.

Instead of a completely modular approach, the scheduling decisions are distributed to kernel components. This complicates the changing of scheduling algorithm but improves context switching performance. When an event moves thread(s) to the ready queue, the OS component checks whether one of the threads has a higher priority than the running one. If true, an OS service for swapping threads' contexts is invoked. The context of a thread is stored in its stack. A running thread can release processor with **os_yield** or it can permanently exit. When there are no threads to schedule, an *idle thread* is scheduled for activating MCU sleep mode through HAL.

Event waiting in SensorOS is implemented by a single interface that allows a thread to wait simultaneously for multiple events. The events include timeout, message received, character device read and write, peripheral device, and user generated events. Function **os_poll_event** loops actively while **os_wait_event** blocks the thread until any of the events occur. When an event for a thread is raised, the scheduler checks whether the thread waits for the event and if it does performs scheduling.

Timer – Timer operation is mainly implemented in HAL but API and scheduling on timeouts are provided by the OS component. The system time is obtained with the function **os_get_time**. Accurate timestamps for events are set with **os_get_entryperiod**, which returns the internal time at the moment of the function call. Both utilize microsecond resolution timer.

The accurate microsecond resolution wait is implemented by **os_wait_until**. The thread is blocked until a threshold before the deadline. The atomic spinwait in HAL is used to suspend the operation until the timestamp. In the current implementation, only one thread can issue **os_wait_until** at a time to guarantee accurate timing.

Table 2. SensorOS system call interface, categorized by components

Thread and scheduler management system calls
void **os_thread_create**(os_proc_t *p, os_pid_t pid, os_priority_t pri, char *stack, size_t stackSize, prog_counter_t entry)
void **os_yield**(void)
os_eventmask_t **os_wait_event**(os_eventmask_t events)
os_eventmask_t **os_poll_event**(os_eventmask_t events)
Timer system calls
uint32_t **os_get_time**(void)
os_uperiod_t **os_get_entryperiod**(void)
int8_t **os_wait_until**(os_uperiod_t event)
void **os_set_alarm**(uint16_t timeout)
IPC system calls
os_status_t **os_msg_send**(os_pid_t receiver, os_ipc_msg_t *msg)
os_ipc_msg_t* **os_msg_recv**(void)
int8_t **os_msg_check**(void)
Synchronization system calls
void **os_mutex_init**(os_mutex_t *m)
void **os_mutex_acquire**(os_mutex_t *m)
void **os_mutex_release**(os_mutex_t *m)
Memory management system calls
void* **os_mem_alloc**(size_t nbytes)
void **os_mem_free**(void *ptr)
Character device system calls
os_status_t **os_open**(os_cdev_t dev)
void **os_close**(os_cdev_t dev)
int8_t **os_write**(os_cdev_t dev, const char *buf, uint8_t count)
int8_t **os_read**(os_cdev_t dev, char *buf, uint8_t count)
void **os_close**(os_cdev_t dev)

To initialize a millisecond resolution wait, a thread issues **os_set_alarm**. The thread is put to the timer queue that is sorted according to the timeouts. The first item in the queue is passed to HAL in order to trigger a callback function when the timeout expires. The callback function sets the timer event for the first thread in the queue. A zero timeout period can be used with **os_wait_event** to check a status of other events.

***IPC* –** The memory allocation for message envelopes and the contents of messages are left to the application. A message is sent with **os_msg_send** that inserts the message to the recipient's queue and sets the message received event. Each thread has an own message queue in its TCB. A thread can check whether its queue is empty with **os_msg_check**. A message is removed from the queue by calling **os_msg_recv**.

***Synchronization* –** When a mutex is created with **os_mutex_init**, its wait queue and owner are cleared. If the mutex is blocked by another thread when **os_mutex_acquire** is issued, the calling thread is inserted to the wait queue of the mutex. Otherwise the caller becomes the owner of the mutex. When the owner calls **os_mutex_release** and the wait queue is not empty, the highest priority thread is moved to the ready queue, or scheduled immediately if its priority is higher than that of the running thread.

***Memory Management* –** Currently, there are two alternatives for memory management. A *binary buddy* algorithm allows the allocation of different sized blocks, while a more lightweight alternative uses static sized blocks and is mainly targeted to message envelopes. Memory is allocated with **os_mem_alloc** and released with **os_mem_free**.

***Peripherals* –** The interrupt-driven character device drivers are opened and closed by **os_open** and **os_close**, respectively. The device handle contains the owner, type, and event information and defines the HAL routines and data pipe for communication between HAL and OS. Data to the device is sent with **os_write** and received with **os_read**. Both return the number of bytes handled. The completion of a pending operation can be waited either by **os_flush** or **os_wait_event**.

5 Evaluation

The objectives of SensorOS evaluation are the verification of correct functionality and the measuring of OS resource consumption and performance. A test application, which consisting of three tasks and emulates WSN protocol stack and an application, is implemented for evaluation. *Task1* models TDMA-based WSN MAC protocol, *task2* a routing protocol, and *task3* an WSN application with periodic sensing and processing.

The highest priority thread (*task1*) is activated periodically with a hard deadline. It executes for a short period and sends a message to the next highest priority thread (*task2*) on every tenth activation. *Task2* waits for message and processes it when received. Then it sends a message to the lowest priority thread (*task3*). *Task3* is activated periodically and if it has a message it performs lengthy processing.

5.1 Resource Usage

The portable implementation in ANSI C results slightly more inefficient use of resources than an assembly optimized one. The code and static data memory consumption

of each OS component are depicted in Table 3. Help routines include implementations for internal OS lists and a small set of library functions.

The code memory usage of SensorOS with static block memory management is 6964 B and with binary buddy 7724 B, which are 10.6 % and 11.8 % of the available memory, respectively. These do not include an optional I/O library that implements *printf* type routines. Static data memory used by SensorOS is 115 B or 118 B, depending on the used memory management. These do not include thread stacks and TCBs. A thread context takes 36 B on average but in interrupts additional 35 B is stored. Since the context is kept in the thread's stack, a typical stack size is 128 B. The size of TCB is 17 B. Thus, over 20 threads can be active simultaneously in TUTWSN PIC platform.

5.2 Context Switch Performance

The performance of SensorOS is evaluated by measuring the context switch overhead and the executions times of main kernel operations. These are given in Table 4 with timing accuracy results. MCU is run at 8 MHz and loaded by five threads that have averagely 2 ms activation interval. The results are gathered over 50000 iterations.

Context swap time includes the storing of an old and restoring of a new thread to MCU. The initialization of **os_wait_until** sets a trigger to HAL. The thread is woken up 2 ms before the deadline and after a scheduling delay the rest of the time is spent in spinwait. The time in **os_set_alarm** is consumed in timer queue handling and a trigger setting. The **os_wait_event** time is the delay from a timer interrupt to the scheduling of the thread. The IPC delay is measured from the sending of a message from a lower priority thread to its processing in a higher priority one.

The **os_wait_until** is evaluated by measured the absolute error between the resulted timing and the real world time. The maximum inaccuracy is below 5 μs and typically the error is less than 2 μs. The variance is caused by thread atomicity consideration when returning from the spinwait, thus it is affected by MCU clock frequency.

Table 3. Code and data memory usage of different SensorOS components

OS component	Code memory (B)	Data memory (B)
Scheduler	728	38
Thread	184	0
Event handling	384	1
Timer	646	6
IPC	248	0
Mutex	428	0
Binary buddy memory management	1048	5
Static block memory management	288	2
Character device	414	0
HAL	2266	68
Help routines	1378	0
I/O library	862	16

Table 4. SensorOS kernel operation times and timing accuracy

Operation	Time (μs)		
	Min	Mean	Max
HAL context swap	92	92	92
os_wait_until timeout initialization	125	125	125
os_wait_until spinwait time after thread wakeup [1)]	680	1110	1310
os_set_alarm timeout initialization [1)]	222	270	324
os_wait_event context switch from timer interrupt [1)]	486	532	558
IPC from lower priority thread to higher one	346	346	346
os_wait_until timing absolute error	0.0	1.8	4.2

1) The results may vary slightly depending on the number of threads.

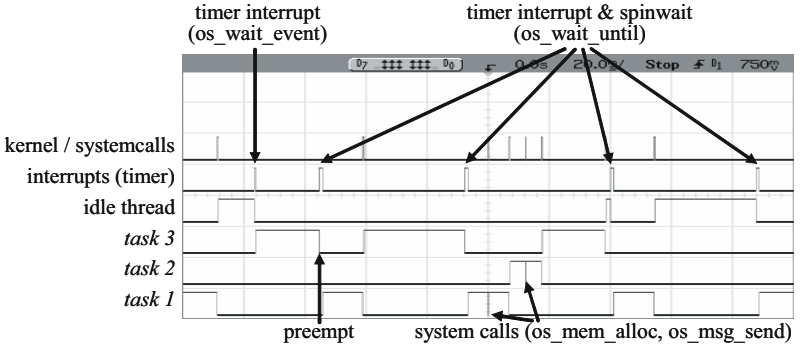


Fig. 4. Task and kernel activation in SensorOS

5.3 Test Application Operation

The scheduling of tasks and kernel components in the test application is depicted in Fig. 4. MCU preemption on periodic scheduling of *task1* is clearly visible. Kernel is activated when system calls are done for timer wait, messaging, and memory allocation. The idle thread is scheduled to activate power saving when other tasks are inactive.

The lengths of timer interrupt periods show the difference between **os_wait_until** and **os_wait_event** triggered by **os_set_alarm**. As the latter can return immediately after the timeout interrupt, the delay is considerably shorter than in **os_wait_until**.

6 Conclusions and Future Work

This paper presents a full functionality OS for resource constrained WSN nodes. Compared to existing WSN OSs, SensorOS implements more accurate time concept and sophisticated power management routines, which are needed by energy efficient and time critical WSN protocols and applications. The portability and conventional API facilitate the implementation of large WSN scenarios with multiple applications. The evaluation shows that SensorOS obtains excellent performance with minimal resources.

Our future work concentrates on implementation and integration of the distributing middleware to OS. Further, we are exploring methods for lightweight dynamic linking of new application threads transferred over wireless link.

References

1. Akyildiz, I.F., Su, W., Sankarasubramaniam, Y., Cayirci, E.: A survey on sensor networks. IEEE Communications Magazine 40(8), 102–114 (2002)
2. Stankovic, J.A., Abdelzaher, T.F., Lu, C., et al.: Real-time communication and coordination in embedded sensor networks. Proceedings of the IEEE **91**(7) (2003) 1002–1022
3. Stallings, W.: Operating systems internals and design principles, 5th edn. Prentice-Hall, Englewood Cliffs (2005)
4. Kuorilehto, M., Hännikäinen, M., Hämäläinen, T.D.: A survey of application distribution in wireless sensor networks. EURASIP Journal on Wireless Communications and Networking (5), 774–788, Special Issue on Ad Hoc Networks: Cross-Layer Issues (2005)
5. Kohvakka, M., Hännikäinen, M., Hämäläinen, T.D.: Ultra low energy wireless temperature sensor network implementation. In: Proc. 16th Annual IEEE International Symposium on Personal Indoor and Mobile Radio Communications, Berlin, Germany, pp. 801–805. IEEE Computer Society Press, Los Alamitos (2005)
6. Kuorilehto, M., Hännikäinen, M., Hämäläinen, T.D.: A middleware for task allocation in wireless sensor networks. In: Proc. 16th Annual IEEE International Symposium on Personal Indoor and Mobile Radio Communications, Berlin, Germany, pp. 821–826. IEEE Computer Society Press, Los Alamitos (2005)
7. Hill, J., Szewczyk, R., Woo, A., et al.: System architecture directions for networked sensors. In: Proc. 9th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Cambridge, MA, USA, pp. 94–103 (2000)
8. Han, C.C., Kumar, R., Shea, R., et al.: A dynamic operating system for sensor nodes. In: Proc. 3rd International Conference on Mobile Systems, Applications, and Services, Seattle, WA, USA, pp. 163–176 (2005)
9. Lifton, J., Seetharam, D., Broxton, M., Paradiso, J.: Pushpin computing system overview: a platform for distributed, embedded, ubiquitous sensor networks. In: Proc. 1st International Conference on Pervasive Computing, Zurich, Switzerland, pp. 139–151 (2002)
10. Dunkels, A., Grönvall, B., Voigt, T.: Contiki - a lightweight and flexible operating system for tiny networked sensors. In: Proc. 29th Annual IEEE International Conference on Local Computer Networks, Tampa, FL, USA, pp. 455–462. IEEE Computer Society Press, Los Alamitos (2004)
11. Yannakopoulos, J., Bilas, A.: Cormos: a communication-oriented runtime system for sensor networks. In: Proc. 2nd European Workshop on Wireless Sensor Networks, Istanbul, Turkey, pp. 342–353 (2005)
12. Bhatti, S., Carlson, J., Dai, H.: Mantis os: An embedded multithreaded operating system for wireless micro sensor platforms. Mobile Networks and Applications 10(4), 563–579 (2005)
13. Eswaran, A., Rowe, A., Rajkumar, R.: Nano-rk: An energy-aware resource-centric rtos for sensor networks. In: 26th IEEE International Real-Time Systems Symposium, Miami, FL, pp. 256–265. IEEE Computer Society Press, Los Alamitos (2005)

PUBLICATION 4

M. Kuorilehto, M. Hännikäinen, T. D. Hämäläinen, "A Middleware for Task Allocation in Wireless Sensor Networks," in *Proceedings of the 16th Annual IEEE International Symposium on Personal Indoor and Mobile Communications (PIMRC 2005)*, Berlin, Germany, Sep. 11–14, 2005, pp. 821–826.

A Middleware for Task Allocation in Wireless Sensor Networks

Mauri Kuorilehto, Marko Hännikäinen, *Member, IEEE*, and Timo D. Hämäläinen, *Member, IEEE*

Abstract— **Resource constrained platforms, dynamic nature, and complex applications set challenges to the development of Wireless Sensor Networks (WSN). Sophisticated tasking and networking control is required in WSNs to reach lifetimes in order of years. This paper presents a WSN node middleware, which controls task allocation and WSN topology according to the current requirements of the application. The middleware uses a lightweight algorithm that balances communication and computation load between nodes. The discovering of resources and application tasks are comprised by a tuple space that selectively disperses information to nodes. The middleware has been implemented and evaluated in a WIreless SEnsor NEtwork Simulator (WISENES) that models resource usage and network operation accurately. The results show that in a static network configuration the obtained lifetime with our middleware is 6.8 times longer compared to an uncontrolled network while the increase in processing is negligible and the peek data memory usage increases by 11.6%. In a dynamically changing network the lifetime increases by a factor 3.9. Our middleware does not limit the applications and networks and improves the performance and predictability of WSNs significantly.**

Index Terms—**Middleware, Task allocation, Service discovery, Wireless Sensor Network (WSN)**

I. Introduction

WIRELESS Sensor Networks (WSN) consist of numerous resource constrained nodes. These nodes gather data, perform collaborative data processing, and forward data to a sink node for further processing. Tasks in sensor nodes are restricted by limited communication, processing, memory, and energy resources [1].

Potential WSN applications are diverse in numerous domains ranging from military to home appliances. Due to the diversity of applications, also their Quality of Service (QoS) requirements vary in terms of required dynamicity, scalability, resources, lifetime, connectivity, robustness, and realtime [2].

The design and implementation of a general purpose WSN is not feasible but WSN protocols and their configuration parameters must be tailored application-specifically. The most important WSN protocol is a Medium Access Control (MAC) protocol that maintains network topology and controls channel access. WSN topology is either clustered, where a set of *subnodes* is coordinated by a *headnode*, or flat, in which all nodes are equal. The multi-hop paths between nodes are created by a routing protocol on top of MAC.

A need for a common Application Programming Interface (API) that hides the heterogeneity of platforms and protocols has emerged in WSNs as application and platform complexity has increased. A middleware layer has been proposed for that purpose. In addition to pure functional API, middlewares for WSNs incorporate control for task allocation and network topology in order to improve network performance [3].

In this paper, we present a *WSN node middleware* that embodies a computationally lightweight algorithm for load balancing between heterogeneous nodes inside one cluster. The middleware improves WSN longevity and predictability, and allows the network to adapt its functions according to the changes in environmental conditions. The middleware does not restrict task functionality and relations. Thus, it is applicable to applications from simple monitoring to complex signal processing. The middleware is either stand-alone or cooperates with node Operating System (OS) such as *SensorOS*, which is being developed in Institute of Digital and Computer Systems at Tampere University of Technology (TUT).

A *task allocation* in the WSN node middleware lengthens WSN lifetime by an algorithm that adapts network topology and allocates sensing and data processing tasks evenly to nodes in a cluster. The flexibility for adaptation is achieved by a *tuple space* [4] that stores task parameters and application QoS information. In addition, our middleware consists of a *data redirecting* that implements API for application tasks, and of a *task hosting* that initiates optional binary transfers of task executables between nodes.

Our middleware has been developed in WIreless SEnsor NEtwork Simulator (WISENES) on top of TUTWSN protocol stack [5]. WISENES gives accurate information about node and network performance, which are analogous with the TUTWSN prototyping results. WSN node middleware simulation results show a considerable increase in WSN lifetime and predictability when the middleware is used.

This paper is organized as follows. Related research is discussed in Section II. The design of our middleware is presented in Section III. Section IV gives details on the middleware implementation to WISENES followed by simulation results in Section V. Finally, conclusions are given and future work projected in Section VI.

Manuscript received March 11, 2005.

M. Kuorilehto, M. Hännikäinen, and T. D. Hämäläinen are with Institute of Digital and Computer Systems, Tampere University of Technology, Korkeakoulunkatu 1, Tampere, FI-33710 Finland (email: {mauri.kuorilehto, marko.hannikainen, timo.d.hamalainen}@tut.fi).

978-3-8007-2909-8/05/$20.00 ©2005 IEEE

II. Related Work

The ongoing research on WSNs concentrates on MAC and routing algorithms, and node prototype implementations. WSN research on distributed processing is still quite limited.

From the existing middleware proposals, application QoS specification based task allocation and network topology adaptation are done in [6], [7], and [8]. These take adaptive actions within the network. However, the algorithms are communication and computation intensive and are restricted in scalability and applicability for heterogeneous platforms.

A centralized task allocation by a sink node is implemented in [10]. Static topology is assumed, which limits the applicability of the approach in dynamic WSNs. Further, efficient transceivers are required for communication link to the sink node. A design time approach is taken in [11], in which application task communication and dependencies are modified prior to deployment. During runtime, load is balanced by task migration on top of a Virtual Machine (VM). However, the approach is unable to adapt WSN operation according to the changes in environmental conditions. WSN abstraction as a database, e.g. [12], [13], allocates tasks in database queries but is not applicable to more complex tasks.

Task hosting transfers task executables between nodes. Object migration is implemented in several VM architectures [11], [14], [15]. Yet, VM resource usage is extensive for resource constrained WSNs. MANTIS [16] and PushPin [17] implement task binary code transfers. Pushpin uses a simple checksum for integrity checking but neither protects system against malicious code.

A tuple space proposed originally for computer networks in Linda [4] has been adapted also for WSN and mobile ad-hoc environments [14], [15], [18]. Tuples are collections of passive data values that are inserted to, removed from, or read from a shared tuple space.

III. Middleware Design

Unlike the related proposals, our middleware is applicable for resource constrained WSNs. The task allocation algorithm that balances differences in node energy consumption is computationally lightweight, but still leads to excellent results. The task allocation is supported by a tuple space that minimizes memory consumption among the nodes sharing the tuple space. Further, binary transfers are implemented to guarantee needed task population.

A. Objectives and Requirements

The main objective of the WSN node middleware is to maximize WSN lifetime. By the lifetime we mean the time elapsed until the first node runs out of energy. The load is balanced between nodes, so that the remaining lifetime of WSN can be estimated accurately. As communication in WSNs is generally more costly than computation [18], the middleware control messages are decreased even with the cost of increased computation.

Our middleware is designed for different types of WSNs, node platforms, and applications. The only pre-requisites to the node platforms are the ability to estimate remaining energy and available memory, and an option to write to their own instruction memory. The WSN topology and application nature are not restricted.

The middleware adapts its actions according to application QoS requirements. The QoS requirements consist of several *QoS level* specifications. These define for each level the task composition, priorities, and dependencies, data latency and accuracy, and the relations to other QoS levels. Thus, an application must be divided into tasks having a solitary function and QoS levels must be specified for the middleware.

A QoS level lists the tasks that are activated on the level. For each task, the redundancy within a virtual cluster is derived from the data accuracy specification. Needed timing and connectivity are defined by data latency and dependencies. The currently active QoS level depends on the rules, which specify limit trigger values for the available resources in the network, or for a quantity that is being monitored.

B. Middleware Network Topology

Our middleware divides nodes into *virtual clusters* that define the borders for node cooperation. In a clustered network topology, virtual clusters are directly mapped to MAC layer clusters as depicted in Fig. 1 (a). In a flat topology, proximate nodes construct virtual clusters as shown in Fig. 1 (b). The clustered MAC topology is more suitable for our middleware. First, the nodes are spatially proximate justifying data aggregation. Second, the communication infrastructure is set and therefore negotiations for link formation are avoided. Third, the number of hops is limited thus restricting the communication load.

Within a virtual cluster, a *controlling node* coordinates middleware layer actions, while other nodes, subnodes, act accordingly. The communication within the virtual cluster and between nearby virtual clusters is routed by controlling nodes. In a clustered network topology, the controlling node is the headnode of the cluster. In a flat topology the controlling node is elected by the members of the virtual cluster.

C. Middleware Architecture

The four main components in the WSN node middleware and their relations to each others and to other components in a node are depicted in Fig. 2. First, application tasks register themselves to the task hosting (1). After the initialization, the task allocation activates tasks in the node and defines their data dependencies (2). Next, a task sends data to a dependent task (3) through the data redirecting. The data redirecting

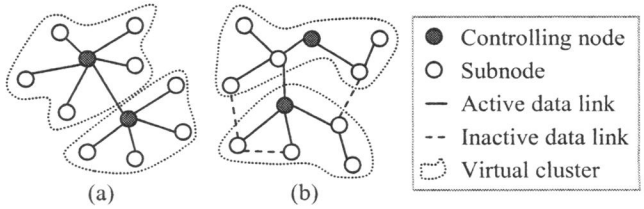


Fig. 1. Virtual cluster composing for (a) clustered and (b) flat MAC layer network topology.

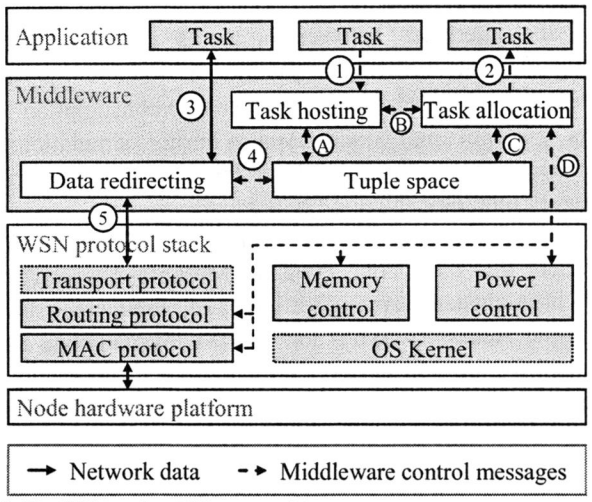


Fig. 2. The WSN node middleware architecture.

identifies a node that executes the dependent task using the tuple space (4). Finally, the data is either routed locally to a task or sent to network through WSN protocol stack.

Middleware components send *control messages* in order to relay information between components. All components pass messages to their counterparts in other nodes through the data redirecting (4). The task hosting stores task binary information to the tuple space (A). The task allocation uses hosting information for task activation decisions (B) and stores active task information to the tuple space (C). The task allocation changes network topology through MAC and routing protocols, and retrieves node resource information from memory and power control (D). A transport protocol and OS kernel are optional components that are not mandatory but can be applied if they offer added value.

D. Task Allocation

The task allocation has three main tasks that are task activation, network topology adaptation, and the controlling of application QoS levels. The task allocation in the controlling node makes decisions, while subnodes supply the controlling node with information. The subnodes send first a register message listing hosted tasks, and after that periodically an update message that defines active tasks and memory and energy resources in the subnode.

The application QoS level specification defines task activity and requirements. In addition to active tasks, the controlling node assignment is allocated periodically. As controlling node is under heavier computation and communication loading than subnodes its resources exhaust more rapidly.

The task allocation determines for each task a set of nodes, S_n, that host the task and have available memory and computation resources for the task activation. For each task, there is a set of nodes where the task is active, $S_a \subseteq S_n$, and a set of nodes where it is inactive, $S_i \subseteq S_n$.

Our task allocation algorithm evaluates all pairs, for which

$$\max_j(E_{S_i}) - \min_j(E_{S_a}) \geq d \quad (1)$$

is true, and changes the task allocation between them. The operators max_j and min_j return the jth maximum and minimum value of remaining energies E_{Sx} respectively. The number of iterations is defined by the task redundancy. The limit is

$$d = w\left(\frac{E_c}{2^k}\right), \quad (2)$$

where k is a coefficient, and E_c is the remaining energy in the controlling node. Factor w sets different limits for task and controlling node allocation. Generally, active tasks are allocated more frequently than controlling node assignment.

The task allocation defines also the currently active QoS level. A QoS level has at least one limit condition, after which another QoS level is activated. When the limit is met, the new level is activated and task allocation changed accordingly.

E. Task hosting

The task hosting keeps track of the task binaries in the nodes within a virtual cluster. If the required QoS cannot be met with the available tasks, the task hosting negotiates a binary transfer between two nodes. A node that hosts the task initiates a peer-to-peer transfer to a node with a similar Micro-Controller Unit (MCU). If such a node is not available, the active QoS requirements cannot be satisfied.

The task binary transfers expose WSN to several security vulnerabilities and consume resources excessively. A bit error may lead to a fatal error while a node is executing an erroneous task. A malicious party can also inject tasks to the network. Therefore, the binary transfers are done only when extremely necessary. We protect the nodes against bit errors using Cyclic Redundancy Check (CRC). When malicious attacks are considered as a risk, the CRC is replaced by digital signatures that verify the validity and origin of the task binary.

F. Tuple space

Task and QoS level information are stored in the tuple space. The tuple space stores internal middleware data used by the other components. A tuple information is shared within a virtual cluster. Nodes store only tuples that are valid in their current virtual cluster. A controlling node minimizes used memory by dispersing tuples only to subnodes that need the information. However, a required redundancy is maintained for recovering from error situations in nodes and from disconnected communication links.

The basic tuple space operations are *in* for inserting a tuple, *out* for removing the tuple, and *read* for getting the tuple data. We divide the tuple hierarchically into subtuples that can be modified by an *update* operation. Thus, a subtuple can be replaced without reinserting the complete tuple.

Fig. 3 (a) shows the task tuple structure. The tuple lists the nodes hosting and actively executing the task. The nodes, to which tuple is sent are listed in dispersed nodes. Task characteristics identify the task, its relations and requirements, and its resource usage provided by benchmarking data. A task tuple is dispersed to nodes executing tasks that have a data relation to the task. All tuples are also updated to the new controlling node when changed.

The QoS tuple is presented in Fig. 3 (b). The tuple defines the remaining energy limit, after which the level is no more activated. Active tasks list the priority, redundancy, data

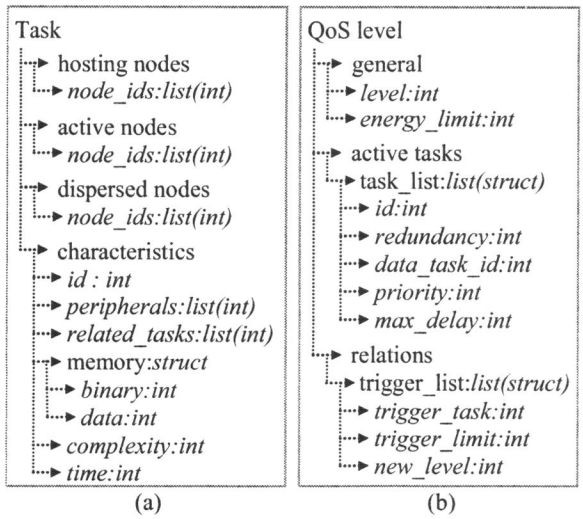


Fig. 3. Structures for (a) task and (b) QoS tuples.

relations, and maximum latency for each task. The relations to other QoS levels are defined by the list of trigger conditions. A trigger condition defines a limit value for a task output, e.g. an averaged temperature, after which a new level is activated. The QoS tuples are dispersed to the new controlling node.

G. Data redirecting

The data redirecting abstracts the nodes inside a virtual cluster to a unified set of resources. It offers API, through which application tasks communicate independent of their hosting nodes. The data redirecting also encapsulates the middleware control messages to network packets, and in the receiving end relays the control message to the correct middleware component.

An application task sends data to another task, not to a node. The data redirecting requests the node for that task from the tuple space. Used communication paradigm is message passing. Underlying OS features are utilized, if available but they can also be replaced by internal implementation.

IV. Implementation For WISENES

The WSN node middleware has been implemented in WISENES in order to test middleware performance and suitability for WSNs. The middleware is simulated on top of TUTWSN protocols that are developed at TUT [20].

A. Simulation Environment

The simulations are done in WISENES that models node resource constraints and consumption accurately, and models also environmental aspects. Prototyping is applicable to the testing of networks with tens of nodes but WISENES extends the evaluation possibility also to large network configurations.

The node platforms and their capabilities are specified to WISENES through an XML interface. From the simulations WISENES outputs information about timing, networking, and power, memory, and processing resource consumption. The output results are congruent to prototype measurements.

WSN protocols in WISENES are implemented in Specification and Description Language (SDL). The layers and the interfaces between protocols are defined by the designer. The WISENES framework models transmission medium, phenomena, and node resource consumption. It also logs protocol and node events during simulations. The framework has a Graphical User Interface (GUI) for runtime network topology and node state monitoring.

TUTWSN energy efficiency derives from a clustered MAC protocol that uses Time Division Multiple Access (TDMA) for intra-cluster communication and Frequency Division Multiple Access (FDMA) for interleaving neighbor clusters. Control for inter-cluster communication is done on a network signaling channel, whereas data are transferred during the access cycle of recipient cluster.

During an access cycle, a headnode sends network beacons to network signaling channel and manages data transfers in a super frame. The super frame consisting of cluster beacons, aloha slots for contention, and reservation data slots. Cluster beacons inform associated nodes of cluster timing and reservation data slots. Contention slots are for occasional data and association or slot reservation requests, and reservation data slots for periodical data transfers.

TUTWSN routing protocol broadcasts requests to neighbor nodes until a path to the destination is found. The neighbor cluster information from MAC protocol is utilized to decrease communication. Each node stores only the next hop node identifier and the total hop count to destinations.

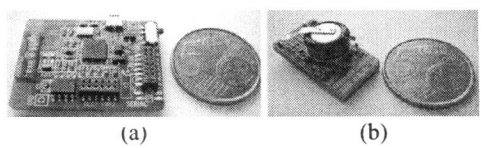
Several prototype platforms have been fabricated for TUTWSN. The main component in TUTWSN Xemics prototype, depicted in Fig. 4 (a), is a Xemics XE88LC02 2 MIPS MCU consisting of a CoolRisc 816 processor core, a 16-bit Analog-to-Digital Converter (ADC), 22 KB program memory, and 1 KB data memory. 2.4 GHz NordicVLSI nRF2401 transceiver unit on the prototype supports 250 kbps and 1 Mbps data rates. The transmit power is adjustable between -20 and 0 dBm.

A more resource constrained TUTWSN nRF prototype is depicted in Fig. 4 (b). 1 MIPS Intel 8051 MCU, 12-bit ADC, and nRF2401 are integrated into a single chip. The prototype has 4 KB of program and 256 B of data memory.

B. Middleware Implementation

The WSN node middleware is implemented on top of TUTWSN routing protocol. The middleware SDL block is depicted in Fig. 5. Each of the four main components is implemented as an SDL process. The SDL processes communicate with asynchronous signals that are routed through signal routes.

The data redirecting SDL process passes data as signals between application, middleware components, and TUTWSN protocol stack. It requests a destination node for the task data



(a) (b)

Fig. 4. TUTWSN (a) Xemics and (b) nRF prototypes.

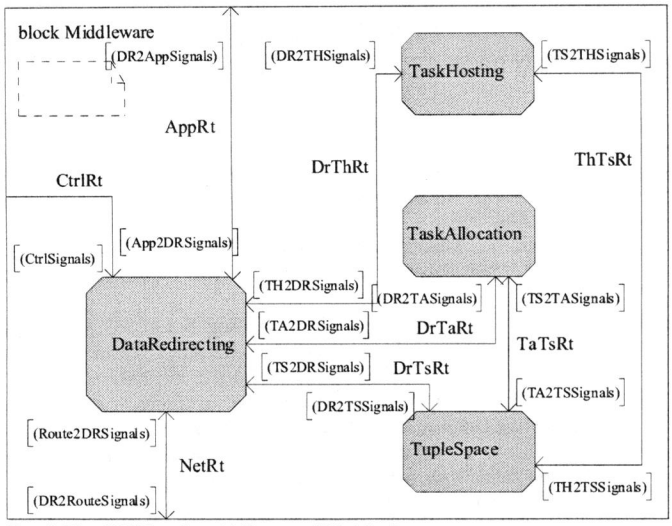


Fig. 5. The WSN node middleware SDL block.

from the tuple space SDL process by a signal.

The tuple space SDL process is implemented according to the design except the tuples are dispersed only to nodes that need the data. The redundancy for fault tolerance is not supported. Tuples are stored in SDL data structures that are interfaced through in, out, read, and update signals.

The task allocation SDL process implements task allocation algorithm and controls application QoS levels. The QoS levels are specified through the XML interface. The control messages are passed as SDL signals. The task hosting SDL process tracks hosted tasks but does not model binary transfers.

V. Simulation Results

The WSN node middleware is evaluated by two simulation cases. The first case tests the functionality and performance of the middleware with a static application and a homogeneous network. The second case evaluates middleware adaptability and performance in a dynamic network with heterogeneous node platforms. In both cases TUTWSN access cycle is ten seconds and the super frame consists of four contention and eight reservation data slots.

For the simulations, both modeled TUTWSN nodes have a 0.22 F capacitor as an energy source. TUTWSN Xemics prototype can operate approximately 1.5 hours as a headnode and 20 hours as a subnode with the capacitor. TUTWSN nRF prototype is able to act only as subnode for 12 hours.

The comparison of the WSN node middleware to the related proposals is not feasible, because the assumptions for the node resources, topology, and communication capacity differ considerably. Further, published results in the related work typically do not provide enough information for comparison, but mainly prove the functionality of the proposal.

A. Static Case

The network consists of ten TUTWSN Xemics prototypes that are within radio range of each others. The application has only two tasks and one QoS level. Temperature sensing task

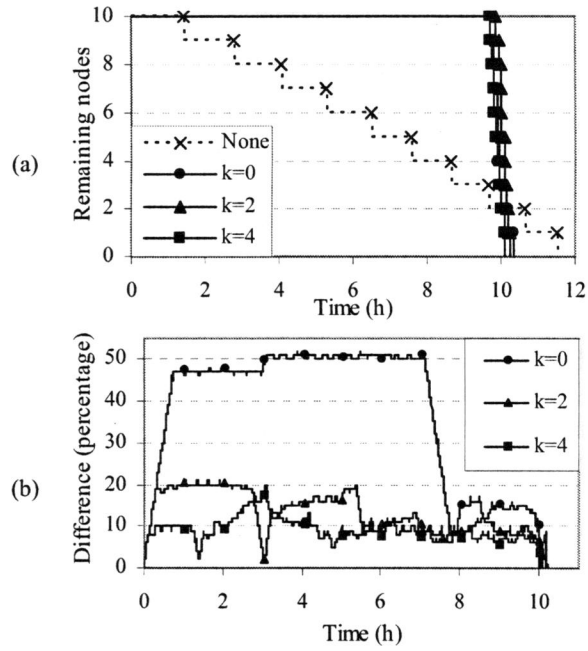


Fig. 6. Static case (a) node lifetimes and (b) peak difference in minimum and maximum remaining energy levels as a function of time.

requires 50% redundancy, and one second delay. The temperature aggregation task averages data.

The remaining nodes in the network as a function of time are depicted in Fig. 6 (a) when k is 0, 2, and 4. A case without task allocation is given as a reference. As shown, our middleware lengthens the network lifetime by a factor 6.85 compared to the reference case. Most importantly, the time between the first and the last node running out of energy is very short when WSN node middleware is used. The value of k does not have significant effect, as the load is balanced more accurately when the remaining energies approach zero.

The difference between minimum and maximum relative remaining energy levels at the nodes is depicted in Fig. 6 (b) with k varying similarly. The difference decreases as the network lifetime approaches, but for smaller values of k the difference is very large at the beginning. Thus, the network lifetime estimation at a random moment is accurate only with larger values of k.

The data memory consumption of a task tuple is 20-30 B depending on the length of the lists. A QoS tuple takes approximately 30 B. During the test case, at most 94 B of data memory is used in the controlling node. This is 11.6% of the overall data memory usage. The required processing for the middleware algorithms is done during MCU idle times while it is actively pending on the transceiver unit. Thus, MCU activation is not increased by the middleware.

B. Dynamic Case

15 TUTWSN Xemics and 15 TUTWSN nRF prototypes are randomly deployed for the second test case. Ten of the nRF prototypes are equipped with a temperature sensor, and five with a humidity sensor. TUTWSN Xemics prototypes include both sensors.

The application has three QoS levels listed in TABLE I. On initiation, the lowest QoS level is activated, and the level is

TABLE I
DYNAMIC CASE APPLICATION QOS LEVELS

QoS level	Sensing tasks (redundancy)	Aggregated data destination	Limit
Low	Temp. (20%)	None	20°C
Medium	Temp. (50%)	Temp. sink node	30°C
High	Temp. (50%) Hum. (20%)	Temp. sink node, Hum. sink node	-

changed when a limit condition is met. The simulated temperature is modeled so that it exceeds 20°C five times and 30°C once.

The change of a controlling node is more problematic when nodes are randomly deployed. Some of the subnodes may be out of range from the new controlling node. They must search a new cluster or form a new one, which consume energy. We use the option in TUTWSN prototypes to adjust transmit power to avoid the cluster scattering in most of the cases.

The network lifetimes until first node runs out of energy are depicted in TABLE II with and without transmit power adjusting. A network without task allocation is again given as a reference, and k is 0, 2, and 4. The lifetime with the WSN node middleware is smaller than in static case, because there are around five subnodes per a controlling node while in the static case the ratio was nine to one.

When a cluster scatters, subnodes consume their resources when they search for a new one. As a smaller value of k lengthens the interval between controlling node changes, resulted network lifetime is longer. The transmit power adaptation also diminishes the number of uncovered nodes, and therefore increases the lifetime, in average by 22%.

The peek data memory consumption of the controlling node in the test case was 192 B, for which the data memory of a TUTWSN Xemics prototype was sufficient.

VI. CONCLUSION

The presented middleware uses a computationally lightweight but still efficient task allocation algorithm to balance load in WSN. Our middleware is applicable for different types of WSNs and applications. It can be used also for network wide task allocation by forming hierarchical clusters, i.e. clusters of clusters.

The WSN node middleware activates tasks and allocates the controlling node assignment evenly to nodes within a virtual cluster. Further, the middleware adapts its actions according to an application QoS specification. The results show that the network lifetime and predictability in terms of remaining lifetime estimation are improved, while the increase in the consumption of other resources is negligible.

Our main future research objectives are on network wide task allocation using hierarchical virtual clusters. The middleware and a custom OS, SensorOS, will be implemented on TUTWSN nodes and tested on the prototypes.

TABLE II
DYNAMIC CASE NETWORK LIFETIMES IN HOURS FOR DIFFERENT VALUES OF K WITH AND WITHOUT TRANSMIT POWER ADAPTATION

Transmit power mode	No task allocation	WSN node middleware k=0	k=2	k=4
No adjust	1.41	4.61	4.02	3.88
Adjusted	1.38	5.44	5.02	4.82

REFERENCES

[1] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci, "A survey on sensor networks," *IEEE Communications Magazine*, vol. 40, issue 8, pp. 102-114, Aug. 2002.

[2] K. Römer, F. Mattern, "The Design Space of Wireless Sensor Networks", *IEEE Wireless Communications*, vol. 11, issue 6, pp. 54-61, Dec. 2004.

[3] J.A. Stankovic, T.F. Abdelzaher, L. Chengyang, S. Lui, J.C. Hou, "Real-time communication and coordination in embedded sensor networks," *Proceedings of the IEEE*, vol. 91, pp. 1002-1022, Jul. 2003.

[4] D. Gelernter, "Generative communication in Linda," *ACM Trans. Programming Languages and Systems*, vol. 7, pp. 80-112, Jan. 1985.

[5] M. Kuorilehto, M. Hännikäinen, T.D. Hämäläinen, "Rapid design and evaluation framework for wireless sensor networks," *Elsevier Ad Hoc Networks*, submitted for publication.

[6] W.B. Heinzelman, A.L. Murphy, H.S. Carvalho, M.A. Perillo, "Middleware to support sensor network applications," *IEEE Network*, vol. 18 issue 1, pp. 6-14, Jan./Feb. 2004.

[7] Y. Yang, B. Krishnamachari, V.K. Prasonna, "Issues in designing middleware for wireless sensor networks," *IEEE Network*, vol. 18 issue 1, pp. 15-21, Jan./Feb. 2004.

[8] P. Scerri, P.J. Modi, W.M. Shen, M. Tambe, "Applying constraint reasoning to real-world distributed task allocation," in *Proc. 1st ACM Int. joint Conf. on Autonomous Agents and Multiagent Systems*, Bologna, 2002.

[9] M. Kafil, I. Ahmad, "Optimal task assignment in heterogeneous distributed computing systems," *IEEE Concurrency*, vol. 6 issue 3, pp. 42-50, Jul.-Sep. 1998.

[10] M. Younis, K. Akkaya, A. Kunjithapatham, "Optimization of task allocation in a cluster-based sensor network," in *Proc 8th IEEE Int. Symp. on Computers and Communication*, Kemer, 2003, pp.329-334.

[11] H. Park and M.B. Srivastava, "Energy-efficient task assignment framework for wireless sensor networks," CENS Technical Report 26, Sep. 2003.

[12] C.C. Shen, C. Srisathapornphat, C. Jaikaeo, "Sensor information architecture and applications," *IEEE Personal Communications*, vol. 8 issue 4, pp. 52 -59, Aug. 2001.

[13] Y. Yao, J. Gehrke, "The Cougar approach to in-network query processing in sensor networks," *ACM SIGMOD Record*, vol. 31 issue 3, pp. 9 -18, Sep. 2002.

[14] M. Storey, G. Blair, A. Friday, "MARE: resource discovery and configuration in ad hoc networks," *Kluwer J. Mobile Networks and Applications*, vol. 7, pp. 377-387, Oct. 2002.

[15] A.L. Murphy, G.P. Picco, G.C. Roman, "LIME: a middleware for physical and logical mobility," in *Proc. 21st Int. Conf. on Distributed Computing Systems*, Phoenix, 2001, pp. 524-533.

[16] H. Abrach, S. Bhatti, J. Carlson, et al., "MANTIS: system support for multimodai networks of in-situ sensors," in *Proc. 2nd ACM Int. Workshop on Wireless Sensor Networks and Applications*, San Diego, 2003, pp. 50-59.

[17] J. Lifton, D. Seetharam, M. Broxton, J. Paradiso, "Pushpin computing system overview: a platform for distributed, embedded, ubiquitous sensor networks," in *Proc. 1st Int. Conf. on Pervasive Computing*, Zurich, 2002, pp. 139-151.

[18] J. Beutel, O. Kasten, F. Mattern, K. Roemer, F. Siegemund, L. Thiele, "Prototyping wireless sensor networks with btnodes," in *Proc. 1st European Workshop on Wireless Sensor Networks*, Berlin, 2004, pp. 323-338.

[19] G.J. Pottie, W.J. Kaiser, "Wireless integrated network sensors," *Communications of the ACM*, vol. 43, issue 5, pp. 51-58, May 2000.

[20] M. Kohvakka, M. Hännikäinen, T.D. Hämäläinen, "Wireless sensor network implementation for industrial linear position metering," *8th EUROMICRO Conf. on Digital System Design*, Porto, 2005, accepted for publication.

PUBLICATION 5

M. Kuorilehto, J. Suhonen, M. Kohvakka, M. Hännikäinen, T. D. Hämäläinen, "Experimenting TCP/IP Performance for Low-Power Wireless Sensor Networks," in *Proceedings of the 17th Annual IEEE International Symposium on Personal Indoor and Mobile Communications (PIMRC 2006)*, Helsinki, Finland, Sep. 11–14, 2006, 6 pages.

The 17th Annual IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC'06)

EXPERIMENTING TCP/IP FOR LOW-POWER WIRELESS SENSOR NETWORKS

Mauri Kuorilehto, Jukka Suhonen, Mikko Kohvakka, Marko Hännikäinen, and Timo D. Hämäläinen
Tampere University of Technology / Institute of Digital and Computer Systems
P.O.Box 553, FI-33101, Tampere, Finland
{mauri.kuorilehto, jukka.suhonen, mikko.kohvakka, marko.hannikainen, timo.d.hamalainen}@tut.fi

ABSTRACT

This paper presents the analysis and real experiments on TCP/IP communication in a low-power monitoring Wireless Sensor Network (WSN). TCP/IP flow control, addressing, and packet fragmentation are adapted in a gateway that relays TCP/IP communication to WSN. The performance of TCP/IP is evaluated between endpoint PCs communicating over TUTWSN (Tampere University of Technology WSN), which is used as a transparent communication medium for TCP/IP data. The evaluation results show that the window-based flow control algorithms of TCP perform too aggressively in WSNs, where random bit errors and topology changes are main reasons for errors instead of congestion. Further, a frequent duty cycle is needed in WSN to compensate the TCP/IP overhead. In TUTWSN, compared to an environmental monitoring application with two second activity cycle and native WSN transport, the enabling of TCP/IP consumes five times more power.

I. INTRODUCTION

Wireless Sensor Networks (WSN) consists of numerous resource constrained sensor nodes that are deployed to perform a specific task. These nodes self-configure and operate autonomously targeting to the network lifetimes in order of years. Due to the large scale and random deployment a node replacement is usually not an option. Thus, energy efficiency, robustness against environmental strains, and self-recovery are the key characteristics of WSNs. Applications for WSNs are diverse in home automation, industrial and environmental monitoring, military, and health care [1, 2].

Transmission Control Protocol (TCP) / Internet Protocol (IP) stack is the de facto standard in Internet, but it is generally not considered as the enabling technology for WSN communication. Considerable header overhead, connection management, and end-to-end flow and congestion control lead to poor performance in low-power WSNs characterized by resource constrained nodes, unreliable communication links, and frequently changing network topology [3, 4].

The integration of WSN to TCP/IP networks is still necessary for remote access (monitoring and control), and in general for enabling the ubiquitous computing [5]. Basically, there are three different integration approaches, depicted in Fig. 1. First, TCP/IP is the enabling communication technology (referred to as *direct tcp*). Second, a WSN gateway operates as a proxy that does address translation and flow control adaptation (*proxy tcp*). Last, WSN is a transparent medium that passes TCP/IP traffic on top of its infrastructure (*native tcp*).

In direct tcp, TCP/IP communication is routed to WSN, whereas in proxy and native tcp the communication is adapted at the gateway. Direct and proxy tcp support sockets and TCP/IP applications in nodes, while native tcp passes TCP communication as WSN application data. In proxy and native tcp, WSN protocols are used for communication. The most important protocols are Medium Access Control (MAC) and multi-hop routing. A MAC protocol controls wireless communication channel access and maintains network topology. A routing protocol creates multi-hop paths for the end-to-end communication between nodes.

The applicability of TCP/IP for wireless ad hoc networks has been widely studied and enhancements proposed [3, 6]. For WSNs, a proxy tcp approach is proposed in micro-IP [7] and nanoIP [8]. Both abstract WSN as an IP-subnet, which restricts the number of nodes to 254. Small memory footprint TCP/IP stacks with the most common features are presented in [9, 10]. These are applicable for resource constrained WSN nodes in terms of memory, but the problems due to communication conventions remain. A customizable protocol stack with TCP/IP support for different wireless environments is proposed in [11]. Even though promising, TCP/IP still needs considerable modifications for the full scale operation in low-power WSNs [12].

In this paper we analyze and evaluate the inter-operability of TCP/IP and a low-power monitoring WSN. We use a native tcp approach to adapt TCP/IP to a Time Division Multiple Access (TDMA) based low data rate WSN. The implementation and experiments are done with TUTWSN (Tampere University of Technology WSN) that consists of full feature protocol stack and node platforms [13, 14]. TUTWSN is used for real experiments but results are valid for TDMA-based WSNs.

This paper is organized as follows. Section II outlines WSN characteristics and compares the requirements and objectives of TCP/IP and WSNs. Section III presents the TUTWSN pro-

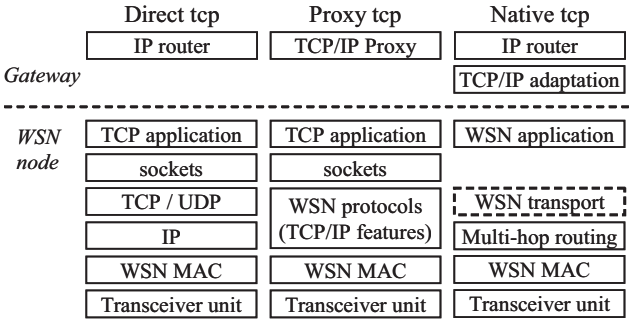


Figure 1: Host and node protocol stacks for different TCP/IP and WSN integration approaches.

tocols and platforms. The implementation of TCP/IP adaptation for Linux and TUTWSN nodes is presented in Section IV and evaluation results in Section V. Finally, conclusions are given in Section VI.

II. WSN AND TCP/IP CHARACTERISTICS

WSNs are characterized by resource constraints and tight integration to the real world. In addition, the communication paradigm differs considerably from that of traditional TCP/IP networks. In WSNs, individual nodes and their identification are not important. Instead, data and data interests are targeted to a location, or to a set of nodes that fulfil the conditions specified in the data or interests [2].

A. WSN Challenges

WSNs are *data-centric* with an objective to deliver time sensitive data to different destinations. Further, a deployed WSN is *application-oriented* and performs a specific task. Due to the tight integration to environment, WSN applications have strict realtime and security requirements [2].

The challenges for WSN protocols are the high density and scale, and unpredictability caused by communication errors and temporarily unavailable nodes. Due to the scale and the nature of deployment, WSNs must operate autonomously, meaning that they must be self-configuring, self-operating, and self-maintaining [1].

The application requirements and communication challenges must be met within the strict constraints set by node platforms. The program and data memory, and processing capacity in WSN nodes are very limited. Further, the nodes are typically battery-powered or the operating energy is scavenged from the environment. Therefore, the energy resources are limited and need to be preserved [1].

B. WSN and TCP/IP Relations

The architecture for TCP/IP networking over WSN discussed in this paper is depicted in Fig. 2. Two or more TCP/IP endpoints (user/server) communicate over WSN through gateways and intermediate nodes. The base WSN infrastructure, sensing operations, and data relaying are maintained simultaneously with TCP/IP communication. The path for TCP/IP communication is referred to as *tcp route* but WSN data are sent also along that route. In case of direct and proxy tcp, every WSN node may be a part or a communication endpoint of the tcp route, while in native tcp nodes only relay TCP/IP data between the gateways.

There are several conceptual differences between WSNs and TCP/IP networks in communication paradigm, flow control, and predictability. In contrast to data-centric WSNs, TCP communication is connection-oriented between specific endpoints. The end-to-end connection maintenance and flow control generates considerable amount of control traffic between connection endpoints. In WSNs, the communication over wireless links is the dominant cause for energy consumption. Therefore, the flow control is performed over a single hop in order to avoid costly end-to-end retransmissions [4].

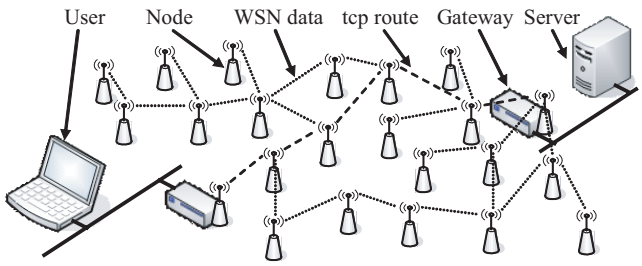


Figure 2: Networking architecture for TCP/IP communication over WSN.

TCP flow control assumes that all communication errors are caused by congestion while the main reasons for errors in WSNs are random bit errors, topology changes, and temporarily unavailable nodes. Further, TCP assumes symmetric uplink and downlink in a connection, but in WSNs their delay and throughput may differ considerably [3, 4].

While direct tcp faces the general TCP/IP drawbacks in wireless environments, proxy tcp allows energy efficient WSN operation but leads to buffering and flow control adaptation challenges in gateway TCP/IP proxy. The generality of native tcp is limited but it is an energy efficient approach for random TCP/IP communication with moderate load.

1) TCP/IP Adaptation to WSNs

In direct tcp, WSN uses legacy TCP/IP for communication. Instead proxy tcp and native tcp require the adaptation of TCP flow control, IP addressing, and packet fragmentation. In proxy tcp, the TCP/IP proxy in gateway converts these to corresponding WSN counterparts. The TCP/IP adaptation layer in gateway handles the addressing and fragmentation in native tcp but relays the flow control over WSN.

The transmission rate in TCP depends on the sizes of the peer receive window and sender congestion window (CWND). The congestion window is resized based on the networking conditions. The utilized window control algorithms depend on the TCP version. In general, the algorithms use received acknowledgements and timeouts for controlling their decisions.

In this paper, we use one of the most widely adopted TCP version, TCP Reno [15]. TCP Reno has four phases, *slow start*, *congestion avoidance*, *fast recovery*, and *fast retransmit*. In slow start, CWND is increased exponentially on reception of an acknowledgement until a threshold value is reached. After this, in congestion avoidance phase CWND is increased linearly on successful transmission. On an error situation, fast recovery and retransmit algorithms are applied. These decrease the size of CWND and retransmit lost or reordered packets in order to avoid slow start phase, which is reactivated by a retransmission timeout (RTO).

The window-based congestion control algorithms of TCP may lead to the aggressive reduction of the transmission rate due to a single bit error or topology change [4]. This is accounted in proxy tcp, which instead of legacy TCP algorithms uses WSN specific flow control algorithms.

The 32-bit addresses in IPv4 or 128-bit addresses in IPv6 must be mapped to WSN addresses in gateways for the routing of TCP data in WSN. Further, a typical IP packet fragment size is 1500 B in Ethernet networks. If WSN does not support frames of that size, these must be fragmented to WSN frames in the gateway.

2) *WSN Configuration for TCP*

In general, direct and proxy tcp WSNs are targeted for TCP/IP communication. In case of native tcp, TCP/IP communication is relayed as application layer traffic. Basically, a low-power WSN throughput is not sufficient for TCP/IP networking. If possible, the WSN MAC and routing protocols should be configured for high throughput and short delay. This can be realized by increasing duty cycle, through route selections, or by diminishing the number of hops by increasing transceiver transmit power.

Due to the limited resources in WSN nodes, existing communication conventions should be exploited as much as possible. Thus, additional code and data memory usage should be minimized. A more frequent duty cycle or higher transmit power results to an increased energy consumption. This cannot be avoided, but it can be adjusted as a trade-off between throughput and energy. Due to the energy trade-off, the number of nodes that are affected by the TCP/IP communication should be minimized. Thus, if possible the configurations should be restricted only to the nodes that are part of tcp routes. This minimizes the effect of the adaptation on overall network performance.

III. TUTWSN Protocols and Platforms

TUTWSN is a full feature framework that consists of several applications, a customizable protocol stack, a family of node platforms, and different monitoring and control user interfaces for mobile and PC gateways. The operation of TUTWSN has been tested and its features improved based on the information gathered on several deployments in real environments with over hundred of nodes [13, 14].

A. *TUTWSN Protocols*

TUTWSN uses a TDMA-based MAC protocol, which minimizes transceiver idle listening time. A clustered network topology is used for improving the overall network energy efficiency. The gradient-based routing in TUTWSN optimizes data relaying costs in network level.

1) *TUTWSN MAC Protocol*

The clustered topology is maintained by a *cluster headnode*. A headnode controls communication within the cluster and performs inter-cluster data routing with other headnodes. Other nodes, referred to as *subnodes*, communicate only with their associated headnode.

The access cycle of a cluster, maintained by the headnode, is depicted in Fig. 3. The headnode transmits periodic *network beacons* advertising its presence on a network channel. These beacons are used for the self-organization and for the control of inter-cluster communication. Each cluster operates on a dedicated cluster channel, which is selected so that it does not overlap with the channels used by the neighbor clusters and their neighbors.

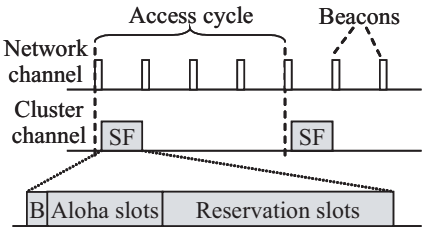


Figure 3: TUTWSN access cycle.

The communication within the cluster and inter-cluster data transfers are made during a superframe (SF). SF is started by a *cluster beacon*, which contains cluster timing and slot reservation information. The beacon is followed by *ALOHA slots*, in which subnodes and neighbor headnodes may send association and slot reservation requests, and occasional data. *Reservation slots* are allocated by the headnode for periodic data transfers. A slot consists of uplink and downlink parts, each being 10 ms.

2) *TUTWSN Routing Protocol*

Our routing protocol creates routes based on the cost-gradients to a gateway. Data routing is performed only when there are active data interests for WSN.

An interest initiated by a gateway is flooded to the network by unicast transmissions. Each headnode that receives an interest scans for nearby clusters and forwards the interest separately to all clusters, except the one it was received from. When forwarding, the headnode adds its cost to the interest. A headnode determines the next hop based on the cost gradients of the received interests. The cost depends on the number of hops to the destination, remaining energy, local loading (number of associated nodes), and the transceiver transmit power required to reach the next hop node.

A node scans the network channel periodically for new neighbors and interests. Once a new next hop is selected, a node associates to the cluster at the MAC layer. A node can maintain simultaneously several alternative routes. The cost gradient for a route is signaled in the MAC layer cluster beacons, thus routing can be easily adapted. Interests are periodically updated by broadcasting them in the downlink slots so that they traverse to the reverse direction along the routes. Thus, after the initial route formation, the energy required for the network maintenance is minimal.

B. *TUTWSN Node Platform*

The TUTWSN node platform, which is used in this paper, is illustrated in Fig. 4. The main component is PIC18LF4620 nanowatt series Micro-Controller Unit (MCU) with 64 KB code and 3986 B data memory. MCU contains also an 10-bit integrated Analog-to-Digital Converter (ADC) and 1 KB of EEPROM as a non-volatile data storage. The power unit con-

Figure 4: TUTWSN PIC node.

sists of a MAX1725 regulator with 2.4 V output voltage and a 3 V CR123A lithium battery. In addition, a DS620 digital thermometer is integrated to the platform.

The radio interface on the platform is a 2.4 GHz nRF2401A transceiver unit, which supports 1 Mbit/s data rate and transmit power between -20. . .0 dBm. The integrated Cyclic Redundancy Check (CRC) calculation and address detection logic of the transceiver unit are used by TUTWSN MAC for the error detection and as the network address. This limits the packet size to 32 B leaving 26 B for MAC after 4 B address and 2 B CRC.

IV. TCP/IP Adaptation Implementation for TUTWSN

We have selected native tcp approach for evaluation, because of its suitability for low-power WSNs, such as TUTWSN. For the evaluation we have implemented software for Linux PCs operating as WSN gateways and configured TUTWSN to a throughput optimized mode. A Linux PC is connected to a gateway WSN node through a serial port with 57600 baud rate.

A. TCP/IP Adaptation Layer Implementation on Linux

Our TCP/IP adaptation layer is implemented by Linux WSN Adaptation (LWA) application on Ubuntu Linux with kernel 2.6.8.1. The software architecture for Linux is depicted in Fig. 5. A network application (or IP router) communicates through a legacy Linux networking stack consisting of sockets and protocol layers. A Universal TUN/TAP driver [16] passes IP packets to LWA, which fragments and relays data to WSN through a serial port.

LWA performs addressing adaptation and IP packet fragmentation to WSN frames, whereas legacy Linux TCP flow control is used. An IP address is mapped to a WSN node address based on the information gathered during connection initiation.

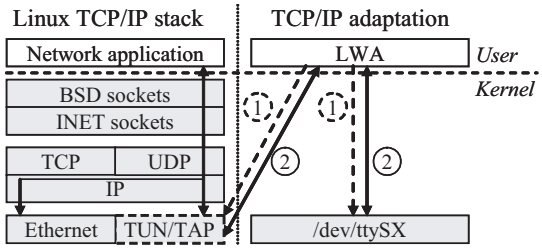


Figure 5: Linux implementation software architecture.

In startup phase (1), LWA creates an instance of a TUN/TAP driver and initiates a file handle to the serial port. TUN/TAP driver network interface is automatically configured as a separate subnet with *ifconfig* command. The data communication between the TUN/TAP driver and LWA during the active phase (2) is made through a file handle retrieved during the initialization. When LWA writes to the file handle data are passed to IP. Respectively, data sent by IP can be read from the file handle. The communication with a gateway WSN node is made similarly through a file handle (/dev/ttySX). The serial port is configured with low latency option using *setserial* command.

B. TUTWSN Protocol Configuration

In TUTWSN, a cluster headnode maintains its own access cycle and communicates with the next hop during SF of that cluster. Rest of the time is spent on a sleep mode with periodically sent network beacons. This idle time can be utilized for configuring the network for higher throughput and shorter delay. Because the routes in TUTWSN are gradients to the gateways, tcp routes are formed automatically. Yet, as uplink and downlink are always successive, the same route is used for both directions.

The idle time can be used efficiently, if the start times of SFs are adjusted so that

$$n_i = n_{i-1} + (a - (l + o_n)), \tag{1}$$

where n_i is the SF start time of ith cluster from the gateway and n_{i-1} the start time for the next hop cluster. The length of an access cycle is denoted by a, the maximum length of SF by l, and o_n is the interval desired between the end of own SF and the start of the next hop cluster SF.

TUTWSN access cycle is configured to a high throughput mode by adding additional SFs during the idle period, as depicted in Fig. 6. Original SFs are used for WSN communication, while the additionals SFs are for TCP data. Because same SF structure is used, additional code is needed only for control.

An additional SF is started by a beacon, which contains timing and slot reservation information. Reservation slots are for associated nodes that are part of the active tcp routes. The start time for jth additional SF during the access cycle is

$$s_j = n + j(l + o_s) \qquad j \in \{1, 2, \ldots, x\}, \tag{2}$$

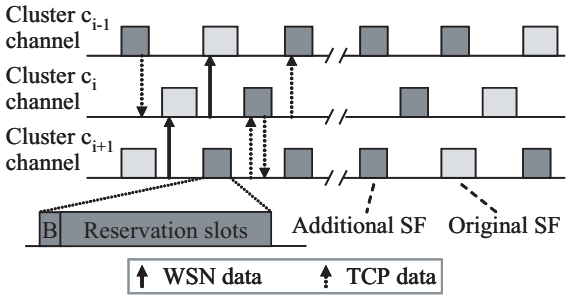


Figure 6: Principle of access cycle adaptation.

where n and l are the start time and length of the original SF, respectively, and o_s the idle time between the additional SFs. The number of additional SFs (x) during an access cycle depends on the lengths of the idle and SF periods.

V. Evaluation results

The evaluation is performed by measuring TCP and WSN performance with physical TUTWSN prototypes and two Linux laptops. TCP internal operation is monitored during active connections over WSN. The applicability of WSN for native tcp is evaluated with HTTP and FTP applications.

For the evaluation purposes, TUTWSN nodes on a tcp route utilize modified SFs. SF consists of cluster beacons, one ALOHA slot, and eight reservation slots. The interval between SFs is 500 ms. Per-hop acknowledgements are used to guarantee frame delivery. TCP version used in the experiments is the default Linux kernel TCP Reno.

A. TCP performance

TCP performance is evaluated by monitoring sender CWND and RTO when a 200 KB file is transmitted through a TCP connection over TUTWSN. The data transfer is made with *netcat*. The error situations are generated to the tcp route by removing and adding nodes.

The variation of sender CWND and RTO values during the connection (from initiation to completion of the data transfer) is depicted in Fig. 7. On top of the figure are the number of hops in the tcp route at the given time intervals and generated topology changes.

A removal of a node from the tcp route leads to the loss of TCP data currently processed by the node, whereas a node addition does not cause errors. As shown in the figure, when data are lost the size of CWND decreases and RTO increases. However, WSN is fully operational almost immediately after the node removal and no congestion occurs. Thus, even though TCP Reno attempts to avoid activation of slow start algorithm, CWND is still downsized too aggressively for WSNs. The RTO is derived from a smoothed Round Trip Time (RTT) of the network. It is increased by the errors but stabilizes afterwards.

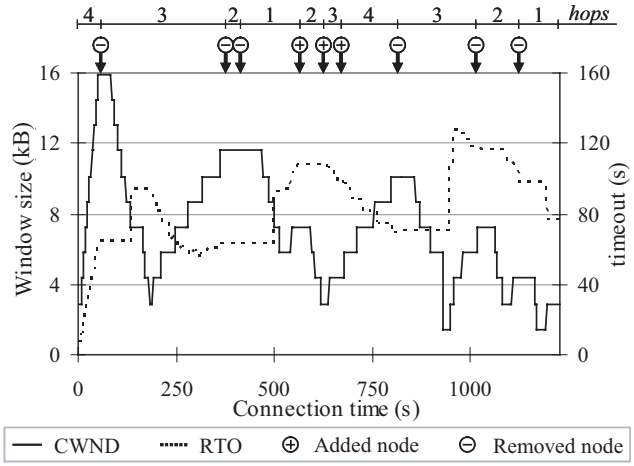


Figure 7: Sender CWND and RTO with varying WSN topology during 200 KB TCP data transfer.

B. Round Trip Time

The performance of an HTTP application depends on RTT of the network. RTTs for a tcp route in TUTWSN are measured using *ping* command. Fig. 8 depicts RTTs with IP packet sizes of 50, 100, 200, 500, and 1000 bytes for different number of hops. The presented results are averaged over 50 pings. The given packet size includes 20-byte IP and 8-byte Internet Control Message Protocol (ICMP) headers. RTTs include Linux PC processing but it is negligible. Yet, due to TDMA in WSN there may be at most 250 ms jitter in the delay at the initiation of transmission.

As shown, RTTs increase steadily for greater number of hops. For the same number of hops, RTT depends on the number of SFs that are required for transmitting the fragments of a packet. Because 50 B and 100 B packets can be sent within a single SF, their results are similar.

C. Throughput

In the used TUTWSN configuration, a single frame carries 19 B of TCP data as payload. Thus, with 500 ms access cycle and eight reservation slots, the theoretical maximum throughput for a tcp route is 2432 bps. The measured throughput of an FTP application through a tcp route for different number of hops is depicted in Fig. 9.

The FTP application is modeled by *netcat* and the throughput is measured at the receiving Linux PC for successfully received IP packets. The given throughput values are averaged over 5 minutes of constant loading. Discarded packets are not included in the results. This and the usage of acknowledgements and retransmissions for guaranteed delivery are the reasons why theoretical maximum is not reached.

D. Power Consumption

In a TDMA-based TUTWSN, a more frequent duty cycle leads to the increased power consumption. The power consumption

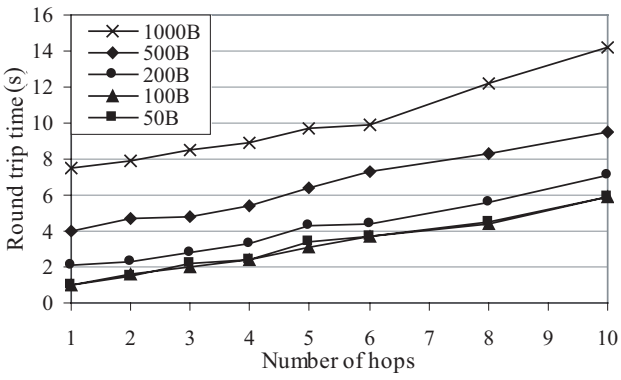


Figure 8: Round trip time with varying packet size for different number of hops.

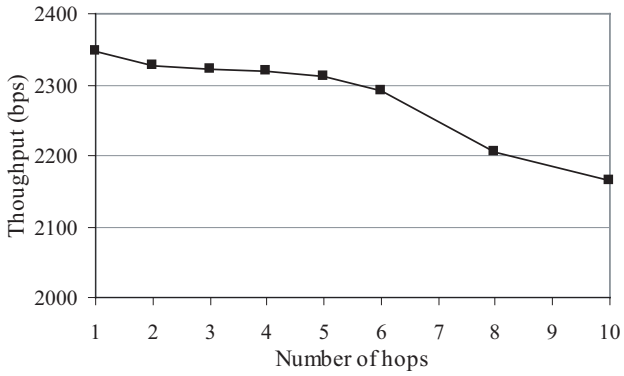


Figure 9: Measured throughput for a tcp route for different number of hops.

has been measured by supplying a node with a 0.4 F capacitor. An average current consumption is determined by the slopes of the capacitor terminal voltages. From the currents, the power consumption is calculated with 3.0 V supply voltage. The power consumption of a node that is part of a tcp route is measured separately when maintaining a connection and during maximum loading.

During the tcp route maintenance without TCP data traffic, a WSN node sends two network and cluster beacons, and listens for ALOHA slot and two reservation slots. Further, it receives the cluster beacon of the next hop node. The measured average power consumption of a node is 1.73 mW. During the maximum loading the average power consumption is 3.82 mW.

As a comparison, in an environmental monitoring TUTWSN [14] with two second access cycle a cluster headnode that routes data from a single subnode to the next hop cluster consumes averagely 0.72 mW. The headnode maintains its own access cycle, depicted in Fig. 3, and sends the data received from subnode and initiated by itself to the next hop during SF of that cluster. Both nodes initiate a new frame approximately every tenth second. Thus, the increase of the power consumption due to the TCP communication is not significant considering the increased activity and more frequent duty cycle.

VI. Conclusions and Future Work

This paper experiments the performance of TCP/IP in a low-power WSN. The differences in their communication profiles are considerable, but TCP/IP and WSN integration is needed for ubiquitous computing. The presented results indicate that TCP flow control algorithms perform poorly in WSNs and low-power WSNs need a frequent duty cycle in order to provide adequate throughput. Further improvements can be achieved by using a proxy approach that diminishes the header overhead and adapts TCP flow control.

Our future work will focus on the the better integration of the throughput optimized mode to the environmental monitoring TUTWSN. In TUTWSN, the trade-off between performance and energy can be adjusted by shortening or lengthening slots and access cycle. Yet, this requires additional considerations in the implementation in order to maintain the basic WSN operations.

References

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "A survey on sensor networks," *IEEE Communications Magazine*, vol. 40, no. 8, pp. 102–114, Aug. 2002.

[2] J. A. Stankovic, T. F. Abdelzaher, C. Lu, L. Sha, and J. C. Hou, "Real-time communication and coordination in embedded sensor networks," *Proceedings of the IEEE*, vol. 91, no. 7, pp. 1002–1022, July 2003.

[3] Y. Tian, K. Xu, and N. Ansari, "Tcp in wireless environments: Problems and solutions," *IEEE Communications Magazine*, vol. 43, no. 3, pp. S27–S32, Mar. 2005.

[4] W. Chonggang, K. Sohraby, H. Yueming, L. Bo, and T. Weiwen, "Issues of transport control protocols for wireless sensor networks," in *Proc. 2005 Int. Conf. on Communications, Circuits and Systems*, HongKong, China, May 2005, pp. 442–426.

[5] M. Weiser, "Hot topics-ubiquitous computing," *Computer*, vol. 26, no. 10, pp. 71–72, Oct. 1993.

[6] H. Elaarag, "Improving tcp performance over mobile networks," *ACM Computing Surveys*, vol. 34, no. 3, pp. 357–374, Sept. 2002.

[7] D. Göthberg. (2004, June) Micro-ip for embedded systems. [Online]. Available: http://www.micro-ip.org/

[8] Z. Shelby, P. Mähönen, J. Riihijärvi, O. Raivio, and P. Huuskonen, "Nanoip: the zen of embedded networking," in *Proc. IEEE International Conference on Communications*, vol. 2, Anchorage, Alaska, USA, May 2003, pp. 1218–1222.

[9] A. Dunkels, "Full tcp/ip for 8-bit architectures," in *Proc. 1st International Conference on Mobile Systems, Applications and Services*, San Francisco, CA, USA, May 2003, pp. 85–98.

[10] L. Xiaohua, Z. Kougen, P. Yunhe, and W. Zhaohui, "A tcp/ip implementation for wireless sensor networks," in *Proc. IEEE International Conference on Systems, Man and Cybernetics*, vol. 7, Hague, Netherlands, Oct. 2004, pp. 6081–6086.

[11] I. Solis and K. Obraczka, "Flip: a flexible interconnection protocol for heterogeneous internetworking," *Mobile Networks and Applications*, vol. 9, no. 4, pp. 347–361, Aug. 2004.

[12] A. Dunkels, J. Alonso, T. Voigt, H. Ritter, and J. Schiller, "Connecting wireless sensornets with tcp/ip networks," in *Proc. 2nd International Conference on Wired/Wireless Internet Communications*, ser. Lecture Notes in Computer Science, vol. 2957. Frankfurt, Germany: Springer, Feb. 2004, pp. 143–152.

[13] M. Kohvakka, M. Hännikäinen, and T. D. Hämäläinen, "Ultra low energy wireless temperature sensor network implementation," in *Proc. 16th Annual IEEE International Symposium on Personal Indoor and Mobile Radio Communications*, Berlin, Germany, Sept. 2005.

[14] J. Suhonen, M. Kohvakka, M. Hännikäinen, and T. D. Hämäläinen, "Design, implementation, and experiments on outdoor deployment of wireless sensor network for environmental monitoring," in *Proc. Embedded Computer Systems: Architectures, Modeling, and Simulation*, Samos, Greece, July 2006, accepted.

[15] K. Fall and S. Floyd, "Simulation-based comparisons of tahoe, reno and sack tcp," *Computer Communication Review*, vol. 26, no. 3, pp. 5–21, July 1996.

[16] M. Krasnyansky. (2005, Aug.) Universal tun/tap driver. [Online]. Available: http://vtun.sourceforge.net/tun/

PUBLICATION 6

M. Kuorilehto, J. Suhonen, M. Hännikäinen, T. D. Hämäläinen, "Tool-Aided Design and Implementation of Indoor Surveillance Wireless Sensor Network," in *Embedded Computer Systems: Architectures, Modeling, and Simulation*, Lecture Notes in Computer Science, vol. 4599, S. Vassiliadis, M. Bereković, T. D. Hämäläinen (Eds.), Springer-Verlag, Heidelberg, Germany, 2007, pp. 396–407.

Tool-Aided Design and Implementation of Indoor Surveillance Wireless Sensor Network

Mauri Kuorilehto, Jukka Suhonen, Marko Hännikäinen, and Timo D. Hämäläinen

Tampere University of Technology, Institute of Digital and Computer Systems
P.O. Box 553, FI-33101 Tampere, Finland
{mauri.kuorilehto, jukka.suhonen, marko.hannikainen,
timo.d.hamalainen}@tut.fi

Abstract. This paper presents the design and implementation of an indoor surveillance Wireless Sensor Network (WSN) using tools for hastening and facilitating the different phases in the WSN development. First, the application case is described in WISENES (WIreless SEnsor NEtwork Simulator) framework by four models, which define application, communication, node, and environment. WISENES enables a graphical design of the models combined with accurate simulations for performance evaluation. Next, surveillance application tasks and communication protocols are implemented on node platforms on top of SensorOS Operating System (OS). A congruent programming model of SensorOS allows a straightforward mapping of WISENES models to the final implementation. The evaluation of the indoor surveillance WSN implemented with Tampere University of Technology WSN (TUTWSN) protocols and platforms reaches a lifetime in order of years while still ensuring reactive operation. Further, the results show only 9.5 % and 6.6 % differences in simulated and measured networking delay and power consumption, respectively. Our results indicate that accurate early design phase simulations can relieve the burden of prototyping and low level implementation by a realistic configuration evaluation during design time.

1 Introduction

Wireless Sensor Networks (WSN) are an emerging ad hoc networking technology, in which a large number of miniaturized sensor nodes organize and operate autonomously. Communication, computation, energy, and memory capacities of individual nodes are limited, but the overall network capability results from the cooperation of nodes [1].

The envisioned applications for WSNs are diverse in environmental monitoring, home, industry, health care, and military. In spite of the diversity of applications, they possess common domain independent characteristics. A typical application gathers measurement data from different sensors, aggregate them, and route data to a central gathering point, a *sink*. Alternatively, nodes perform in-network data fusion and make either independent or distributed control actions through actuators [2].

The communication in WSNs is controlled by a layered protocol stack. The key layers are Medium Access Control (MAC) that manages channel access and network topology, and routing that creates and maintains multi-hop paths between end points [1]. The communication requirements for a WSN are application-specific, thus a single protocol stack is not suitable for all cases. Yet, the maturity and properties of WSN

S. Vassiliadis et al. (Eds.): SAMOS 2007, LNCS 4599, pp. 396–407, 2007.

protocol stacks are evolving, which allow the optimization of existing protocols and software architectures for a variety of applications through configuration [3].

As the number of design choices and complexity of WSN applications increases, the management of the vast design space and the configuration exploration for an application requires design automation tools. Until recently, design automation has not been considered in WSN community, but the main focus of research has been on energy efficient and scalable protocols [2]. However, considering the rapid evolution, the burden of WSN design without tools will evidently be unbearable. In order to substantially hasten and facilitate WSN development, tools need to support all phases in the WSN design and help the designer to make reasonable the design choices [4].

The WSN design flow used in this paper is presented in Fig. 1. Design dimensions extract the key parameters from the application requirements for steering system design and implementation phases. In the design phase, a system is divided into separate models for *application*, *communication*, *node*, and *environment*. These models are defined, configured, and evaluated to obtain a suitable system composition for hardware and software implementation. The system is evaluated by both simulations and prototyping. In general, simulations reveal possible performance tweaks in large-scale and long-term deployments, while prototyping verifies the operation of the implementation in its final execution environment. Before the final deployment, required phases are iterated until application requirements are met.

In this paper, we present the design and implementation of an indoor surveillance WSN using support tools. For WSN design, our WIreless SEnsor NEtwork Simulator (WISENES) [5] defines methods for the formal description of application and communication model functionality and dependencies. WISENES allows accurate performance evaluation of graphical Specification and Description Language (SDL) models through simulation. For the final deployment, application and communication models are implemented on top of SensorOS [6] Operating System (OS) that offers a congruent interface with WISENES. The indoor surveillance application is designed and implemented with TUTWSN (Tampere University of Technology WSN) node platforms and protocols [7]. The evaluation of the design case shows the feasibility of the tools and their applicability for rapid development of application-specific WSNs.

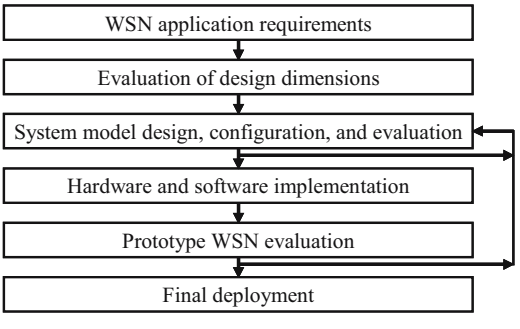


Fig. 1. Different phases in the proposed WSN design flow

The rest of the paper is organized as follows. Section 2 discusses related research in area of WSN design. The requirements for the surveillance application, and TUTWSN protocols and platforms are presented in Section 3. Section 4 shows WSN design with WISENES. The prototype implementation of the surveillance WSN on top of SensorOS is presented Section 5. Finally, conclusions are given in Section 6.

2 Related Work

While a wide variety of system architectures are proposed for WSNs, TinyOS [8] has gained the most popularity. TinyOS is a component-based OS for event-driven WSNs. TinyOS component development is facilitated by nesC [9] programming language that adopts the TinyOS programming model. TOSSIM [10] implements a simulation environment for large scale WSNs built with TinyOS. For data gathering applications, TinyOS can be supplemented with TinyDB [11] that abstracts WSN as a database.

Several higher abstraction level design tools are also proposed for TinyOS systems. VisualSense [12] extends Ptolemy II with WSN features and allows the development of WSN protocols and applications using different Models-of-Computation (MoC) available in Ptolemy II. Applications developed with VisualSense are integrated to TinyOS and TOSSIM through a Viptos interface. GRATIS [13] introduces a graphical tool for TinyOS component design and management. In [14], the mapping of applications implemented as SDL models to TinyOS components is proposed. A loose relation to TinyOS is present also in Prowler [15] that is a MATLAB-based simulation environment targeted for application algorithm testing and optimization for TinyOS nodes.

A platform based design methodology for WSNs is proposed in [4]. The approach is based on three abstract models for applications, protocols, and hardware platforms. A Rialto tool defines requirements for the protocols by exploring all possible communication combinations of the application model. The protocol stack configuration is optimized to meet application requirements in the constraints set by platform candidates. A quite similar approach is taken in [16], in which communication protocols and platform are abstracted to a virtual architecture for algorithm design and synthesis. The proposed design flow concentrates on the modeling concepts and does not provide tool support. In [17], a system level design methodology is proposed for cost function guided optimization of mainly hardware parameters. The tool supported optimization utilizes static network graphs and energy models for design space exploration.

From the related proposals, most are singular tools addressing only a minor part of the WSN design. The platform based design in [4] and VisualSense are nearest to our approach. Compared to [4], WISENES allows the graphical design of not only application models but also communication protocols. Further, the mapping of the models to the final implementation is more straightforward through SensorOS. Compared to VisualSense, WISENES design abstractions and interfaces are more comprehensive and oriented for WSN applications in particular. Further, due to detailed modeling WISENES outputs more accurate performance results in earlier phase, which hastens the overall development by avoiding unnecessary iterations caused by flawed design choices.

3 Indoor Surveillance WSN

An indoor surveillance WSN monitors temperature and detects motion in the public premises of a building. The WSN has three active tasks; motion detection, temperature sensing, and a sink task for data gathering. The relations between tasks together with the basic network architecture are illustrated in Fig. 2a.

The surveillance WSN is designed and implemented with TUTWSN protocols and platforms. In addition to a configurable protocol stack and a family of node platforms, TUTWSN consists of several applications and different monitoring and control User Interfaces (UI) [7]. TUTWSN is accessed through an Application Programming Interface (API) that defines the data interests for the network.

3.1 Surveillance WSN Requirements

The motion detection task interfaces a Passive Infra-Red (PIR) sensor for generating movement alerts, which are forwarded to the sink task. The temperature sensing task measures surrounding temperature periodically and sends it to the sink task. Temperature sensing task is activated once per minute in all nodes. The motion detection task is present only on nodes located in public premises, such as isles. The sink task is executed on a gateway that connects WSN to external networks. The sink stores data to a database and forwards alerts to monitoring UIs.

The requirements for the two sensing tasks differ significantly. The motion detection task is event-based and activated by movement. Generated alerts have high priority, are delay critical, and need reliable transmission. Instead, periodic temperature measurements are low priority packets and occasional data losses are acceptable. In order to avoid constant maintenance, the WSN should operate approximately a year without battery replacements.

3.2 TUTWSN Protocols

TUTWSN MAC protocol combines slotted-ALOHA and reservation data slots for adaptive and extremely energy efficient operation. The clustered topology is maintained with periodic beacons by cluster *headnodes* that also perform inter-cluster communication by synchronizing to neighbor headnodes. *Subnodes* maintain synchronization and communicate only with their parent headnodes. The MAC protocol provides a reliable data transmission service for upper layers. A bandwidth allocation within a cluster is controlled by an adaptive algorithm that reacts to the communication profile changes.

TUTWSN routing protocol forms routes towards a sink based on cost-gradients. Each node maintains several alternative routes each with a different cost function. Routing selects typically two or three synchronized parents for MAC according to the next hops for routes. The cost function used for application data depends on the traffic class. The cost information is updated in the network maintenance communication, thus additional control communication is needed only for adaptive recovering from link failures.

TUTWSN protocols can be tailored for different kinds of applications with a rich set of configuration parameters during both design and runtime. Design time MAC layer configuration includes e.g. access cycle and network maintenance timing, role selection

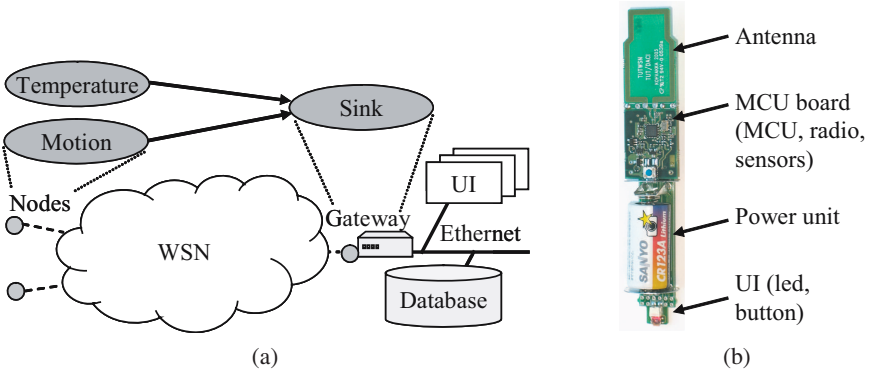


Fig. 2. An overview of (a) surveillance WSN architecture and (b) TUTWSN PIC node

directions, intervals for topology control, and bandwidth reservation and usage parameters. Cost function coefficients and protocol reactiveness can be configured for routing at design time. During runtime, MAC layer can be configured by altering number of Time Division Multiple Access (TDMA) slots and node roles, and routing by selecting used cost functions. The applications of a network can be configured by changing the data interests.

3.3 TUTWSN Prototype Platform

TUTWSN node platform used in this paper is illustrated in Fig. 2b. The main component is PIC18LF4620 nanowatt series Micro-Controller Unit (MCU) with 64 KB code and 3986 B data memory. MCU contains also an 10-bit integrated Analog-to-Digital Converter (ADC) and 1 KB of EEPROM as a non-volatile data storage. The power unit consists of a MAX1725 regulator with 2.4 V output voltage and a 3 V CR123A lithium battery with 1600 mAh capacity. In addition, a DS620 digital thermometer is integrated to the platform. A PIR-sensor is attached to a connector provided for external sensors. The radio interface on the platform is a 2.4 GHz nRF2401A transceiver unit, which supports 1 Mbit/s data rate and transmit power between -20...0 dBm.

4 WSN Design with WISENES

In WISENES [5], protocols and applications are designed in high abstraction level using SDL. SDL components, blocks, can be structured hierarchically in order to clarify the presentation, while the functionality is implemented in processes described as Extended Finite State Machines (EFSM). Processes communicate with signals that initiate state transitions at the recipient processes. Such a programming model is suitable for communication systems and for WSN applications that are typically activated by an event and alter their state and processing depending on the events and their parameters.

WISENES utilizes commercial Telelogic TAU SDL Suite for the graphical design of protocols and applications, and for the code generation of simulation cases. The core functionality, modeling approach, simulation framework, and environment and platform

description are independent of the Telelogic tools. The characteristics of the environment and sensor nodes are defined accurately for WISENES in a set of eXtensible Markup Language (XML) configuration files. The detailed parameters and the realistic modeling of wireless transmission medium, physical phenomena, and sensor node hardware capabilities result to accurate performance information of simulated protocols, nodes, and networks. Simulation results are stored to logs for the post-processing of the energy, processing, memory, and sensor usage statistics for individual nodes. Further, networking performance for the whole WSN and for different applications is output in terms of delays, throughput, collisions, and bandwidth utilization.

4.1 WISENES Model Abstraction

WISENES incorporates four models defined by a designer. These are *application model*, *communication model*, *node model*, and *environment model*. The hierarchy and main properties of the models are depicted in Fig. 3. The environment model defines the parameters for wireless communication (signal propagation, noise), describes overall characteristics (average values, variation) for different phenomena, and specifies separate target areas (e.g. buildings) and objects (e.g. humans, animals, vehicles). The environment model defines also the mobility of nodes and target objects.

WISENES application model allows a designer to describe the functionality and requirements of an application separately. This eases the exploration of the performance and suitability of application configurations for different kinds of networks and environments. The functionality of an application is divided into tasks that are implemented as SDL statecharts. The operational parameters and requirements of the application are specified in XML configuration files. These parameters define the dependencies between the application tasks, task activation patterns, and Quality of Service (QoS) requirements for the applications. The QoS parameters define task priorities, networking requirements in terms of data reliability and urgency, and an overall network optimization target. The optimization target is used to steer communication model and it can be for example a maximal network lifetime, load balancing, or high performance.

The communication model specifies the networking for WSN applications. The WISENES communication model consists of a protocol stack implemented in SDL and

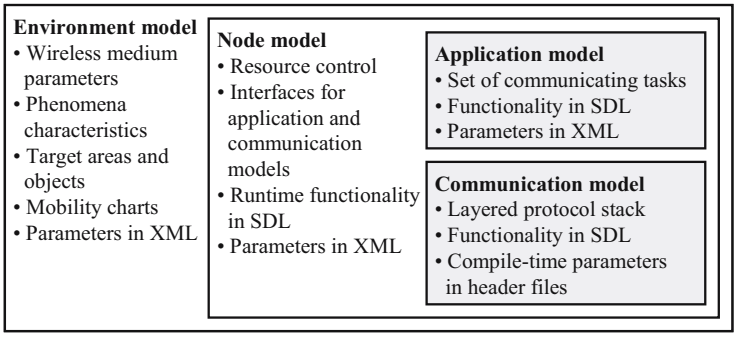


Fig. 3. WISENES models for designer and their main characteristics

a set of configuration parameters. Protocol configuration parameters are set at design time, while application-specific requirements, such as network optimization target, are input from the application model during runtime.

The node model describes the characteristics and capabilities of physical node platforms. The node model is implemented by WISENES framework and it is parameterized in XML configuration files that define node resources, peripherals and transceivers. WISENES SDL node model implements an OS type interface to applications and communication protocols for resource management and execution control.

4.2 Surveillance WSN Design

In WSN design with WISENES, the initial selection of the communication protocols and their parameters is made by the designer. It is our belief that hands-on knowledge and past experiences always result to a sophisticated selection for a starting point.

WISENES Model Design. WSN design in WISENES starts with the definition of models. The environment model is defined in XML. For the indoor surveillance WSN case, it specifies a slightly error-prone communication environment, stationary node locations, typical average values for phenomena, and few target objects with random mobility patterns. A node model is implemented for TUTWSN PIC node by describing its physical characteristics in XML configuration files. The main functionality for the designed WSN is defined in application and communication models.

The application model consists of the three tasks. Their parameters are given in XML configuration files. Fig. 4a shows configuration parameters for the motion detection task. The functionality of tasks is implemented as SDL statecharts. WISENES implementation of the motion detection task is depicted in Fig. 4b. The task is activated by two events; a motion detection event from a PIR-sensor and a timer event for PIR-sensor reactivation. A motion detection event triggers a data transmission and the initialization of a timeout, while a timer event reactivates the PIR-sensor. Timeout is needed to avoid continuous alerts. The periodical sensing task initiates a temperature measurement on a timer event, and sends data and initializes the timer after a sensor event. The sink task stores received data to a database.

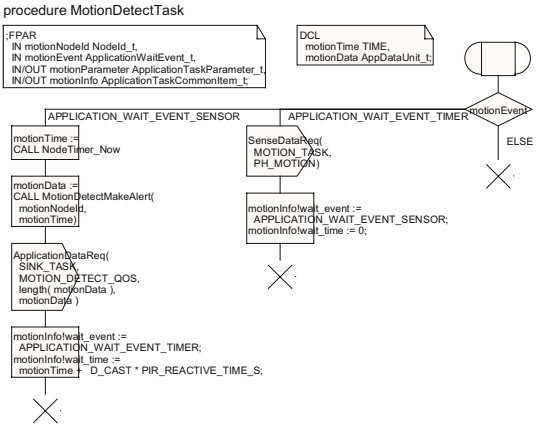
The communication model design for TUTWSN consists of five SDL processes. TUTWSN API is implemented in a single process, while routing and MAC layers are divided into two separate processes. In routing layer, topology and route management are implemented in one process, and data handling in another. In MAC, channel access is implemented separately from TDMA adaptation control. The processes incorporate totally 48 states, 372 transitions, and 4937 different execution paths.

The communication model configuration is based on the application requirements. TUTWSN MAC protocol uses 2 s access cycle to balance energy-efficiency, scalability, and delay-critical operation. The number of ALOHA slots is set to four and reservation slots to eight. Bandwidth allocation parameters are explored to obtain the most suitable configuration. Two different cost functions are defined for routing; a delay optimized for motion alerts and a network lifetime optimized for temperature measurements.

Simulation Results. The performance of the surveillance WSN with the presented model implementations is evaluated with WISENES simulations. A network of 150

```
<application_model>
  <task id="MOTION">
    <interval ms="0"/>
    <priority level="1"/>
    <sensors>
      <sensor id="PIR"/>
    </sensors>
    <data>
      <target task="SINK"/>
      <priority level="1"/>
      <reliable set="yes"/>
      <urgent set="yes"/>
      <opt target="DELAY"/>
    </data>
  </task>
  <task id="TEMPERATURE">
    ...
  </task>
</application_model>
```

(a)



(b)

Fig. 4. Example WISENES application model (a) XML parameters and (b) SDL implementation for motion detection task

semi-randomly deployed nodes is simulated for a 24-hour period. 50 nodes include a PIR-sensor and three operate as a sink, while the rest measure temperature and perform data routing. Different configurations are evaluated by parameterizing bandwidth allocation algorithm. A default reservation slot interval (r) sets the maximum time between granted reservation slots for a member node. In simulations r is set to 6 and 12 seconds.

Application requirements are verified by monitoring the delay of motion alerts, and node power consumption. The delay of temperature measurements is considered for comparison. Fig. 5a shows the average delays of motion alerts and temperature measurements for different r values as the function of number of hops from a sink node (hop count). As shown, with a same hop count, the alerts experience slightly smaller delay than the temperature measurements, because of the delay optimized routing of alerts. Further, in TUTWSN a delay optimized route has typically less hops, which further improves the performance compared to temperature measurements. The average power consumptions of 10 randomly selected headnodes are 650 μW and 635 μW, and of 10 subnodes with PIR-sensor 434 μW and 443 μW for $r = 6$, and $r = 12$, respectively.

The simulation results of initial communication model configuration are acceptable. The configuration with $r = 6$ obtains a slightly better performance and balances networking reactiveness and lifetime. Assuming that 90 % of the battery capacity can be exploited, with 1600 mAh CR123A battery the simulated power consumptions indicate lifetimes of 276 and 414 days for TUTWSN headnode and subnode, respectively. By rotating headnode and subnode roles the network can reach a lifetime of a year.

5 WSN Prototype Implementation

After the design is validated by simulations, the prototype implementation is made. The application tasks and protocols are implemented manually on top of SensorOS [6]

according to the WISENES application and communication models. SensorOS offers a congruent programming interface with WISENES, which makes a fluent transition between phases possible.

5.1 SensorOS

SensorOS is a pre-emptive multi-threading OS targeted for time critical WSN applications and very resource constrained nodes. The scheduling algorithm of SensorOS is priority-based, but for less time critical applications SensorOS incorporates an optional more lightweight kernel with a polling run-to-completion scheduler. SensorOS kernel includes Inter-Process Communication (IPC), timing, memory, and power management services and drivers for interrupt-driven peripherals. Mutexes are included to the pre-emptive kernel for the synchronization of thread execution. Application tasks and communication protocols are implemented as threads on top of SensorOS API.

WISENES interfaces are adopted in SensorOS by message-passing IPC and an event waiting interface. The message-passing allows a similar communication between threads as SDL signals. The event waiting interface enables the implementation of a state-based operation similar to WISENES by offering a single function for the waiting of timeouts, external peripheral events, and IPC messages. Further, the power and memory management in SensorOS are identical to those of WISENES node model.

5.2 Surveillance WSN Implementation on TUTWSN Prototypes

The prototype implementation of the surveillance WSN is realized in a limited scale with 28 nodes (10 with PIR-sensors) on a realistic deployment environment. The full version of SensorOS is used in nodes equipped with PIR-sensors to guarantee reactiveness, while the rest of the nodes have a lightweight kernel that allows longer packet queues. The topology and environment for the prototyped surveillance WSN is depicted on a TUTWSN UI screen capture in Fig. 6. Arrows in the figure show the latest route.

Prototype Implementation. Application tasks are implemented as SensorOS threads. Fig. 7 lists the code of the thread implementing the motion detection task. For

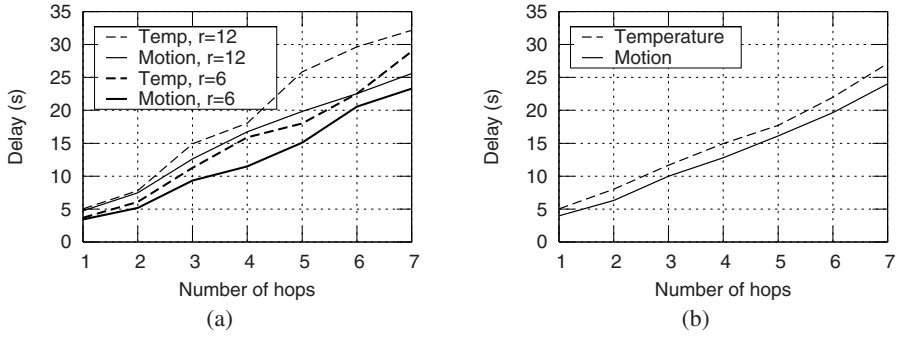


Fig. 5. Average (a) simulated and (b) measured (with r=6) delays for motion alerts and temperature measurements

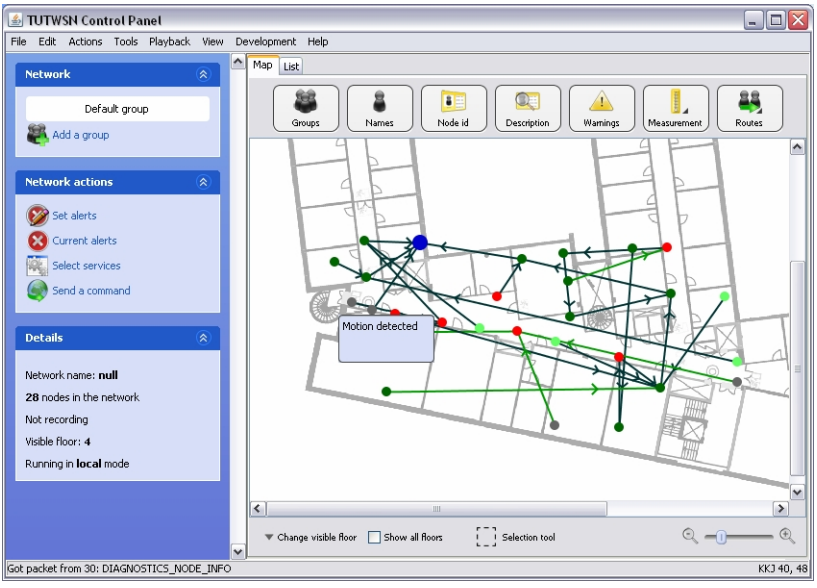


Fig. 6. A screen capture from TUTWSN UI illustrating the prototyped surveillance WSN operation

readability, the details of PIR-sensor interfacing and data message construction are left out. Other application tasks are implemented similarly. The TUTWSN protocol stack is implemented in four threads. API, data routing, and MAC channel access are implemented as separate threads similarly to WISENES communication model design, but MAC and routing layer management operations are integrated to a same thread in order to diminish IPC messaging. TUTWSN protocols are parameterized with the values obtained in WISENES design.

The subnode implementation on TUTWSN PIC node with full feature SensorOS consumes 38.1 KB of code and 2253 B of data memory. These are 60 % and 57 % of available memory resources, respectively. The data memory consumption does not include a heap reserved for dynamic memory. The implementation of temperature measurement application and TUTWSN protocols on top of a lightweight SensorOS kernel takes 58.2 KB code and 2658 B data memory, which are 91 % and 67 % of available memory, respectively.

Prototype Results. The same performance metrics gathered from WISENES simulations are evaluated also for the prototype implementation in order to verify the accuracy of WISENES models and to validate the implementation for final deployment. The delays of motion alerts and temperature measurement data as the function of hop count are depicted in Fig. 5b. In the prototype implementation r is 6 seconds.

The results correspond closely to those obtained from WISENES, average difference being 8.9 % for motion alerts and 10.2 % for temperature data. The reason for a slightly better performance obtained in WISENES is less retransmissions due to a bit optimistic

```
void motion_detect (void) {
    os_eventmask_t event;
    os_ipc_message_t *msg;

    activate_pir ();
    while (1) {
        event = os_wait_event (EVENT_ALARM | EVENT_PIR_INTERRUPT);
        if (event & EVENT_ALARM) {
            activate_pir ();
        } else if (event & EVENT_PIR_INTERRUPT) {
            msg = make_motion_alert_msg (SINK_TASK);
            os_msg_send (API_PID, msg);
            os_set_alarm (PIR_REACTIVATE_TIMEOUT_MS);
        }
    }
}
```

Fig. 7. The implementation of the motion detection application task as a SensorOS thread

environment model used in simulations. The additional loading due to the larger number of nodes in WISENES simulations is balanced by three sinks.

The averages of measured power consumptions are 693 μW for a headnode and 467 μW for a subnode equipped with a PIR-sensor. For comparison, the measured power consumption of a subnode running a lightweight kernel without a PIR-sensor in the same WSN is 257 μW. These are also analogous with WISENES results, average difference being 6.2 % for headnodes and 7.0 % for subnodes with PIR-sensor.

6 Conclusions

In this paper, we present the design and implementation of an indoor surveillance WSN with WISENES and SensorOS. The high abstraction level models are first designed in graphical environment and then implemented on top of a full feature OS on node platforms. By enabling a realistic evaluation of different configurations during the design phase, the presented tools hasten the WSN development significantly. Further, the congruent programming models make the implementation of applications and protocols on top of SensorOS straightforward according to the WISENES models. The surveillance WSN evaluation proves the accuracy of WISENES simulations and shows the suitability of TUTWSN and SensorOS for the application implementation.

References

1. Akyildiz, I.F., Su, W., Sankarasubramaniam, Y., Cayirci, E.: A survey on sensor networks. IEEE Communications Magazine 40(8), 102–114 (2002)
2. Akyildiz, I.F., Kasimoglu, I.H.: Wireless sensor and actor networks: research challenges. Elsevier Ad Hoc Networks 2(4), 351–367 (2004)
3. Römer, K., Mattern, F.: The design space of wireless sensor networks. IEEE Wireless Communications 11(6), 54–61 (2004)
4. Bonivento, A., Carloni, L.P., Sangiovanni-Vincentelli, A.: Platform based design for wireless sensor networks. Mobile Networks and Applications 11(4), 469–485 (2006)

5. Kuorilehto, M., Kohvakka, M., Hännikäinen, M., Hämäläinen, T.D.: High level design and implementation framework for wireless sensor networks. In: Proc. Embedded Computer Systems: Architectures, Modeling, and Simulation, Samos, Greece, pp. 384–393 (2005)
6. Kuorilehto, M., Alho, T., Hännikäinen, M., Hämäläinen, T.D.: Sensoros: a new operating system for time critical wsn applications. In: Proc. Embedded Computer Systems: Architectures, Modeling, and Simulation, Samos, Greece (2007)
7. Suhonen, J., Kohvakka, M., Hännikäinen, M., Hämäläinen, T.D.: Design, implementation, and experiments on outdoor deployment of wireless sensor network for environmental monitoring. In: Proc. Embedded Computer Systems: Architectures, Modeling, and Simulation, Samos, Greece, pp. 109–121 (2006)
8. Hill, J., Szewczyk, R., Woo, A., et al.: System architecture directions for networked sensors. In: Proc. 9th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Cambridge, MA, USA, pp. 94–103 (2000)
9. Gay, D., Levis, P., Behren, R.v., Welsh, M., Brewer, E., Culler, D.: The nesc language: A holistic approach to networked embedded systems. In: Proc. ACM Conference on Programming Language Design and Implementation, San Diego, CA, USA, pp. 1–11 (2003)
10. Levis, P., Lee, N., Welsh, M., Culler, D.: Tossim: accurate and scalable simulation of entire TinyOS applications. In: Proc. 1st ACM Conference on Embedded Networked Sensor Systems, Los Angeles, CA, USA, pp. 126–137 (2003)
11. Madden, S., Franklin, M.J., Hellerstein, J.M., Hong, W.: The design of an acquisitional query processor for sensor networks. In: Proc. ACM International Conference on Management of Data, San Diego, CA, USA, pp. 491–502 (2003)
12. Baldwin, P., Kohli, S., Lee, E.A., Liu, X., Zhao, Y.: Modeling of sensor nets in ptolemy II. In: Proc. 3rd International Symposium on Information Processing in Sensor Networks, Berkeley, CA, USA, pp. 359–368 (2004)
13. Völgyesi, P., Lèdeczi, À.: Component-based development of networked embedded applications. In: Proc. 28th Euromicro Conference, Dortmund, Germany, pp. 68–73 (2002)
14. Dietterle, D., Ryman, J., Dombrowski, K., Kraemer, R.: Mapping of high-level sdl models to efficient implementations for tinyos. In: Proc. Euromicro Symposium on Digital System Design, Rennes, France, pp. 402–406 (2004)
15. Simon, G., Völgyesi, P., Maròti, M., Lèdeczi, À.: Simulation-based optimization of communication protocols for large-scale wireless sensor networks. In: Proc. 2003 IEEE Aerospace Conference. vol. 3., Big Sky, MT, USA pp. 1339–1346 (2003)
16. Bakshi, A., Prasanna, V.K.: Algorithm design and synthesis for wireless sensor networks. In: Proc, International Conference on Parallel Processing, Montreal, Quebec, Canada pp. 423–430 (2004)
17. Shen, C.C., Badr, C., Kordari, K., Bhattacharyya, S.S., Blankenship, G.L., Goldsman, N.: A rapid prototyping methodology for application-specific sensor networks. In: Proc. IEEE International Workshop on Computer Architecture for Machine Perception and Sensing, Montreal, Quebec, Canada (2006)