available at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cosrev

Survey

Distributed algorithm engineering for networks of tiny artifacts☆

Tobias Baumgartner[a], Ioannis Chatzigiannakis[b,c], Sándor P. Fekete[a], Stefan Fischer[d], Christos Koninis[b,c], Alexander Kröller[a], Daniela Krüger[d], Georgios Mylonas[b,c], Dennis Pfisterer[d,*]

[a] Braunschweig University of Technology, IBR, Algorithms Group, Germany
[b] Research Academic Computer Technology Institute (RACTI), Patras, Greece
[c] Computer Engineering and Informatics Department (CEID), University of Patras, Patras, Greece
[d] University of Lübeck, Institute of Telematics, Ratzeburger Allee 160, Lübeck, Germany

ARTICLE INFO

Article history:
Received 15 June 2010
Received in revised form
24 August 2010
Accepted 16 September 2010

Keywords:
Real-world internet
Experimentally-driven research
Sensor networks
Testbeds
Simulation tools
Hybrid tools
FRONTS

ABSTRACT

In this survey, we describe the state of the art for research on experimentally-driven research on networks of tiny artifacts. The main topics are existing and planned practical testbeds, software simulations, and hybrid approaches; in addition, we describe a number of current studies undertaken by the authors.

1. Introduction

Our modern world is largely ruled and shaped by decentralized mechanisms and processes: its breath-taking complexity has been possible not *despite* of the lack of centralized control, but largely *because* of it. This trend has been ever-accelerating because of technological advances, as computing devices keep getting smaller, more numerous, more powerful, ubiquitous—and better connected.

These advances offer new opportunities, but also new challenges. Innovative qualities of turning a collection of tiny

☆ This work has been partially supported by the ICT Programme of the European Union under contract number ICT-2008-215270 (FRONTS).

* *Corresponding author.* Tel.: +49 451 500 5383; fax: +49 451 500 5382.
E-mail addresses: t.baumgartner@tu-bs.de (T. Baumgartner), ichatz@cti.gr (I. Chatzigiannakis), s.fekete@tu-bs.de (S.P. Fekete), fischer@itm.uni-luebeck.de (S. Fischer), koninis@cti.gr (C. Koninis), a.kroeller@tu-bs.de (A. Kröller), krueger@itm.uni-luebeck.de (D. Krüger), mylonasg@cti.gr (G. Mylonas), pfisterer@itm.uni-luebeck.de (D. Pfisterer).

1574-0137/$ - see front matter
doi:10.1016/j.cosrev.2010.09.006

computing devices into networked *societies* are matched by novel issues of scalability. This requires a novel combination of theory and practice, i.e., a close interplay between algorithmic studies with experimentally-driven research, balancing theoretical methods with evaluation by simulations as well as real-world experimental facilities.

Currently, a lot of researching groups are working on technologies to form a Real-World Internet (RWI) where all kinds of (typically resource-constraint and wirelessly connected) devices will extend the Internet to the physical world. Application development for the RWI is complex as it unites the challenges of distributed applications and embedded programming. In addition, heterogeneity, unpredictable environmental influences and the size of the networks further complicate this situation. To master this complexity, simulations are a standard means for testing and optimizing applications in a controllable environment before the applications are tested in real networks of limited size. As Mark Weiser announced in the nineties [1],

"the research method for ubiquitous computing is standard experimental computer science: the construction of working prototypes of the necessary infrastructure in sufficient quantity to debug the viability of the systems in everyday use".

Hence, beyond simulation, experimentation is an equally important issue in realizing RWI services, protocols, and applications as also argued in [2]. Experimentation allows researchers to escape the inherent limitations of simulation regarding the available hardware characteristics (e.g., buffer sizes, available interrupts) and communication technology behavior (e.g., transmission rates, interference patterns). However, only a few researchers have compared testbed results with simulation ones as for example in [3–5].

In recent years, the need for a closer interaction between theory and practice has also become apparent within algorithmic research itself. Classical theoretical computer science has focused on abstract models, provable properties, and worst-case complexity. On the other hand, real-world problems require practical solutions; this makes it important to consider practically relevant models, algorithmic methods that can actually be evaluated, and a feedback into models and theory, leading to better results. This leads to a development cycle that is driven by falsifiable hypotheses and ties together theory and practice: realistic models lead to design and analysis of algorithms, their implementation makes it possible to perform experiments that are based on real inputs, which leads to better models and designs. This is the basic idea behind the new field of *Algorithm Engineering*, which has seen an increasing amount of attention, including the conference series "Workshop on Algorithm Engineering and Experiments" (ALENEX), "Symposium on Experimental Algorithms" (SEA), and "Workshop on Algorithm Engineering" (WAE), as well as a number of special research programs.

This focus shift in the theory community resembles similar developments in more practically oriented communities. The so-called *Experimentally-Driven Research* has been instrumental in advancing the state-of-the-art in RWI research. Here, an iterative approach to research and development is used. An algorithm, protocol, or application is implemented, simulated, and experimentally-validated. The data obtained from this stage then helps in optimizing and improving the original design. For this approach it is imperative that both simulation tools and experimental facilities of high quality are available and easy to use. In addition, the transition from simulation to experimentation must be straightforward or can even be used simultaneously. Such a hybrid approach allow users to choose any suitable point in the spectrum between simulation-only and experimentation-only.

In this survey, we describe the state of the art for research on experimentally-driven research on networks of tiny artifacts, i.e., algorithm engineering for distributed systems. The main topics are existing and planned practical testbeds (Section 2.1), software simulations (Section 2.2.), and hybrid approaches (Section 2.3). In addition, we describe a number of current studies undertaken by the authors, including the Wiselib (Section 2.4), and a variety of studies that have been conducted as part of the collaborative project FRONTS (Section 3), as well as particular case studies (Section 4).

2. Tools of practical assessment

Successful Algorithm Engineering for tiny artifacts requires the validation and evaluation of algorithms in experiments, preferably on networked embedded devices, or alternatively, in simulation. Such networks and simulators are therefore fundamental tools for the algorithm developer. In the following, Section 2.1 gives an overview of currently existing experimental facilities and Section 2.2 summarizes the state-of-the-art in simulation.

2.1. *Experimental facilities*

An overview of well-known discontinued, active, and planned testbeds is shown in Table 1. This section discusses properties of active testbeds, mentions some of their shortcomings, and briefly introduces planned or currently developed testbeds.

MoteLab

MoteLab [10] is an indoors testbed deployed on the campus of Harvard University that is open to the public. It is comprised of 190 Tmote Sky sensor nodes. It provides a web-based interface for re-programming and receiving data from sensor nodes and offers debugging capabilities. During the execution of uploaded programs, data coming from the sensor nodes are logged to a central database, which users can access after job completion. Nodes run the TinyOS operating system and are programmed in the nesC programming language.

TWIST

TWIST [11] resides indoors in a building in the campus of the Technical University of Berlin. The testbed spans across several floors and a total of 200 nodes of two different types (100 Tmote Sky and 100 EyesIFXv2) are available—thus heterogeneity is supported to some extent. It provides advanced control features with fault and event generation capabilities to the researchers-users. The testbed is open for job submissions to registered users.

Table 1 – Overview of well-known discontinued, active, and planned testbeds.

Name	Active	Nodes	Services/Architecture/Notes
Trio [6]	–	557 (solar-powered) trio motes	Over-the-air-programming of sensor nodes, routing, time synchronization, network management, and reliable data collection, 7 gateways. Primarily intended for multi-target tracking applications
Intel sensorNet [7]	–	100 micaZ sensor nodes	
Kansei [8]	–	210 sensor nodes	Extreme scale motes (XSM) with a 802.11 extreme scale stargates (XSS) gateway attached.
sMote [9]	–	78 Mica2DOT	Powered via power over ethernet, wired reprogramming of nodes. Replaced by the motescope testbed in April 2007.
Omega [9]	–	28 Telos motes (rev B)	Motes connected via USB for power, programming and debugging.
MoteLab [10]	×	190 Tmote Sky	Wired to programming boards allowing for direct reprogramming and communication.
TWIST [11]	×	100 Tmote Sky and 100 eyesIFXv2	3 tiers—servers, super nodes and sensor nodes. Wired connections form a testbed backbone, with most of them being USB connections.
TutorNet [12]	×	104 Tmote Sky	Each cluster of at most 7 nodes is attached to a stargate gateway.
Emulab / truemobile [13]	×	6 mobile robots + 25 fixed Mica2 motes.	Mobile robots equipped with Mica2 motes and intel stargates with 802.11. Reprogramming of nodes supported.
Motescope [9]	×	78 micaZ	Follow-on to the sMote testbed. Motes connected via USB for power, programming and debugging.
Oulu smart city [14]	×	IEEE 802.15.4 technology on the 868 MHz band	Multi-hop half-duplex IP connectivity using 6LoWPAN. access points route collected data to a GSN (Global Sensor Network) server that saves the data, refines it, and forwards it to the client. Smart city project to study urban computing, or the interaction between urban spaces, human, information technology and information.
Friedrichshafen [15]	×	PDAs, sensors, smartphone, etc.	Smart city project to evaluate prototypical services such as e-Learning, smart metering, e-Health, e-Tourism, etc.
Cambridge citysense [16]	×	100 802.11 nodes	Deployed across a city (light poles, private or public buildings). Node will consist of an embedded PC, 802.11a/b/g interface, and various sensors for monitoring weather conditions and air pollutants.
MetroSense—Bikenet [17]	×		Sensor nodes are mainly mobile, attached to people or vehicles. Emphasis on mobile aspects of sensor networks.
FRONTS	×	22 iSense nodes	See Section 3.1 for details.
sensLAB	×	1024 custom nodes	Includes both static and mobile sensors. Includes nodes with 868 MHz free MAC layer or IEEE 802.11 radio interface.
WISEBED	July 2010	>1000	Reprogramming, heterogeneity, federation of distributed laboratories, and virtual testbed technology.
SmartSantander	Project starts 10/2010	>15 000 (planned)	Large-scale future internet testbed facility project. City-scale experimental research facility in support of typical applications and services for a smart city.

TutorNet

TutorNet [12] is deployed inside a building in the campus of the University of Southern California and it currently comprises 13 clusters. Each cluster features a Stargate gateway station and several motes connected through a USB hub. Gateway stations communicate with the testbed

servers wirelessly over 802.11b and provide the capability to reprogram and interact with the sensor nodes. The total number of Tmote Sky nodes is currently 104. Authenticated users connect to the testbed servers and use command-line tools to control the testbed nodes.

MetroSense—Bikenet

MetroSense—Bikenet [17] emphasis is on mobile aspects of sensor networks. Although it is not yet a full-fledged sensor testbed like the others mentioned in this section, at the moment there exist some small testbeds like Bikenet. A 3-tier network architecture is also followed here, with testbed servers and gateway stations, but sensor nodes are mainly mobile, attached to people or vehicles. Regarding BikeNet, which features sensor nodes attached to monitor a small group of cyclists moving in the campus area of the Dartmouth College, a web-based interface is available to the public to monitor the situation inside the testbed and watch sensor readings.

TrueMobile

TrueMobile (Mobile Emulab wireless sensor net testbed, [13]) is an extension to the popular EmuLab wireless ad-hoc network testbed. The mobile testbed currently covers a total area of 60 m^2, and is situated indoors. The testbed includes six mobile robots and 25 fixed Mica2 motes with serial programming boards, 10 of which also have full sensor boards. The robots carry an Intel Stargate sensor gateway station running Linux, an 802.11 wireless card, and a Mica2 sensor node. Robot motion can be scripted using a specific scripting language developed for the EMuLab or can interactively controlled from a Java applet setting starting and ending points on the testbed's floorplan (i.e., two-dimensional moves only). EmuLab is an open testbed to the public (registered users).

sensLAB

The *sensLAB* [18] platform will be distributed among 4 sites and will be composed of 1024 nodes. Each location will host 256 sensor nodes with specific characteristics in order to offer a wide spectrum of possibilities and heterogeneity. The four test beds will however be part of a common global testbed as several nodes will have global connectivity such that it will be possible to experiment a given application on all 1 K sensors at the same time.

WISEBED

The WISEBED experimental facility consists of a number of independent sensor networks located at 9 locations throughout Europe. A key characteristic is that its underlying physical infrastructure adopts a *federated* architecture that incorporates physical equipment from a number of different sites across Europe. It offers a single sign-on approach to provide researchers with access to this large federation of distributed WSN resources of up to 2000 heterogeneous sensor nodes including iSense, Telos B, Pacemate, Sun SPOT, Tmote Sky, and G-Node nodes. Users can freely configure a virtual topology on top of the physical topology by configuring so-called virtual-links [19] between nodes located in different testbeds. To master the heterogeneity during application development WISEBED provides a hardware-independent, generic algorithms library (the Wiselib, [20]) allowing one to write code that is resolved and bound at compile time, resulting in virtually no memory or computation overhead at run time.

Smart cities

Apart from the general-purpose testbeds mentioned above, there are a few initiatives to create smart city environments. Notable examples are Oulu in Finland [14], Cambridge, Massachusetts [16], Friedrichshafen, Germany [15], or Santander (the EU FP7 project SmartSantander starts by the end of 2010).

2.2. *Software simulators*

This section presents a selection of important simulation tools widely used by the community. The crucial point of these simulation tools is that each of them has its area of expertise in which it excels. There is no single, all-purpose, tool which is suitable in all conditions or applicable to all possible research questions (see Table 2 for a summary).

Ns-2

The *Network Simulator-2* (Ns-2, [21]) is a general-purpose, discrete event simulator targeted at network research. Nowadays, Ns-2 is the most prominent network simulator used in WSN research [30]. It focuses on the simulation of ISO/OSI layers including energy consumption and phenomena on the physical layer. Ns-2 includes a vast repository of protocols, traffic generators and tools to simulate TCP, routing and multicast protocols over wired and wireless networks. It features detailed simulation tracing and includes the visualization tool *network animator* (nam) for later playback of the observed traffic. Special support for sensor network simulations has also been integrated [31,32], including sensor models, battery models, lightweight protocol stacks and scenario generation tools. The highly detailed packet level simulations lead to a runtime behavior closely coupled with the number of packets being exchanged, making it nearly impossible to simulate large networks. Ns-2 is capable of handling up to 16,000 nodes but the detail level of its simulations render working with more than 1000 nodes virtually impossible in terms of runtime and memory consumption.

JSim

J-Sim [22] is an open source simulation and emulation environment for wireless sensor networks, fully implemented in Java. The framework provides a component definition of target, sensor and sink nodes, sensor and wireless communication channels, and physical media such as seismic channels, mobility models and power models (both energy-producing and energy-consuming components). Customized application-specific models can be readily defined and implemented by sub-classing appropriate classes of the simulation framework and customizing their behaviors. J-Sim also supports cross-layered approaches using autonomous component architecture by appropriately connecting the ports of a protocol class to those of another protocol class (not immediately above/beneath the former protocol class). J-Sim provides a script interface that allows its integration with different script languages such as Perl, Tcl, or Python. In

Table 2 – Overview of simulators for networks of tiny artifacts.

Name	Language	Size (# Nodes)	Notes
Ns-2 [21]	C/Tcl	10^3	Simulation of ISO/OSI layers including energy consumption and phenomena on the physical layer.
JSim [22]	Java	10^3	Feature rich definitions sensor network specifics (control center, sensor channels, seismic channels, mobility models, power models).
OMNeT++ [23]	C++	10^2	Component architecture provides very flexible composition of simulations.
SENSE [24]	C++	10^3	Specifically developed for the simulation of sensor networks. Offers different battery models, simple network and application layers and an IEEE 802.11 implementation.
TOSSIM [25]	nesC	10^2	Specifically developed for the simulation of sensor networks executing TinyOS.
Avrora [26]	Java	10^2	Provides a cycle-accurate simulation of device and communication behavior. Provides a highly accurate energy model for power profiling and lifetime prediction of WSN.
WSim [27]	C++	10^2	Provides a cycle-accurate simulation of device and communication behavior. Allows debugging and evaluating performances of the full system at the assembly level.
Shawn [28,29]	C++	10^6	Simulates the effect caused by a phenomenon, not the phenomenon itself. This leads to very fast simulation of large-scale networks.

particular, the latest release of J-Sim (version 1.3) has been fully integrated with a Java implementation of a Tcl interpreter, called Jacl, with the Tcl/Java extension. Therefore, similar to ns-2, J-Sim is a dual-language simulation environment in which classes are written in Java (for ns-2 in C++) and "glued" together using Tcl/Java. However, unlike ns-2, classes/methods/fields in Java need not be explicitly exported in order to be accessed in the Tcl environment. Instead, all the public classes/methods/fields in Java can be accessed (naturally) in the Tcl environment.

OMNeT++

The *Objective Modular Network Testbed in* C++ (OMNeT++, [23]) is an object-oriented, modular discrete event simulator. It is very similar to Ns-2 and also targets the ISO/OSI model. It can handle a few thousands of nodes and features a graphical network editor as well as a visualizer for the network and the data flow. The simulator is written in C++ and comes with a homegrown configuration language called *NED*. OMNeT's main objective is to provide a component architecture through which simulations can be composed very flexible. Components are programmed in C++ and then assembled into larger components using NED. It is free for academic use only and a commercial license is available.

SENSE

The *Sensor Network Simulator and Emulator* [24] is a simulator specifically developed for the simulation of sensor networks. The authors mention extensibility and reusability as the key factors they address with SENSE. Extensibility is tackled by avoiding a tight coupling of objects through a *component-port* model, which removes the interdependency of objects that is often found in object-oriented architectures. This is achieved by their proposed simulation component classifications, which are essentially interfaces, enabling the exchange of implementations without the need to change the actual code. SENSE offers different battery models, simple network and application layers and an IEEE 802.11 [33] implementation. In its current version, it provides a sequential simulation engine that can cope with around 5000 nodes. Depending on the communication pattern of the network, this number may drop to 500. The authors plan to support parallelization of the simulations to increase the overall performance.

TOSSIM

The mote simulator [25] emulates TinyOS [34–36] motes down to the bit level and is hence a platform specific simulator. It compiles code written for TinyOS to an executable file that can be run on standard PC equipment. It ships with a GUI (TinyViz), which can visualize and interact with running simulations. PowerTOSSIM [37], a power modeling extension has been integrated into TOSSIM. PowerTOSSIM models the power consumed by TinyOS applications and includes a detailed model of the power consumption of the Mica2 [38] motes. Using this technique, developers can test TinyOS applications without deploying them on real sensor network hardware. TOSSIM can handle scenarios with a few thousand virtual TinyOS nodes.

Avrora

Avrora [26] is an open-source simulator for embedded sensing programs. The current release is written in Java. Avrora is scalable like TOSSIM. The event-queue model for cycle-accurate simulation of device and communication behavior allows improved interpreter performance and enables an essential sleep optimization. A highly accurate energy model is available, enabling power profiling and lifetime prediction of sensor networks. Avrora can emulate two typical platforms sensors, namely Mica2 and MicaZ.

WSim

WSim [27] is a cycle accurate full platform simulator using microprocessor instruction driven timings. WSim is able to perform a full simulation of hardware events that occur in the platform and to give back to the developer a precise timing analysis of the simulated software. The native software of the node can be used in the simulator without the need to reconfigure or recompile the software. Wsim uses a classical cross-compiler tool-chain, so that the simulation is neither attached to any particular language nor operating system.

WSim thus allows debugging and evaluating performances of the full system at the assembly level. A precise estimation of timings, memory consumption and power can be obtained during the simulation.

Shawn

Shawn [28,29] is an open-source discrete event simulator. It does not compete with the above-mentioned simulators in the area of network stack simulation but focuses on algorithms, large-scale networks (up to 10^6 nodes), and speed. One central approach of Shawn is to simulate the effect caused by a phenomenon, not the phenomenon itself. For example, instead of simulating a complete MAC layer including the radio propagation model, only the effects (i.e., packet loss and corruption) are modeled in Shawn. This has several implications for simulations: they get more predictable and meaningful, and there is a huge performance gain, because such a model can often be implemented very efficiently. This also results in the inability to come up with the detail level that, say, Ns-2 provides with respect to physical layer or packet level phenomena. The authors argue that it makes sense to simplify the structure of some low-level parameters since their time-consuming computation can be replaced by fast simulations, as long as the interest in the large-scale behavior of the macro-system focuses on unaffected properties.

2.3. *Hybrid approaches*

In Real-World Internet applications several types of networks' tiny artifacts will coexist in the same structure alongside traditional processors. For example, embedded wired sensors and wireless sensor nodes will provide complementary functionality for Internet servers to process and present. These differences also imply different capabilities, operational range and requirements in resources, thus may also require applying different protocols in order to be more efficient. Developing applications to run on such a diverse set of devices, requires coordinating multiple languages, especially very different ones like hardware and software languages. Additionally, implementing communication mechanisms between different devices unnecessarily increases development time. Simulating such a system is complicated because of the need to coordinate compilers and simulators, often with very different interfaces, options, and fidelities. Recently, a small number of *hybrid approaches* have been made that combine physical elements with simulation and emulation. A hybrid approach refers to the interaction of the simulation with external entities such as real network nodes, user applications or some other external source of input/output to a running simulation. A hybrid approach has some similarities with network emulation.

SensorSim

SensorSim [32] is a simulation framework that introduces new models and techniques for the design and analysis of sensor networks. SensorSim inherits the core features of traditional event driven network simulators, and builds up new features that include the ability to model power usage in sensor nodes, a hybrid simulation that allows the interaction of real and simulated nodes, new communication protocols and real time user interaction with a graphical data display.

H-TOSSIM

H-TOSSIM [39] attempts to bridge the gap between simulation and real-world performance by exploiting the TinyOS programming model to generate discrete-event simulations directly from TinyOS component graphs. A hybrid testbed is proposed, which extends TOSSIM with physical nodes. The resulting platform is primarily a simulator with a small number of attached physical nodes. H-TOSSIM uses three physical nodes, of which, one shares the simulated environment with all virtual nodes to test the WSN program, and the other two bridge the real world and the simulated environment. H-TOSSIM combines the advantages of both the simulation in physical node and the simulation testing tools in WSN software testing. The level of detail provided by this simulation tool is resource-demanding and limits simulations to small scenarios.

Emstar

Emstar [40,41] is a software environment for developing and deploying complex applications of heterogeneous sensor network systems that are based on TinyOS. Emstar is designed to leverage the additional resources of "microservers" (a general term for platforms like Stargate) by trading off some performance for system robustness in sensor network applications. It enables fault isolation, fault tolerance, system visibility, in-field debugging, and resource sharing across multiple applications. This framework uses TOSSIM to emulate motes and EmStar [42] to emulate "microservers". A wrapper library is employed to glue the two simulation systems together. All applications must be recompiled and linked to the EmStar library if they are to be emulated by the system.

A similar system is presented in [43], that simulates a heterogeneous network comprised of simple (base level) sensor devices that are compatible with TinyOS [35] and more powerful Intel Stargate devices [44] that are used as intermediate gateways.

X-Sim

X-Sim [45] is a simulation environment that is part of the larger Auto-Pipe system. Auto-Pipe is a performance-oriented development environment for heterogeneous systems. It concentrates on streaming applications placed on pipelined architectures. Auto-Pipe applications are expressed using a data flow coordination language. Individual computational tasks can be implemented in various languages for any subset of the available platforms (e.g., C/C++ for general-purpose processors, HDL for FPGAs, assembly language for network processors and DSPs). The applications may then be compiled and mapped on to complex sets of devices, simulated, and optimized.

Sensornet Checkpointing

Sensornet Checkpointing [46] is a drastically different approach for simplifying migration from simulation to testbed. When developing sensor network applications, the shift from simulation to testbed causes application failures, resulting in additional time-consuming iterations between simulation and testbed.

In Sensornet Checkpointing every node exists both in testbed and simulation. However, the entire network is at any

given time executing only in either simulation or testbed. A checkpoint transfers the sensor network's state between simulation and testbed. By transferring the network's state to simulation, one can benefit from the inherent properties of simulation to non-intrusively extract network details. The developer can deterministically repeat execution in simulation, or repeatedly roll back a single network state to testbed and benefit from the realism of real hardware.

Virtual Links

Virtual Links [19] can be used to federate physically separate sensor networks into larger virtualized ones. A virtual link is established between two nodes in the same or different networks. It works using "virtual radio" interfaces. Whenever one node sends a message to its neighborhood, the virtual radio interface will duplicate it and send one copy to the other node of the link, where it will be processed as if being received through the radio. Thereby, the two nodes of a virtual link become neighbors, resulting in a federation of two networks.

There are several options on how messages are transmitted through the link, usually involving two gateways between the sensor network and the Internet. It is favorable to employ out-of-band communication between sensor nodes and gateway, to avoid having undesired effects on the regular in-network communication. This can be done using a wired backbone, which is already present in many WSN testbeds to ease maintenance and data collection.

Using a number of virtual links, two or more networks can be federated. The application code on the nodes will simply see a single network. This allows for a number of scenarios where virtualized topologies can benefit the algorithm developer. For instance, one can join networks with different hardware characteristics or environmental influences, to test the interoperability of an implementation. Another extremely useful application is to establish virtual links between a real network and nodes in a simulator. This allows tests in huge networks. The developer can evaluate the physical nodes, which do actual wireless communication, suffer from interference and from hardware and software limitations. The simulated nodes allow to assess scalability of the algorithm. The user can therefore enter the next iteration in algorithm development without requiring access to a full-size network. It even allows one to evaluate the feasibility of a network's deployment, by physically installing only a small fraction of the nodes and leaving the largest part in simulation. So, one can use algorithm feedback to change the installation plan before expensive investments are made.

2.4. *The wiselib: a library of algorithms*

Algorithm Engineering requires the algorithm developer to actually implement algorithms. This step from theory into practice is often considered hard and requires programming skills in addition to knowledge in algorithm theory. This also explains why many theorists, having only little Software Engineering experience, never engage in Algorithm Engineering.

This situation is particularly alarming in distributed embedded systems. There are several complex issues in addition to those present in classic Algorithm Engineering:

Lack of established programming standards

Many algorithm researchers know at least one of a few standard programming languages—usually C, C++, or Java. When an algorithm is to be implemented, one can usually choose a language just by personal taste, unless third-party libraries are involved. Algorithm implementations are often independent of the OS, and therefore usually portable. All of this does not apply to embedded systems. Here, the hardware dictates what firmware, and thus language, one can use. This may even involve learning a custom programming language, such as nesC, if one chooses to use the popular TinyOS [34].

Compatibility issues

Hardware and OS choice define tight programming constraints. Given that not every OS runs on every device, this means that an implementation is generally compatible with a single or a few device classes only. This makes comparative implementation evaluations hard, whenever the implementations are not compatible with a single common platform.

Absence of low-level convenience features

Embedded devices are often heavily constrained. Standard operations are often missing. Often, devices even lack support for dynamic memory allocations (`malloc()` resp. `new()`), meaning that standard algorithmic techniques, such as using efficient dynamic data structures, are infeasible.

Absence of algorithm building blocks

The worst issue comes from the fact that algorithm composition from standard blocks is impossible, if the said blocks are not available for a given system. The lack of dynamic memory usually implies that standard data structures (trees, queues, lists, etc.) are not available. This also applies to basic algorithms, such as sort and binary search. Often, algorithms use other algorithms from the literature as subroutines. This requires the developer to spend substantial amounts of time in re-implementing standard routines for the platform at hand, slowing down the Algorithm Engineering cycle considerably.

One way to remedy this situation is to restrict oneself to using a single generic operating system, such as TinyOS [34] or Contiki [47]. This has the benefit that, over time, more implementations become available and some of the aforementioned issues diminish. However, two drawbacks remain: First, one is locked into one scheme, unable to use code from other systems. Second, one has to re-learn programming, if the system uses a custom programming language like nesC.

Another way is offered by the *Wiselib* [20]. It addresses all of the above issues in a standard, algorithmist-friendly way. It is a code library, that allows implementations to be OS-independent. They can be recompiled for several platforms and firmwares, without the need to change the algorithm code. It can interface with systems requiring different languages, demonstrated for C (Contiki), C++ (iSense), and nesC (TinyOS). Furthermore, it also runs on the simulator Shawn [29], thereby easing the transition from simulation to actual devices.

It is written in a well-defined subset of C++, briefly summarized as C++ including templates, but without virtual inheritance and exceptions. Developers skilled in C++ can implement code for embedded devices using the Wiselib, without having to deal with too many low-level issues.

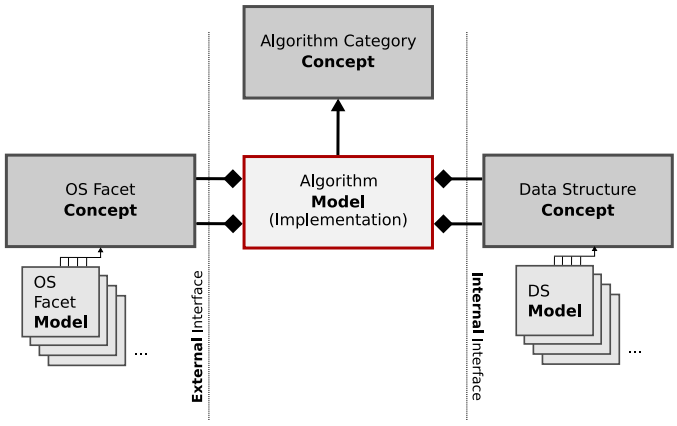


Fig. 1 – Wiselib architecture.

Architecture

Code in the Wiselib is based on *concepts* and *models*. These are used in the same way as with other template-based code libraries for desktop systems, such as Boost [48] or CGAL [49]. See Fig. 1. The underlying OS is abstracted in so-called *OS facets*, which offer a consistent interface to low-level OS functions. Algorithms and data structures are implementations of predefined interfaces (algorithm concepts resp. data structure concepts). The developer can base his work on these interfaces, and later choose the actual implementations to be used. The compiler will produce highly efficient code for a given platform, as template-based architectures expose full implementation details to the compiler's optimization routines. This means that the price of generality will be paid at compile-time, but not at runtime.

pSTL

The Wiselib offers a stripped-down version of the STL, providing standard data structures and algorithms written for constrained devices with or without dynamic memory. This allows the algorithm developer to focus on higher-level issues.

pMP

Similarly, for cryptographic algorithms, a generic implementation of big integers is available. It currently supports only a few operations, but is expected to grow in the near future.

Algorithms

At the core of the Wiselib, there are interface definitions for the algorithms that are typically researched in networks of tiny artifacts. For each algorithm class, several implementations are already available. These classes are

1. Localization,
2. Clustering,
3. Coloring,
4. Cryptography,
5. Routing,
6. Time Synchronization,
7. Topology Control, and
8. Tracking.

As of mid 2010, the Wiselib includes about 40 Open Source implementations of standard algorithms, and is scheduled to grow to 150–200 algorithms by the end of 2011. Using the generic algorithm interfaces, an algorithm developer can consider all of them as building blocks for his own implementations, and can later replace one implementation by another without the need to change his own code.

To summarize, the Wiselib offers several interesting tools to the Algorithm Engineer. It does not make development as easy as on standard desktop systems, but it comes close. The burden of having to deal with low-level platform issues is lifted, and the developer can focus on actually researching algorithms.

3. Algorithm engineering in FRONTS

Most of the time in computer science, researchers tend to design an algorithm in an abstract way. This happens because an algorithm is a model that should be able to be used in many different situations and it is up to the developer to decide the way it should be turned into code for a real system. However, most state-of-the-art foundational approaches are often too abstract, missing a satisfactory accuracy of important technological details and specifications. Almost every time the developer finds many limitations in the ways he can operate within the given hardware and software specifications that are posed by the given platform.

Project FRONTS follows a multifaceted approach and by integrating algorithms, simulations and even real world experiments, the project attempts to come up with easily implementable solutions that still satisfy the critical quality of service guarantees. In this section we describe the equipment and the structure of the FRONTS testbed as well as software tools for a proper algorithm engineering using the testbed.

3.1. *Hardware*

The FRONTS testbed consists currently of 22 iSense sensor nodes with a 32 Bit RISC Controller running at 16 MHz and an IEEE 802.15.4 compliant wireless radio interface

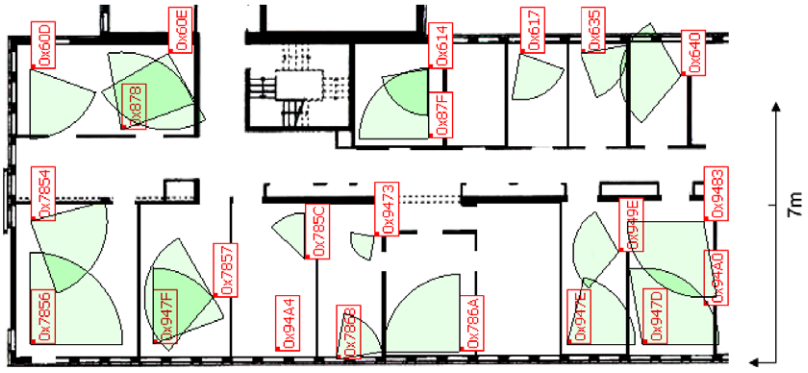


Fig. 2 – Positions and ranges of the sensors.

including hardware AES encryption and SMA antenna. All nodes come with 96 KB RAM and 128 KB Flash of shared memory for program and data as well as a permanent energy source allowing for an arbitrary duty cycle. 20 of them are additionally equipped with a Security Module comprising a passive infrared sensor and an accelerometer. The former is capable of detecting moving objects in a 110° angle within a range of up to 10 m depending on temperature difference, speed and size of the objects. The latter 3-axis accelerometer can be configured to cover accelerations of ±2 g or ±6 g or to generate interrupts on movement, direction change or free fall.

The other 2 nodes – connected to a PC – are remotely accessible via the management and visualization tool *iShell*.

Fig. 2 shows the sensor node's positions – indicated by the rectangles and IDs – within the Institute of Telematics at the University of Lübeck. They are distributed over the whole institute in order to obtain the greatest possible network diameter. Nevertheless, we verified that the resulting density remains reasonably high. The green cones indicate the sensing range of the PIR sensors.

iShell does not only provide functionality to program the network and to exchange messages with the connected node, but also allows for the integration of application specific plug-ins, e.g. to configure, control and execute experiments automatically which greatly speeds up the engineering process. Moreover, visualizing results such as routing paths, helps to understand unexpected effects.

A remotely accessible testbed which can permanently run is a comfortable means to verify algorithms and sensor network applications.

3.2. *Simulating the testbed*

A typical use-case for simulations is optimizing applications before they are deployed on hardware. Hence, a primary goal of our testbed is to allow simulations of application before it is deployed on real hardware. Simulations allow both verification and optimization of algorithms, especially in larger or denser scenarios to test the scalability of an algorithm. However, as simulations may differ significantly from real-world results and the complexity of detailed simulations of real-world effects impedes productive development, it is of major importance that simulations support different abstraction levels. Reducing the complexity allows for concentrating on single problems and clarifies the influence of each parameter. With increasing comprehension the degree of faithfulness to reality must be stepped up. Due to the model architecture of Shawn the abstraction degree can be adjusted in a fine-grained way.

In Fronts, Shawn is used as the primary simulation tool which supports using the same code for the pacemate, TelosB, and iSense sensor nodes as well as for the simulation tool thus relieving developers from a repeated implementation of the application for different sensor nodes or the simulator. To simulate the testbed scenario we created a representation of the node positions within the institute. Herewith, experiments running on the testbed can directly be reproduced in Shawn as iSense software runs on both devices and in Shawn.

Apart from reproducing the topology, this includes modeling aspects of communication, i.e., choosing the best communication model in the simulator and parameterize it such that it provides a good approximation of real data transmissions. Using Shawn, a frequently used model is the so-called *stochastic communication model* which honors the fact that the probability of a successful reception diminishes with increasing distance. It defines two distances r_1 and r_2 with $r_1 < r_2$. For $0 < d < r_1$ a constant probability $p_{\max}$ is assumed while for $d > r_2$ no communication is possible. For $r_1 \leq d \leq r_2$, the packet reception probability decreases linearly from $p_{\max}$ to 0. We have derived parameterizations for this model from a set of measurements in different scenarios (e.g., indoor, outdoor, different antennae and heights above the ground). Fig. 3 shows an example where the values $r_1 = 17.6, r_2 = 25.2$, and $p_{\max} = 0.96$ were derived by averaging a set of indoor measurements using iSense sensor nodes with an IEEE 802.15.4 compliant radio interface.

While the stochastic communication model already improves the realism of Shawn simulations compared to earlier models (e.g., the unit disk graph model), this model is still a substantial abstraction from reality. The packet delivery ratio over distance may vary for each pair of nodes depending on the actual environment (e.g., walls that impede communication or different heights above ground). To better approximate reality for a given deployment, a promising option is to use measurements from this deployment to parameterize a communication model for each pair of nodes

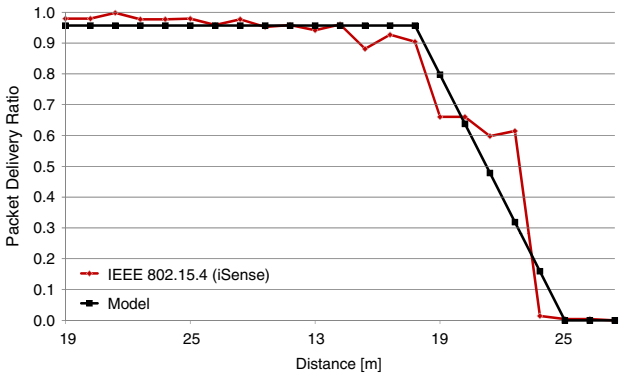


Fig. 3 – Parameterization for Shawn's stochastic communication model derived from real-world measurements.

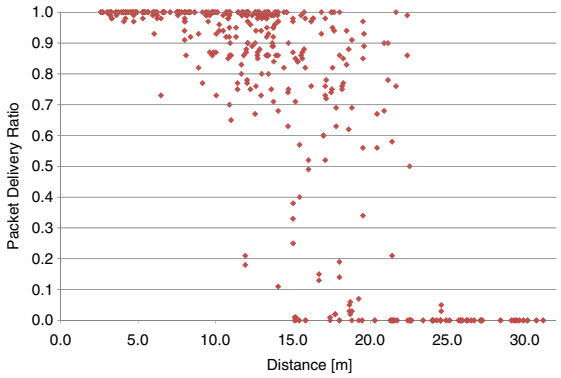


Fig. 4 – Packet delivery ratio measured a testbed of 22 nodes.

individually. Therefore, we implemented a communication model for Shawn which accepts packet delivery probabilities for each link between two nodes.

To parameterize this model, we measured the packet delivery ratio for each individual link in our testbed and logged the data to a file. We have implemented an application that identifies receiving rates and RSSI values for each pair of nodes in a network to build a realistic communication model. Since these values potentially vary due to temperature fluctuations, moving objects and WLAN activity we logged them over several days and in different weeks of the year. Each node sent a hello message every 20 s, which corresponds to about one message per second on a global level, causing nearly no collisions. The receivers of a message store the link quality indicator and increment a receive counter for the sender. Once per hour each node sends the mean values for all neighbors to the sink node which writes them into a file. From this file, packet delivery probabilities for each unidirectional link are calculated and written to a Shawn configuration file. Fig. 4 visualizes the measurements for the 22 nodes. Comparing this plot to Fig. 3, it is obvious that the stochastic model can only provide a very rough approximation of the prevailing packet delivery ratios in the real testbed—which is especially true in indoor deployments.

Shawn also provides a motion event generator for simulating moving objects which are detected by the PIR sensors. Furthermore, Shawn can be configured to execute simulations in realtime and by connecting Shawn to the management tool iShell, it looks like a real sensor network to the application programmer except that it is possible to visualize debug output of all nodes and not only of the connected gateways.

3.3. *Experiments repository*

FRONTS is a project involving 11 partner sites across Europe. In order to better coordinate the activities of the partners, a central Experiments Repository of algorithms and experiments has been developed. A central goal of the repository is to provide a pool of implemented algorithmic solutions that can be used as building blocks for developing systems and applications for networks of tiny artifacts. Its purpose is to archive the implemented software and the corresponding documentation in a central place. Furthermore, the repository attempts to facilitate the reuse of existing code, transfer the results and enforce a productive collaboration between the sites by enabling the different sites to compare new solutions with old ones, evaluate existing solutions in new models, and evaluate existing solutions under new performance measures.

Currently the Experiments Repository maintains 25 software components that are implemented following three different approaches: (A) in specific simulation platforms (e.g., ns-2, matlab, OMNeT++) or specific hardware platforms (e.g., SUN SPOT, TinyOS) or stand-alone software (e.g., C++, Java, Python, Ruby); (B) compatible with the centrally coordinated and experimentation test bed (as described in the previous sections). The software components include solutions for adaptive & secure routing, window streaming algorithms for data aggregation, self-synchronized duty-cycling, load and communication balancing, target tracking, robot coordination, privacy for RFID, power-aware online file allocation, self-organization of the infrastructure and duplicate insensitive counting.

4. Case studies

In order to evaluate the proposed methodology we will present two case studies involving protocols developed under FRONTS project. The development process for the first case study follows a well-defined set of individual phases that forms a cyclic model of Algorithm Engineering. It starts from the formal theoretical analysis of the algorithm that provides a solid foundation. Then continues with the implementation of the protocol and then with moving to simulations and experiments that tests its behavior under diverse scenarios. The empirical results gathered from the last phases provide insights into the inner workings of the protocol, that help identify the design flaws and the beneficial elements. This process leads back in the previous phases in order to modify parts of the algorithm or to expand and revise the theory. This Algorithm Engineering lifecycle is depicted in Fig. 5.

In the second case study we examine a problem where designing a fully distributed protocol is not straightforward.

It starts by performing a structural analysis of the problem at hand and by designing a simple and correct centralized algorithm. The next step is the implementation of the centralized algorithm in order to achieve a rapid prototype version. The centralized implementation has full access to all nodes and has a global, flat view of the network. This provides a simple means to obtain results and get a first impression of the overall performance of the algorithm. The results emerging from this process provide optimization feedback for the algorithm design. After archiving a satisfactory state of the centralized version, the next step is to implement a simplified decentralized protocol. This simplified version is transformed gradually into a protocol that actually exchanges network messages containing protocol payload. With the protocol and data structures in place, the performance of the distributed implementation can be evaluated. This development cycle helps the developer to start from an initial idea and gradually leads to a fully distributed protocol with a expanded and revised theory. This protocol development lifecycle is depicted in Fig. 7.

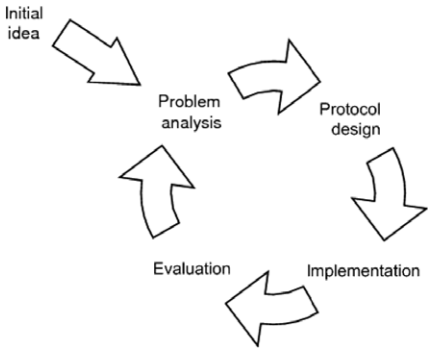


Fig. 5 – Algorithm engineering lifecycle.

4.1. *Adaptive multipath fair communication*

The subject of the first case study is communication protocols for static, homogeneous Wireless Sensor Networks (WSN's), where the energy reserves of the tiny artifacts are finite. A protocol's goal should be to prolong the system's *functional life* and maximize the number of events which get successfully reported. The protocol has to be decentralized and adaptive to current conditions.

There is a great variety of different protocols for WSN's for data aggregation. A well known data aggregation paradigm is *Directed Diffusion* [50] whose key element is data propagation through multiple paths. In this *multi-path* data propagation, information is *broadcast* to more than one node in each single hop, *without*, however, *flooding* the network. The receiving nodes continue to broadcast until the message reaches its final destination. In this approach, that trade-offs of energy consumption and fault-tolerance, and node failures do not need to be handled in a special way since a single successful delivery is enough and the message is always propagated simultaneously to several nodes. Thus protocols are less affected by node failures and ensure successful message deliveries. Additionally they are easier to implement since they do not involve a route discovery process. On the other hand, they tend to spend more energy than the single-path protocols since they engage more sensor devices to the propagation process than the single-path ones.

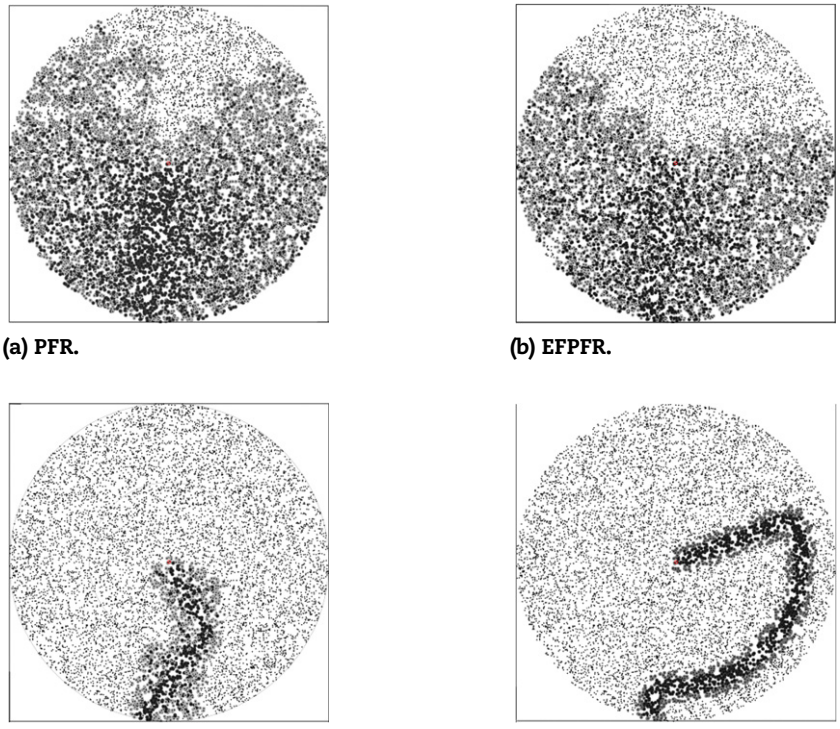
The starting point of this case study is the rigorously analyzed *Probabilistic Forwarding Protocol*—PFR [51]. It is a lightweight, probabilistic and totaly distributed algorithm. PFR is thoroughly analyzed considering correctness, energy efficiency and robustness. The first step was to implement PFR and test it on diverse operation scenarios. The series of experiments revealed a series of design flaws that were not evident in the theoretical analysis. Then, based on the original protocol, the next step was to design new protocols by modifying specific components of PFR. The resulting protocols were again experimentally assessed and the impact of the modified components was evaluated. The process yielded interesting ideas which were simulated and evaluated as well. Corrections were suggested and experiments took place to identify the best setting of their parameters. From this procedure protocols EFPFR and MPFR emerged as descendants of PFR, and protocol TWIST evolved, in turn, from MPFR.

Probabilistic data forwarding

PFR is *a multi-path* protocol based only on local information. It succeeds in efficiently propagating information to the control center without flooding the network (although each transmitting device broadcasts around it). This is done by probabilistically favoring certain close to optimal "paths" towards the control center and also by avoiding any control messages.

The basic idea of the protocol lies in probabilistically favoring transmissions towards the control center within a *thin zone* of sensor devices around the line connecting the sensor device sensing the event $\mathcal{E}$ and the control center. Note that transmission along this line is energy optimal. However it is not always possible to achieve this optimality, basically because certain sensors on this direct line might be inactive, either permanently (because their energy has been exhausted) or temporarily (because these sensors might enter a sleeping mode to save energy). Further reasons include (a) *physical damage* of sensors, (b) *deliberate removal* of some of them (possibly by an adversary in military applications), (c) *changes in the position* of the sensors due to a variety of reasons (weather conditions, human interaction etc.). and (d) *physical obstacles* blocking communication. The protocol evolves in two phases:

> The "Front" Creation Phase. When a new event $\mathcal{E}$ is detected by a sensor device, the *Front Creation* phase is initiated and it lasts for β rounds. During the Front Creation phase every sensor device w receiving the message deterministically forwards it. The effect of this phase is that a sufficiently large front is built, to ensure that the data propagation process survives for an appropriately large period of time.
>
> The Probabilistic Forwarding Phase. After the Front Creation phase data propagation is done in a probabilistic manner. When a sensor device v receives a new message m, it decides to propagate based on the so called angle ϕ. ϕ is the angle formed between the sensor A that original



(a) PFR. (b) EFPFR. (c) MPFR. (d) TWIST.

Fig. 6 – Single message's propagation according to the four protocols. Control center's position is at the center of the disk, while the message's source device is at disk's circumference. The light grey spots stand for devices having received the message, while the dark ones for devices having received and transmitted it.

generated the message, sensor device v and the control center S ($\phi = \widehat{AvS}$). Based on ϕ, device v broadcasts the message to all its neighbors with probability $\Pr_{fwd}$ (or it does not propagate any data with probability $1 - \Pr_{fwd}$) defined as follows:

$$\Pr_{fwd} = \begin{cases} 1 & \text{if } \phi \geq \phi_{\text{threshold}} \\ \frac{\phi}{\pi} & \text{otherwise} \end{cases}$$

where $\phi_{\text{threshold}} = 134°$ (the selection reasons of this $\phi_{\text{threshold}}$ is a result of the theoretical analysis, [51]).

The analysis conducted in [51] shows that PFR successfully propagates messages to the control center if the whole network is operational (see Lemma 1, [51]). The energy efficiency of PFR is $\Theta\left(\left(\frac{n_0}{n}\right)^2\right)$ where $n_0 = |AS|$ and $n = \sqrt{N}$ (see Theorem 2, [51]). For $n_0 = |AS| = \mathcal{O}(n)$, this is $\mathcal{O}(1)$. Finally, in terms of robustness, it is shown that even in cases where sensor devices that are positioned on the "optimal" (or nearby) fail, PFR still manages to propagate the crucial data across lines parallel with AS, and of constant distance, with *fixed* nonzero probability, not depending on n or $|AS|$ (see Lemma 3, [51]).

The theoretical analysis conducted in [51] examines the general case where nodes are homogeneous and there is no distinction on the basis of energy consumption. As expected, results from the simulations conducted in [52] revealed serious drawbacks when considering such issues. Fig. 6(a) illustrates the propagation of a single message; it clearly demonstrates the multiple paths behavior of the protocol. In order to increase the probability of successful delivery PFR eventually activates the vast majority of the network's sensor devices. Inevitably this leads to the dissipation of the energy resources of the devices. Therefore PFR needs to be modified and extended to fix this deficiency in order to maximize the lifetime of the network. Since it is very difficult to deal with such issues via a theoretical analysis (introduction of energy levels at device level significantly increases the complexity of the theoretical model and the tractability of the analysis), an Algorithm Engineering approach is followed.

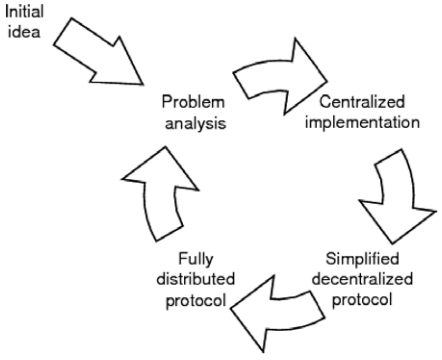


Fig. 7 – The protocol development model.

Energy fairness

The EFPFR protocol is the first attempt to solve the previous problem and it is based on the observation that PFR's propagation has a strong flooding behavior thus it is

important to limit the number of broadcasts. EFPFR has similar phases with PFR but takes into account the energy levels of the sensor devices. More specifically every node periodically broadcasts information about its energy level. This information is used during the Probabilistic Forwarding Phase in order to modify the transmission probability according to local energy ranking of the sensor device. So the sensor devices that have more energy are given higher probability to forward a message than those with lower energy. This design concept is also known as *Fairness*; in the context of WSN's, has been addressed by many researchers in the past (e.g. [53]).

The term implies that the protocols must be able to identify each sensor device's particularities and roles and consequently treat them respectively. For instance important nodes should be relieved from some of their burden, while the seemingly less important ones, should be prompted to participate more in the data propagation process. This could be realized either by prohibiting transmissions from the frequently used, "poor", in terms of energy sensor devices, or by encouraging propagations through paths different from the greedy, shorter ones. The goal is to amplify "solidarity" within the network and keep it functional for more time.

Experimental work [52] shows that EFPFR improves PFR performance because it reduces the unnecessary message transmissions and it can adapt to different network conditions. However the simulations showed that flooding is not avoided since in a dense network a few transmissions are required to propagate data everywhere. This is depicted in Fig. 6(b).

Multiple forwarding phases

By observing the shortcomings of EFPFR, a different approach was needed in order to achieve a more fair energy dissipation. The new designed protocol employed a novel solution, that is to call PFR in multiple phases in order to form the indirect, crooked transmission path. The benefit is that traffic will be diverted from the sensor devices on the "optimal line" and the network will be functional for more time. MPFR divides the network area using concentric rings, with the center being the control center. A message is forwarded from each outer circle to the inner one using PFR. MPFR results showed a significant improvement over the EFPFR in delivery rate and in the number of alive sensor devices as we can see in Fig. 6(c).

Twisting forwarding

Careful examination of simulation results of MPFR, give cause of reflection on the ways to improve the performance of the MPFR. The main problem of MPFR was that the propagation lines are not "thin" enough to avoid flooding and achieve optimal performance. The solution was to replace the use of PFR with a direct propagation scheme. TWIST runs in three phases. During the first one, the message is forwarded from the source sensor device A along the "optimal line" but only until a randomly selected point; the twisting point T. Then the second phase, called the "twisting phase", is initiated and the message is forwarded along the perimeter of the circle having as center the control center and whose radius is equal to the distance between the control center and the twisting point. In the third phase the message follows a straight route towards the control center. An example of TWIST propagation is illustrated in Fig. 6(d).

Conclusions

In this case study three novel probabilistic forwarding protocols for WSN's where developed by carefully experimentally studying a theoretically designed protocol. This gave us better insight into the original protocol's virtues and flaws. The experimental study identified the impact of specific parts of the protocol in regard to the performance of the protocol. Based on this knowledge a series of modifications and solutions were proposed which all served the same ideal; the "fair" exploitation of all the particles in a network. This concept was approached using different ideas like forcing a neighborhood's "rich" particles to work for the benefit of "poor" ones, or avoiding the greedy propagation paths as the source-destination straight lines.

The results of the algorithm engineering process showed that *fairness* is an important property, since it allows the network to keep its topology and its structure for more time and thus elongate its operational life. Indicatively while the original protocol's success rate was 69.5%, the last version achieved 97.5% and its standard deviation of energy was almost half of that of the initial protocol indicating a much fairer distribution of the communication workload.

4.2. *Game-theoretic vertex coloring*

The second case study looks into one of the most important optimization problems in computer science: the problem of vertex coloring of graphs. Given a graph $G = (V, E)$ of n vertices, assign a color to each vertex of G so that no pair of adjacent vertices gets the same color (i.e., so that the coloring is *proper*) and so that the total number of distinct colors used is minimized. The *chromatic number* of G, denoted by $\chi(G)$, is the global optimum of vertex coloring, i.e., the minimum number of colors needed to properly color the vertices of G.

Finding a good (with respect to the total number of colors used) coloring of the nodes of a WSN has many practical applications: First, colors may be seen as frequencies, so that a proper coloring of the nodes corresponds to a solution to the frequency assignment problem; Furthermore, a coloring of a WSN actually partitions its nodes into subsets (each corresponding to a color), such that no communication link exists between any pair of nodes in the same subset, and such a partition might be useful when designing sleep/awake protocols in order to save energy or providing secure group communication.

The challenge of designing vertex coloring algorithms applicable in WSNs lies in both the intrinsic difficulty of the original problem of vertex coloring and the particularities of WSNs. More specifically, the global optimum of vertex coloring is NP-hard [54], and the best polynomial time approximation algorithm achieves an approximation ratio of $O(n(\log\log n)^2/(\log n)^3)$ [55] (n being the number of vertices). In the distributed setting of wireless sensor networks the problem of vertex coloring has been studied in [56] where a randomized algorithm presented which produces a proper coloring with high probability, required $O(\Delta \log n)$ time and uses $O(\Delta)$ colors, where n and Δ are the number of nodes in the network and the maximum degree, respectively. This algorithm requires knowledge of a linear bound on n and Δ. This result was improved in [57], where the coloring problem

is solved in $O(\Delta + \log \Delta \log n)$ time, given an estimate of n and Δ, and $O(\Delta + \log^2 n)$ without knowledge of Δ, while it needs $\Delta + 1$ colors.

The objective of this case study is to devise coloring protocols for WSN's that are easy to implement, require short period of time to complete and minimize energy consumption. These protocols must always guarantee a proper coloring of the network. On the other hand they have to be decentralized and adaptive to current conditions.

Game-theoretic vertex coloring

Panagopoulou and Spirakis [58] proposed an efficient vertex coloring algorithm that is based on *local search*: Starting with an arbitrary proper vertex coloring (e.g., the trivial proper coloring where each vertex is assigned a unique color), each vertex (one at a time) is allowed to move to another color class of higher cardinality, until no further local moves are possible. This local search method is illustrated in [58] via a game-theoretic analysis, because of the natural correspondence of the local optima of the proposed method to the pure Nash equilibria of a suitably defined *strategic game*. In particular the vertices of the graph are seen as players in a strategic game, where each player has to choose some color, and the payoff of a player is the total number of players that have chosen the same color as his own (unless some neighbor has also chosen the same color, in which case the payoff is 0). The players are allowed to perform selfish steps, i.e., change their colors in order to increase their payoffs, and in [58] it was shown that this selfish improvement sequence converges, in polynomial time, into a pure Nash equilibrium of the game, which is actually a proper coloring of the vertices of the graph that uses a total number of colors satisfying all known upper bounds on the chromatic number of the graph (that is, the minimum number of colors needed to color the graph).

Centralized implementation

A first step in the protocol development is not the straight design of a fully distributed protocol, but to implement the algorithm in a simplified centralized matter. This provides a fast prototype version of the protocol and gives a first impression of the overall performance of the algorithm. This simplified version can be transformed gradually into a distributed protocol that actually exchanges network messages containing protocol payload. The results emerging from this process generally helps by providing optimization feedback for the algorithm design. This development cycle depicted in Fig. 7 helps the developer to start from an initial idea and gradually leads to a fully distributed protocol. In [59] a simple version of the game-theoretical coloring algorithm is provided where every node in the beginning is assigned a different color and then in each step the protocol iterates the nodes assigning the color with the maximum cardinality.

The first attempt to design an algorithm from the game-theoretic centralized method described above faced two challenges, first each node needs to be aware of the cardinalities of each color class in order to decide whether to perform a selfish step and second no pair of neighboring nodes should be able to change their colors simultaneously. In order to deal with both challenges, some kind of coordination between the nodes must be provided. Specifically a central node is introduced that is called the *judge*; it collects information about the cardinalities of each color and sends sequentially a list with all the cardinalities to the nodes. Then each node, upon receiving the list of colors, it has to decide whether to take a selfish step or not and informs the Judge about its new color. Communication between nodes is based on the WiseLib implementation of the well-established ad-hoc routing protocol TORA [60].

As expected, the evaluation of this protocol conducted in [59] indicated the very poor scalability of the system: as the number of nodes increases, both the number of messages and rounds required increase. However, the chromatic number has a dominating role in the overall time complexity. A higher number of colors requires that the nodes take more selfish steps until they reach an equilibrium. This also causes more message exchanges. It is clear from the results that as the number of nodes increases, the number of rounds and messages required gets extremely high making this algorithm impractical.

A major practical problem is also revealed: with the hardware platform used to test the algorithm, the available payload of each packet was about 120 bytes. Therefore a single message can store about 30 colors. So when the number of vertices grew beyond this number, the messages containing the list with the color cardinalities were fragmented in two (or more) packets thus drastically decreasing the available bandwidth and leading to longer execution times.

Simplified distributed implementation

The above algorithm implements closely the design of the theoretical algorithm in a distributed environment, guarantees termination and finds a proper coloring. Still, it has a major drawback: a single node (the Judge) must coordinate the coloring procedure of the entire network. Clearly, the Judge must have significant computational and power resources. Furthermore, the position of the Judge in the network is of paramount importance: almost all messages are either originated by or targeted to the Judge, therefore the distance between a node and the Judge significantly affects the number of messages that the node sends and receives. The existence of a single point of coordination also affects the balance of energy dissipation: nodes that are closer to the Judge participate in more message forwarding than nodes that are further away. Thus, energy is dissipated unevenly by the nodes of the network, yielding to the premature depletion of certain keystone nodes, which in turn results to the abbreviation of the operational life of the whole network. Finally, the Judge is a single point of failure in the network.

In [59] a simplified distributed extension is proposed that replaces the single centralized Judge by introducing multiple Judges: each Judge is in charge of only a region of the network. The idea is that the subset of nodes that belongs to a specific Judge is colored as above. Since only a fraction of the nodes are used, fewer messages will travel in the network and smaller hops will be required to reach all the nodes. Furthermore, Judges require less computational and power resources. The Judges are invoked sequentially passing all the information about the colors used to the next Judge, until the entire network is colored. The sequence of executions

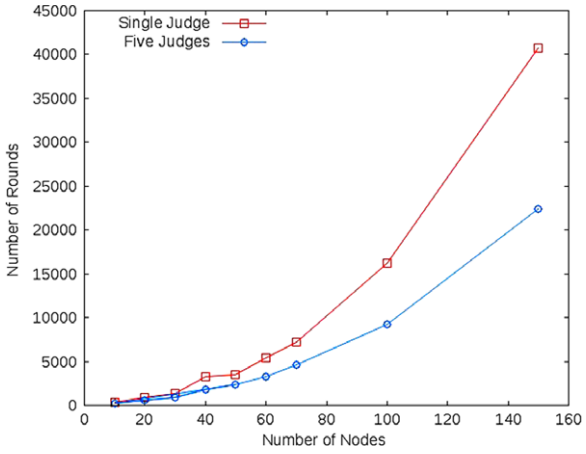


Fig. 8 – Time complexity of the centralized algorithm and the simplified distributed algorithm for different topologies.

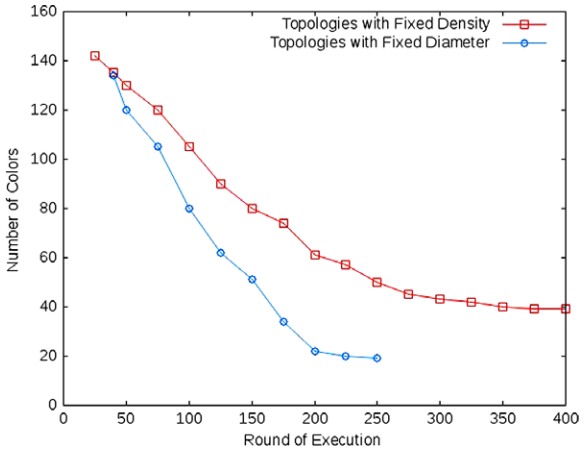


Fig. 9 – Colors used as the simulation evolves for the fully distributed implementation of vertex coloring (using color aggregation) for different type of topologies of fixed number of nodes ($n = 150$).

is done based on priority assigned to the Judges during the partitioning of the network in regions.

Clearly maintaining multiple judges improves the overall communication and time complexity. The simulations conducted in [59] revealed that the improvement seems to experience a threshold effect around 5 Judges beyond which no further reductions are achieved. Based on this result, we here include Fig. 8 that depicts the number of rounds (time complexity) of the single judge protocol and the version where five judges are available. Although the number of rounds is reduced, it is evident that the multi judged protocol remains impractical for large n.

Fully distributed implementation

It is now clear that the algorithm of [58] relies on several assumptions that obstruct its straightforward implementation in a distributed setting. It required (i) only a single node each time can perform a selfish step (mutual exclusion) and (ii) all nodes know the cardinalities of all color classes (global knowledge). The next phase of the Algorithm Engineering cycle was to provide an implementation that deals with these issues, thus enabling its application to distributed settings, while preserving both the efficiency (in terms of time complexity) and the quality (in terms of number of colors used) of the coloring produced.

The solution produced by the Algorithm Engineering cycle relies on an extension of the results of [58,59]. Namely, by raising the requirement that only one vertex at a time is allowed to perform a selfish step in order to guarantee polynomial time convergence into a Nash equilibrium coloring. Additional theoretical analysis conducted in [61] shows that if any subset of non-neighboring vertices perform a selfish step in parallel, then a Nash equilibrium coloring satisfying the bounds given in [58] can still be reached in polynomial time.

This potentiality of parallelization of selfish steps allows one to derive a fully distributed implementation which computes, in polynomial time, a proper coloring of the vertices of a graph that still uses a total number of colors satisfying all known upper bounds on the chromatic number of the graph (that is, the minimum number of colors needed to color the graph). To the best of our knowledge, this is the first distributed, polynomial-time implementation of a vertex coloring algorithm that achieves all the above bounds. The fully distributed algorithm requires $O(n)$ memory to maintain the local lists with the cardinalities of all color classes. Also, the protocol does not require any initial knowledge on the network (e.g., network size, average node degree etc.) but gathers all necessary information dynamically.

The evaluation conducted in [61] examined the ability of the fully distributed algorithm to perform parallel selfish steps. Fig. 9 depicts the number of colors used as the network evolves over time for different types of topologies when $n = 150$. Observe that the algorithm reaches the equilibrium fast and succeeds in parallelizing the process of selfish coloring.

After achieving satisfactory results from the simulated executions of the protocol, the next step was to port the distributed algorithm into the real hardware platform (see Section 3.1) as a final step of validation and evaluation of its performance. This immediately revealed a serious problem related to wireless communication, that of medium congestion. The implementation of the simultaneous execution produced a large number of concurrent exchanges and thus message collisions that were unresolved by the MAC. To reduce the collisions, a short random delay was imposed before each transmission; yet, some messages were still not delivered properly. This was inevitable due to the non-deterministic behavior of the wireless medium.

The experiments allowed one to determine the effect of network density in the execution of the protocol (i.e., collisions, etc.). It was observed that up to network sizes of 8 nodes the results where always correct (i.e., messages were delivered properly), while in larger networks message losses always occurred forcing the protocol to produce inconsistent results. The experiments also allowed to determine the real execution time of a protocol phase being about 130 ms. Finally the experiments allowed one to measure the real time required to collect the colors used by the nodes and distribute the aggregated list back to the network. The initialization of this process required about 750 ms and then for each further hop an additional time of about 300 ms were required.

Conclusions

This case study that was conducted using both simulated and experimental environments indicates interesting aspects of the vertex coloring problem. The Algorithm Engineering process led to a practical algorithm that can be executed in real networks. Both theoretical and experimental analysis indicate that the final solution successfully raises the limitation of [58] where only one vertex at a time is allowed to perform a selfish step in order to guarantee polynomial time convergence into a Nash equilibrium coloring. The fully distributed algorithm reduces both time and communication complexity while producing colorings of the same quality with the initial algorithm.

REFERENCES

[1] M. Weiser, Some computer science issues in ubiquitous computing, Commun. ACM 36 (1993) 75–84.

[2] B.M.E. Moret, Towards a discipline of experimental algorithmics, in: M.H. Goldwasser, D.S. Johnson, C.C. McGeoch (Eds.), Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges, in: DIMACS Monographs, vol. 59, AMS Press, 2002, pp. 197–213.

[3] J. Lee, M. Perkins, S. Kyperountas, Y. Ji, RF propagation simulation in sensor networks, in: SENSORCOMM'08: Proceedings of the 2008 Second International Conference on Sensor Technologies and Applications, IEEE Computer Society, Washington, DC, USA, 2008, pp. 604–609.

[4] D. Kotz, C. Newport, R.S. Gray, J. Liu, Y. Yuan, C. Elliott, Experimental evaluation of wireless simulation assumptions, in: MSWiM'04: Proceedings of the 7th ACM International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems, ACM Press, New York, NY, USA, 2004, pp. 78–82.

[5] S. Ivanov, A. Herms, G. Lukas, Experimental validation of the ns-2 wireless model using simulation, emulation, and real network, in: 4th Workshop on Mobile Ad-Hoc Networks, WMAN'07.

[6] P. Dutta, J. Hui, J. Jeong, S. Kim, C. Sharp, J. Taneja, G. Tolle, K. Whitehouse, D. Culler, Trio: enabling sustainable and scalable outdoor wireless sensor network deployments, in: 5th International Conference on Information Processing in Sensor Networks, IPSN, ACM Press, New York, 2006, pp. 407–415.

[7] B. Chun, P. Buonadonna, A. AuYoung, C. Ng, D. Parkes, J. Shneidman, A. Snoeren, A. Vahdat, Mirage: a microeconomic resource allocation system for sensornettestbeds, in: 2nd IEEE Workshop on Embedded Networked Sensors.

[8] E. Ertin, A. Arora, R. Ramnath, M. Nesterenko, V. Naik, I. Bapat, V. Kulathumani, M. Sridharan, H. Zhang, H. Cao, Kansei: a testbed for sensing at scale, in: 5th International Conference on Information Processing in Sensor Networks, IPSN.

[9] UC Berkeley, WSN test beds at UC Berkeley, 2010. http://www.millennium.berkeley.edu/sensornets/.

[10] G. Werner-Allen, P. Swieskowski, M. Welsh, Motelab: a wireless sensor network testbed, in: IPSN-SPOTS'05, California, USA, pp. 483–488.

[11] V. Handziski, A. Kopke, A. Willig, A. Wolisz, TWIST: a scalable and reconfigurable wireless sensor network testbed for indoor deployments, Technical Report TKN-05-008, 2005.

[12] TUTORNET, Tutornet: a tiered wireless sensor network testbed, 2009. http://enl.usc.edu/projects/tutornet/.

[13] D. Johnson, T. Stack, R. Fish, D. Flickinger, L. Stoller, R. Ricci, J. Lepreau, Mobile emulab: a robotic wireless and sensor network, in: 25th IEEE Conference on Computer Communications, INFOCOM, pp. 23–29.

[14] Oulu, Oulu Smart City, 2010. http://www.ubiprogram.fi.

[15] German Telekom, City of Friedrichshafen, Friedrichshafen Smart City, 2010. http://www.telekom.com/dtag/cms/content/dt/en/395380.

[16] Cambridge (MA) Smart City, CitySense—an open, urban-scale sensor network testbed, 2010. http://www.citysense.net.

[17] MetroSense, Secure people-centric sensing at scale, 2009. http://metrosense.cs.dartmouth.edu/.

[18] SensLAB, Very large scale open wireless sensor network testbed, 2010. http://www.senslab.info/.

[19] T. Baumgartner, I. Chatzigiannakis, M. Danckwardt, C. Koninis, A. Kröller, G. Mylonas, D. Pfisterer, B. Porter, Virtualising testbeds to support large-scale reconfigurable experimental facilities, in: J.S. Silva, B. Krishnamachari, F. Boavida (Eds.), Proceedings of the 7th European Conference on Wireless Sensor Networks, EWSN 2010, in: LNCS, vol. 5970, Springer, Heidelberg, 2010, pp. 210–223.

[20] T. Baumgartner, I. Chatzigiannakis, S.P. Fekete, C. Koninis, A. Kröller, A. Pyrgelis, Wiselib: a generic algorithm library for heterogeneous sensor networks, in: J.S. Silva, B. Krishnamachari, F. Boavida (Eds.), Proceedings of the 7th European Conference on Wireless Sensor Networks, EWSN 2010, in: LNCS, vol. 5970, Springer, Heidelberg, 2010, pp. 162–177.

[21] University of Southern California, Information Sciences Institute (ISI), Ns-2: Network Simulator-2, 2010. http://www.isi.edu/nsnam/ns/.

[22] J. Kačer, Discrete event simulations with J-Sim, in: J.F. Power, J.T. Waldron (Eds.), Proceedings of the Inaugural Conference on the Principles and Practice of Programming in Java, Trinity College Dublin, Department of Computer Science, National University of Ireland, Maynooth, Co. Kildare, Ireland, 2002, pp. 13–18.

[23] A. Varga, OMNeT++: objective modular network testbed in C++, 2009. http://www.omnetpp.org.

[24] B.K. Szymanski, G. Chen, J.W. Branch, L. Zhu, SENSE: sensor network simulator and emulator, 2009. http://www.cs.rpi.edu/~cheng3/sense/.

[25] P. Levis, N. Lee, M. Welsh, D. Culler, TOSSIM: accurate and scalable simulation of entire TinyOS applications, in: Proceedings of the First ACM Conference on Embedded Networked Sensor Systems, SenSys 2003. http://www.cs.berkeley.edu/~pal/research/tossim.html.

[26] B. Titzer, D.K. Lee, J. Palsberg, Avrora: scalable sensor network simulation with precise timing, in: Proceedings of the Fourth International Symposium on Information Processing in Sensor Networks, IPSN 2005, IEEE, 2005, pp. 477–482.

[27] A. Fraboulet, G. Chelius, E. Fleury, Worldsens: development and prototyping tools for application specific wireless sensors networks, in: IPSN'07: Proceedings of the 6th International Conference on Information Processing in Sensor Networks, ACM, New York, NY, USA, 2007, pp. 176–185.

[28] A. Kröller, D. Pfisterer, C. Buschmann, S.P. Fekete, S. Fischer, Shawn: a new approach to simulating wireless sensor networks, in: Proceedings of the Design, Analysis, and Simulation of Distributed Systems Symposium 2005, DASD'05, pp. 117–124.

[29] S.P. Fekete, A. Kröller, S. Fischer, D. Pfisterer, Shawn: the fast, highly customizable sensor network simulator, in: Proceedings of the Fourth International Conference on Networked Sensing Systems, INSS 2007.

[30] S. Kurkowski, T. Camp, N. Mushell, M. Colagrosso, A visualization and analysis tool for Ns-2 wireless simulations: iNSpect, in: MASCOTS'05: Proceedings of the 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, IEEE Computer Society, Washington, DC, USA, 2005, pp. 503–506.

[31] Naval Research Laboratory, ns2sensors: NRL's sensor network extension to Ns-2, 2007. http://nrlsensorsim.pf.itd.nrl.navy.mil/.

[32] S. Park, A. Savvides, M.B. Srivastava, Sensorsim: a simulation framework for sensor networks, in: MSWIM'00: Proceedings of the 3rd ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems, ACM, New York, NY, USA, 2000, pp. 104–111.
[33] IEEE 802.11 working group, IEEE 802.11 wireless local area networks, 2009. http://www.ieee802.org/11/.
[34] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, D. Culler, TinyOS: An Operating System for Wireless Sensor Networks, Springer.
[35] The TinyOS 2.x working group, TinyOS 2.0, in: SenSys'05: Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems.
[36] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, K. Pister, System architecture directions for networked sensors, SIGOPS Oper. Syst. Rev. 34 (2000) 93–104.
[37] V. Shnayder, M. Hempstead, B.-R. Chen, G.W. Allen, M. Welsh, Simulating the power consumption of large-scale sensor network applications, in: Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems, ACM Press, 2004, pp. 188–200.
[38] Crossbow Technology Inc., Mica2Mote, 2009. http://www.xbow.com.
[39] W. Li, X. Zhang, W. Tan, X. Zhou, *H*-tossim: extending tossim with physical nodes, Wirel. Sensor Netw. 1 (2009) 324–333.
[40] L. Girod, T. Stathopoulos, N. Ramanathan, J. Elson, D. Estrin, E. Osterweil, T. Schoellhammer, A system for simulation, emulation, and deployment of heterogeneous sensor networks, in: SenSys'04: Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems, ACM, New York, NY, USA, 2004, pp. 201–213.
[41] L. Girod, N. Ramanathan, J. Elson, T. Stathopoulos, M. Lukac, D. Estrin, Emstar: a software environment for developing and deploying heterogeneous sensor-actuator networks, ACM Trans. Sens. Netw. 3 (2007) 13.
[42] L. Girod, J. Elson, A. Cerpa, T. Stathopoulos, N. Ramanathan, D. Estrin, Emstar: a software environment for developing and deploying wireless sensor networks, in: USENIX Tech. Conf.
[43] Y. Wen, S. Gurun, N. Chohan, R. Wolski, C. Krintz, Accurate and scalable simulation of network of heterogeneous sensor devices, J. Signal Process. Syst. 50 (2008) 115–136.
[44] Stargate, 2009. http://platformx.sourceforge.net/.
[45] S. Gayen, E.J. Tyson, M.A. Franklin, R.D. Chamberlain, P. Crowley, *X*-sim: a federated heterogeneous simulation environment, in: 10th High Performance Embedded Computing Workshop, pp. 75–76.
[46] F. Österlind, A. Dunkels, T. Voigt, N. Tsiftes, J. Eriksson, N. Finne, Sensornet checkpointing: enabling repeatability in testbeds and realism in simulations, in: Proceedings of the 6th European Conference on Wireless Sensor Networks, EWSN'09, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 343–357.
[47] A. Dunkels, B. Gronvall, T. Voigt, Contiki—a lightweight and flexible operating system for tiny networked sensors, in: LCN'04: Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks.
[48] B. Karlsson, Beyond the C++ Standard Library: An Introduction to Boost, Addison-Wesley, Crawfordsville, IN, 2005.
[49] L. Kettner, Software design in computational geometry and contour-edge based polyhedron visualization, Ph.D. Thesis, ETH Zurich, Switzerland, DISS. ETH No. 13325, 1999.
[50] C. Intanagonwiwat, R. Govindan, D. Estrin, J.S. Heidemann, F. Silva, Directed diffusion for wireless sensor networking, IEEE/ACM Trans. Netw. 11 (2003) 2–16.
[51] I. Chatzigiannakis, T. Dimitriou, S.E. Nikoletseas, P.G. Spirakis, A probabilistic algorithm for efficient and robust data propagation in wireless sensor networks, Ad Hoc Networks 4 (2006) 621–635.
[52] I. Chatzigiannakis, L.M. Kirousis, T. Stratiotis, Probabilistic protocols for fair communication in wireless sensor networks, in: Algorithmic Aspects of Wireless Sensor Networks, Fourth International Workshop, ALGOSENSORS 2008, Reykjavik, Iceland, July 2008. Revised Selected Papers, in: Lecture Notes in Computer Science, vol. 5389, Springer, 2008, pp. 100–110.
[53] S. Rangwala, R. Gummadi, R. Govindan, K. Psounis, Interference-aware fair rate control in wireless sensor networks, SIGCOMM Comput. Commun. Rev. 36 (2006) 63–74.
[54] R.M. Karp, Complexity of Computer Computations, Plenum Press, pp. 85–103.
[55] M.M. Hallórsson, A still better performance guarantee for approximate graph coloring, Inform. Process. Lett. 45 (1993) 19–23.
[56] T. Moscibroda, R. Wattenhofer, Coloring unstructured radio networks, in: SPAA'05: Proceedings of the Seventeenth Annual ACM Symposium on Parallelism in Algorithms and Architectures, ACM, New York, NY, USA, 2005, pp. 39–48.
[57] J. Schneider, R. Wattenhofer, Coloring unstructured wireless multi-hop networks, in: PODC'09: Proceedings of the 28th ACM Symposium on Principles of Distributed Computing, ACM, New York, NY, USA, 2009, pp. 210–219.
[58] P. Panagopoulou, P. Spirakis, A game theoretic approach for efficient graph coloring, in: The 19th International Symposium on Algorithms and Computation, ISAAC 2008, 2008, pp. 183–195.
[59] I. Chatzigiannakis, A. Kataras, P. Panagopoulou, P. Spirakis, Game-theoretic vertex coloring in wireless sensor networks, Technical Report, 2010.
[60] V. Park, M. Corson, A highly adaptive distributed routing algorithm for mobile wireless networks, in: 16th Annual IEEE International Conference on Computer Communications and Networking, INFOCOM 1997, IEEE, 1997, pp. 1405–1413.
[61] I. Chatzigiannakis, C. Koninis, P. Panagopoulou, P. Spirakis, Distributed game-theoretic vertex coloring, in: 14th International Conference on Principles of Distributed Systems, OPODIS, in: LNCS, Springer-Verlag, Berlin, Heidelberg, 2010.

Tobias Baumgartner received his Diploma in Computer Science at Braunschweig University of Technology, Germany (2006). He is currently a Ph.D. student in the Algorithms Group of Prof. Fekete at Braunschweig University of Technology, Germany. In 2007, he worked as a Software Developer in the field of wireless sensor networks for ScatterWeb GmbH, Berlin, Germany. His main research interests are distributed algorithms for sensor networks, and programming paradigms for tiny embedded systems—with a particular focus on sensor nodes.

Ioannis Chatzigiannakis obtained his Ph.D. from the Department of Computer Engineering & Informatics of the University of Patras in 2003. He is currently Adjunct Faculty at the Computer Engineering & Informatics Department of the University of Patras (since October 2005). He is the Director of the Research Unit 1 of RACTI (since July 2007). He has coauthored over 70 scientific publications. His main research interests include distributed and mobile computing, wireless sensor networks, algorithm engineering and software systems. He has served as a consultant to major Greek computing industries. He is the Secretary of the European Association for Theoretical Computer Science since July 2008.

Sándor P. Fekete received his Doctoral Degree in Combinatorics and Optimization from the University of Waterloo Science, Canada (1992). Currently, he is full professor for Algorithmics in the Department of Computer Science of the Braunschweig University of Technology, where he is also Department Chair. His research interests lie in both theoretical and practical aspects of algorithms, with a large variety of interdisciplinary collaborations. One of his main interests have been distributed algorithms, in particular in the context of large sensor networks. He has coauthored about 150 refereed scientific publications.

Stefan Fischer is a full professor for computer science and director of the ITM. After his Ph.D. in 1996 (University of Mannheim, Germany) and being a postdoc researcher at the University of Montreal, he joined the International University in Germany as an Assistant Professor, in 1998. In 2001 he became an Associate Professor at the TU Braunschweig, and finally he joined the University of Lübeck in 2004. His current research concentrates on massively distributed systems such as sensor networks. He has published over 100 reviewed scientific articles and 6 books.

Christos Koninis received his Diploma in Computer Engineering and Informatics (2008) and is working as a researcher in Research Unit 1 of CTI since 2007. He is currently pursuing his Ph.D. in the Dpt. of Computer Engineering and Informatics, at the University of Patras. His research interests lie in the area of wireless sensor networks and large-scale computer networks simulation.

Alexander Kröller is a senior researcher and lecturer at TU Braunschweig. He received his Diploma in Mathematics from TU Berlin and his Ph.D. from TU Braunschweig; his thesis on Algorithmic Methods for Wireless Sensor Networks was nominated for the Dissertation Award of the German Society of Computer Science (GI). He specialises in algorithmic methods for distributed systems, e.g., wireless sensor networks. Dr. Kröller is a member of the Steering Committee of Wireless Sensor Networks: Theory and Practice (WSN).

Daniela Krüger is an assistant researcher at the Institute of Telematics at the University of Lübeck. She received her diploma in computer science from the University of Lübeck in 2005. Her research interests span programming techniques, middleware and context awareness in sensor networks. Additionally, she was involved in a research project that focussed on secure area monitoring using sensor networks.

Georgios Mylonas received his Ph.D. degree from the Department of Computer Engineering and Informatics at the University of Patras, Greece, in December 2008. He is currently working as a postdoc researcher at the Research Academic Computer Technology Institute, Patras, Greece. His research interests lie in the areas of wireless sensor networks and distributed systems and games. He has been involved in the AEOLUS, WISEBED and ALGODES PENED research projects, funded by the European Union and the Greek Government, which focus on the algorithmic and software issues related to wireless sensor networks.

Dennis Pfisterer is a senior researcher and tenured lecturer at the ITM. After his studies of Information Technology at the University of Applied Sciences in Mannheim, he worked as a research assistant at the European Media Laboratory (http://www.eml.org) in Heidelberg in the area of resource adaptive systems. In 2003, he joined Prof. Fischer's group at the Braunschweig Institute of Technology, and followed him in 2005 to Lübeck to continue his research on sensor networks at the Institute of Telematics. After his Ph.D., he broadened his research interests and now works on sensor networks in general, their integration with the Future Internet as well as Service Oriented Architectures (SOA). He has published more than forty reviewed publications in his area of expertise.