Diss. ETH No. 16204

Design and Deployment of Wireless Networked Embedded Systems

A dissertation submitted to the
SWISS FEDERAL INSTITUTE OF TECHNOLOGY ZURICH

for the degree of
Doctor of Technical Sciences

presented by
JAN BEUTEL
Dipl. El.-Ing., ETH Zurich
born August 6, 1973
citizen of Germany

accepted on the recommendation of
Prof. Dr. Lothar Thiele, examiner
Prof. Dr. Jan M. Rabaey, co-examiner

2005

Abstract

The recent rise and widespread adoption of wireless networking technologies for mobile communication applications has sparked numerous visions of an ever more networked and interactive world. One such vision proposed in the late nineties are wireless sensor networks, where wireless communication and computing elements are combined with integrated sensors to enable tightly coupled interaction with the physical world. As a new application domain for wireless technology, key challenges here are (i) the limited resources of the highly integrated nodes that are to be leveraged by the amount of devices deployed and the collaboration between them, (ii) the tight coupling of application, nodes and the environment and (iii) the broad usage profile by systems-experts and non-expert users alike.

First medium-scale experiments and field-trials have reported that it is increasingly hard to design, develop, deploy, test and validate systems consisting of more than a hand-full of nodes, especially when situated in a real-world environment. Prototypical applications are scarce, seldomly consisting of more than a few tens to a hundred nodes. Setting up large, heterogeneous, interactive and functional systems as forecast in the visions is no small task; currently more an art than a systematic engineering effort. Coordinated methods and tools for the design and deployment of wireless networked embedded systems are missing today.

With this work, we to contribute to the design and development of wireless networked embedded systems. The specific contributions are presented and discussed in the context of a vertical slice of the design space concerned and the relevant questions encountered:

- Functional and qualitative requirements of a location management service for wireless sensor networks based on measurements and simulation have been developed. We present one of the earliest algorithms for the distributed computation of node location.
- A novel platform for fast-prototyping of wireless sensor networks has been developed. This platform has successfully served numerous researchers, among ourselves, as an underlying infrastructure for experimentation and education.
- The BTnode platform has been used to develop multihop networks and topology control mechanisms for Bluetooth scatternets. To our knowledge the experiments presented are the largest connected Bluetooth scatternets reported to date.
- The concept of a deployment-support network as a powerful tool for the development, deployment, test and validation of wireless sensor networks is presented in conjunction with experimental evidence on the feasibility of the approach.

Kurzfassung

Die jüngsten Erfolge und verbreitete Akzeptanz drahtloser Netzwerktechnologien für Anwendungen der Mobilkommunikation hat zu zahlreichen Visionen einer immer stärker vernetzten und interagierenden Welt geführt. Eine solche Vision aus den späten Neunzigern sind die drahtlosen Sensornetze, die drahtlose Kommunikation und Rechenelemente mit integrierter Sensorik kombinieren um direkt mit der physikalischen Umgebung interagieren zu können. Als neues Anwendungsgebiet drahtloser Netze sind die hauptsächlichen Herausforderungen hier (i) die eingeschränkten Fähigkeiten der einzelnen, hochintegrierten Knoten, (ii) die enge Koppelung von Anwendung, Knoten und Umgebung sowie (iii) das breite Anwendungsspektrum sowohl durch Experten wie auch ungeschulte Anwender.

Erste mittelgrosse Experimente und Feldversuche berichten von zunehmenden Schwierigkeiten, Systeme bestehend aus mehr als einer Hand voll Knoten zu entwerfen, zu entwickeln, in Betrieb zu nehmen und zu testen; dies vor allem beim Einsatz in einer realen Umgebung. Solch prototypische Anwendungen bestehen selten aus mehr als einigen zehn bis zu hundert Knoten. Die Entwicklung grosser, heterogener und interaktiver Systeme wie in den Visionen beschrieben, gleicht heute mehr einer Kunst als einer systematischen Ingenieursleistung. Aufeinander abgestimmte Methoden und Werkzeuge für die Entwicklung und Inbetriebnahme drahlos vernetzter eingebetteter Systeme fehlen heute weitgehend.

Mit dieser Arbeit tragen wir zur Entwicklung und Inbetriebnahme solcher Systeme bei. Im speziellen zeigen wir anhand eines senkrechten Schnittes des Entwurfsraumes relevante Fragen und Lösungen auf und diskutieren diese:

- Funktionelle und qualitative Anforderungen eines Dienstes zur Positionsbestimmung wurden anhand von Messungen und Simulationen entwickelt. Einer der ersten Algorithmen zur verteilten Berechnung der Knotenposition wird vorgestellt.
- Eine neuartige Platform für den Bau von Prototypen drahtloser Sensornetze wurde enwickelt. Diese wurde von zahlreichen Forschern als Basisinfrastruktur für Ihre Experimente und in der Lehre erfolgreich eingesetzt.
- Die BTnode Platform wurde verwendet um Transportmechanismen und Topologiesteuerung von Bluetooth Scatternetzen zu entwickeln. Die daraus resultierenden Experiemente stellen die grössten, bisher dokumentierten Bluetooth Scatternetze dar.
- Das Konzept des deployment-support networks wird als mächtiges Werkzeug für die Entwicklung, Inbetriebnahme, Test und Abnahme drahloser Sensornetze eingeführt.

Acknowledgements

First of all I would like to thank Lothar Thiele for his visions, guidance and support throughout this research. His excellent questions and valuable discussions have been both inspiring and motivating, the trust and confidence endowed allowed maximum freedom and creativity in the pursuit of new ideas. I am also grateful for the pleasant research environment and the substantial resources provided that enabled to gain such extensive practical experience through the experimental work performed.

Then I would like to thank Jan Rabaey for his spontaneous willingness to co-examine my thesis. He introduced me to academic research during my stay at the Berkeley Wireless Research Center prior to my PhD studies, sparked the initial motivation to pursue this path and paved the broad direction of my research interests. Much of the work on positioning is based on this collaboration.

What started out as a small joint-venture has since evolved into a serious and successful project, a collaboration across lab boundaries, an industrial technology transfer and numerous well-received publications. None of the members of the BTnode core team would have been able to do so alone, least of all myself. For this I am deeply grateful to Oliver Kasten, Matthias Ringwald, Kay Römer, Friedemann Mattern, Matthias Dyer and Lothar Thiele. The National Center of Competence in Research on Mobile Information and Communication Systems (NCCR-MICS) has provided an organizational roof and the necessary funding for our joint endeavor.

Over the years many others have directly contributed to the success of the BTnode platform both through academic affiliation such as Philipp Blum, Lennart Meier, Martin Hinz, Luca Negri, Clemens Moser, Frank Siegemund, Regina O'Dell-Bischoff, Aaron Zollinger and Roger Wattenhofer as well as industrial affiliation such as Michael Scheffler, Etienne Hirt and Andreas Thiel.

Many colleagues from all over the world have been great discussion partners and helped create my vivid network of active discussion: Chris Savarese, Josie Ammer, Robert Szewczyk, Joe Polastre, Philippe Bonnet, Ralph Kling, Lama Nachman, Jeremy Elson, Phil Levis, David Culler, Koen Langendoen, Kathy Sohrabi, Albrecht Schmidt, Holger Karl, Polly Huang, Nirupama Bulusu, Amre El-Hoiydi, Christian Enz and Martin Vetterli. Not to forget the many colleagues here at ETH: Simon Künzli, Christian Plessl, Herbert Walder, Ernesto Wandler, Rolf Enzler, Urs Anliker, Paul Lukowicz, Holger Junker, Matthias Gries, Marco Platzner, Samarjit Chakraborty, Michael Eisenring, Jörn Janneck, Martin May,

Lukas Ruf, Matthias Bosshard, Marc Langheinrich, Marco Laumanns, Jonas Greutert, Alexander Maxiaguine, Stephan Bleuler and Eckart Zitzler.

Contents

Tables

Figures

1

Introduction

Wireless sensor networks, ad hoc networks, pervasive and ubiquitous computing, mobile computing, mobile communications and wearable computing are all technology trends and themes that have aroused broad interest over the past years. Many prominent visions have forecast these to persist and impact large parts of our daily life. The visionary standpoint of ubiquity and electronics embedded everywhere put forward by Weiser in 1991 [Wei91] is certainly one of the most profound visions in the mobile computing community.

> The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it.
>
> — Marc Weiser, "The Computer for the 21st Century"

The move from the one-computer-for-many-users in the early age of the mainframe computers to a one-computer-per-user personal computer era has long been accomplished and with the above mentioned technologies we are currently right on track in transition to the many-computers-per-user scenario of the ubiquitous computing era that Weiser predicted. Currently, trends show that in the near future there will be more and more devices and services interacting, forming a variety of heterogeneous systems.

In their seminal articles, Estrin [EGHK99] and Kahn [KKP99] presented a far-reaching vision of wireless sensor networks (WSN), where collections of tiny autonomous computers would collaboratively and unobtrusively monitor a variety of real-world phenomena with unprecedented quality and scale, bringing substantial benefits to a variety of application areas. Since then, numerous hardware platforms have been developed, operating system abstractions have been established, a large number of protocols and algorithms for networking, communication, and information processing have been proposed, and various fundamental capabilities and limitations of these sensor networks have been examined. Based on these ingredients, prototypical applications, e.g. [JOW+02, SBP+04, SOP+04], have been developed, some of which consist of more than 100 sensor nodes. Such networks are formed from a set of small sensor devices, the nodes, that are deployed in an

ad hoc fashion and cooperate in sensing a physical phenomenon. The wireless sensor network (WSN) nodes [CM04] are a novel platform class in the sense of Gordon Bell's law of 1972 that states that technology advances enable a new, lower-priced, higher-volume computing platform or class to form every decade. In a WSN node communication, computation and storage as well as interface and sensing are combined to perform all necessary tasks on a single device. Constituting a key challenge, this is quite a different approach from the traditional internet-based computing world, where many dedicated devices each performing single dedicated functions such as routing, storage, web-services, connections to peripherals and terminal functions to interface to users are coupled together to form systems.

Wireless sensor networks regarded from a second perspective are not a single area of novel fundamental research but a collection of often well-known techniques and established know-how applied to a new domain. Many of the application classes envisioned for sensor networks actually have a long tradition and specific solutions exist. Examples of such traditional sensor network application classes are global networks of weather stations, underwater submarine surveillance developed during the cold war, environmental monitoring applications, building surveillance and factory automation. They typically employ wireline communication, make heavy use of infrastructure that is deployed and installed at designated locations, consist of complicated, full-custom components with little opportunity for reuse and as a result are exceedingly costly.

Scalability to vast numbers of nodes has been an important focus in past wireless sensor network research but the amount of nodes making up a sensor network is debatable, depending on the application settings envisioned [RM04]. Resource constrained devices are not new either, as embedded and also real-time systems have a long tradition. Novel is indeed that wireless sensor networks are (i) a new application domain for wireless technology, (ii) the very limited resources of each node are to be leveraged by the amount of devices deployed and the collaboration between them, (iii) the tight coupling of application, nodes and the environment and (iv) the broad usage profile that envisions that non-expert users, e.g. a marine biologist, design, deploy and maintain these new computing and communication systems according to their needs.

To a large part driven by the achievements in miniaturization, the advances in integration of complex wireless integrated products and the increase in performance and functionality of the design tools, the requirements and properties of a single device are today well understood. Yet it is not simple and straightforward to implement the WSN concepts into a functional prototype system or even a commercial product. The interaction of many devices is often underestimated and design, test, deployment and validation of wireless sensor networks today are in their infancy, posing hard research and technological questions. Especially wireless sensor networks are often proposed to follow a cross-layer approach, taking care of many design decisions and requirements from a unified standpoint. In the experience gained from the implementation of WSN experiments and deployments such as documented by Szewczyk [SPMC04], Cerpa [CEH+01], Simon [SBP+04], Zhang [ZSLM04] and others it has been shown that the system aspects are much more complex than initially

anticipated. Development is hard, error-prone and repetitive. Since wireless sensor networks are usually assumed to be implemented as distributed systems using small, resource constrained devices that are remotely deployed, meticulous care has to be taken to design simple, robust and failure tolerant algorithms. The features and application support offered on highly integrated devices such as the Berkeley Motes [HSW+00, HHKK04], the Intel Imotes [KAH+04] or the BTnodes [BKM+04] is quite the opposite when compared to the ever increasing performance available in the traditional computing world. In order to fully understand the complexity of the matter from a system perspective it is necessary to not only model and/or simulate but also to implement and test on real-world systems.

From these medium-scale field experiments, the step to real-world applications of sensor networks under production conditions does not seem to be a large one at first. However, taking a closer look at the development process of these prototypical applications reveals that putting a functional and operable sensor network in place is currently an art. Setting up large, interactive systems playing together over a large scale of heterogeneous devices and technologies as forecast in the visions is no small task. Systems issues are the hard parts: Bringing together optimized solutions for the many cross-layer issued from a unified systems perspective for power/energy, timing, performance, lifetime, duty cycles, form factor, cost, flexibility etc. In order to close the gap between current proof-of-concept and real-world sensor networks, this artwork has to be replaced with an efficient, coordinated design and development process equally accessible to different users, e.g. assuming roles as (i) hardware and systems software engineer, (ii) application programmer, (iii) deployment technician and (iv) end-user of a sensor network such as the marine biologist mentioned earlier.

In this thesis we examine challenges and specific research issues that have to be addressed in order to enable such a development process. We are raising the question why only few have managed to create viable and sustainable solutions and we identify the key obstacles and problems that appear in practical multihop networking for wireless sensor networks. In the work presented here, we want to highlight the idiosyncrasies of multihop ad hoc networking when implemented on real devices. More specifically, we will focus on the area of Bluetooth scatternet formation, where many different algorithms for mesh structures either conforming to the Bluetooth standard [Bas02], with star [PBC03] or tree topologies [ZBC01] have been proposed. A recent distributed algorithm with bounds on complexity proposed by Law [LMS03] has been validated by simulations as well as a comparative study by Basagni [BBMP04]. Often such work takes into account assumptions on physical prerequisites not readily available in real devices and systems, or simply perfect performance, which is not encountered in wireless systems embedded in the physical world. The extensive work on ad hoc networking simulation is most suitable to answer specific, singular questions and tradeoffs but seemingly inappropriate when approaching system level topics. Generally accepted assumption for simulation work like an 802.11 physical model or an example 1000x1000 grid of evenly distributed nodes and random way-point models certainly lack when relating simulation to real-world experimentation with interference, random failures, imperfections, and human interaction.

1.1 Characteristics of Deeply Embedded Networks

Mobile ad hoc networks (MANETs) that are usually characterized by themes such as high overall node mobility, considerable power and resource consumption at the nodes and moderate network sizes have brought about quite a change to the traditional connection-oriented, infrastructure-dominated telecommunications world [IK96, Per01]. Wireless sensor networks have departed from this traditional viewpoint in an even more radical way as they are envisioned to be deeply embedded in very large quantity into the physical world. Myriads of smart devices ranging from naive identity tags and simple sensors broadcasting singular values such as temperature readings to much more sophisticated computing devices with multiple sensor/actor subsystems, advanced signal processing, data aggregation, extensive storage and diverse communication capabilities will be in constant interaction with each other, the surrounding ubiquitous digital infrastructure and human users. Wireless sensor networks are at one extreme of the design space, embracing such paradigm as immense scale, self-containment, self-organization, deep embedment and therefore also limited access of and to the devices involved [ECPS02].

1.1.1 Wireless Sensor Network Visions

An early realization of "Anytime, Anywhere, Anyform"information systems was proposed by Stemm [SK99] and Katz [KB95] in the context of the BARWAN project [KBA+96], where wireless overlay networks offer a hierarchical structure of room-size, building-size, and wide area data networks, to solve the problem of providing network connectivity to a large number of mobile users in an efficient and scalable way.

The vision of “Smart Dust” put forward by Kahn [KKP99] in 1999 has sparked numerous activities in both theoretical and applied research of ultra-small networked embedded systems. Their vision foresaw a myriad of cubic millimeter, self-powered micro electromechanical systems (MEMS) devices using optical links to communicate to a central system. Here the claim that only a varying percentage of the total devices deployed are contributing to an actual network is key. Devices are thought to be so small and cheap that the extreme over-provisioning of the dust resources can actually become a reality. In the following this vision has lead to the development of numerous comodity-off-the-shelf (COTS) dust platforms [WLLP01, HSW+00].

Under the main prerequisite of wireless sensor networks, namely their stringent energy budget, the first “classical” approaches have focused on efficient, symmetrical ad hoc network configurations, meaning that the devices at both ends of a communication channel were essentially of the same architecture. These early prototypical “sensor nodes” are now constantly maturing and shift the focus from the fabrication of the single device as done by Hill [HSW+00] and Rabaey [RAdSJ+00] toward the management of large, heterogeneous systems and architectures and the services embedded into these. In order to be able to scale to large networks consisting of the most heterogeneous clustered devices, mechanisms and services need to be tailored specifically for interoperability and the optimal management of the limited resources available on such nodes.

The paintable computing effort by Butera [But02] carries on this vision and sees miniature computing devices embedded into paint for easy application onto all surfaces whereas the Amorphous Computing project [AAC+00] focuses on biologically inspired self-assembly mechanisms. Key questions being targeted are coherent behavior from large numbers of cooperating, unreliable parts that are interconnected in unknown, irregular, and time-varying ways as well methods suitable for the instruction of myriads of programmable entities.

Address freedom in networking was a radical theme taken up by the directed diffusion research of Estrin et al. [EGHK99, IGE00] and has since been on the agenda of algorithmic research and scale-free networks by Barabasi [BB03]. Here data is forwarded not toward explicit destinations but alongside gradients of interest.

The definition of the "Terminode" was coined by Hubaux et al. [HLBGH99, HLBG+00, HGLBV01, BBS+01] combining the vision of pervasive and ubiquitous access to information with ad hoc and mobility requirements on a larger scale.

Recently also industry has taken up the realm of sensor networks as a focus area and already more and more companies are churning out sensor network devices, services and first commercial applications.

The US based Committee on Networked Systems of Embedded Computers defined in their 2001 report "Embedded Everywhere - A research agenda for Networked Systems of Embedded Computers" [Com01] the following general characteristics for such EmNets:

- Multiple interacting nodes
- Embeddment into control systems without human intervention
- Purpose other than general computing and communication
- Natural or engineered contexts

Furthermore the following special characteristics of EmNet requirements and their applicable technical solutions were specified:

- Energy-constrained nodes versus non-energy-constrained nodes
- Fixed topology versus flexible topology
- Safety-critical applications versus non-safety critical applications
- Highly engineered versus unconstrained, ad-hoc systems

Specifically, EmNets present the challenge of building large systems that are

- Tightly coupled with the physical world
- Resource-constrained environment
- Persist for long periods of time
- Many interacting components
- Used and interacted with by non expert users

Due to the wide intended application domain of EmNets and the apparent uncertainty of a unified approach or technology it is apparent that broad system solutions will have to be sought. Many of the technologies involved have specialized and highly sophisticated and established solutions available today that can and must be employed to form this new composite technology. It is a great challenge to devise a process of incremental improvements here.

1.1.2 Sensor Network Applications

Wireless sensor networks are applied in a wide variety of areas, such as video surveillance, traffic monitoring, air traffic control, robotics, cars, home monitoring, manufacturing, environmental and wildlife habitat monitoring, industrial automation and military applications. A few sample applications are listed here:

Environmental Monitoring

The Sensor Web [Del02, DJJ+05] is a new class of geographic information system (GIS) developed at NASA's Jet Propulsion Laboratory consisting of a sensor network for environmental monitoring and control using live, real-time streaming data from deployed systems. Sensor Webs have been field tested in many areas including the Huntington botanical gardens for botanical conditions including soil moisture and temperature, Antarctica to monitor micro climate conditions for extreme life detection, and, in cooperation with the University of Arizona, in the Avra Valley storage and recovery project for flood detection.

A new tool for environmental monitoring and assessment of the world's ocean [FP05b, Gou05] is the Argo system that uses neutrally buoyant floats to observe the salinity, temperature and surface current flow of oceans. Floats descend to up to a few thousand meters in depth to take measurements and then resurface independent of central control. The data collected is relayed via satellite up-links to central data collection units on shore. Currently there are close to 2000 floats deployed and in continuous operation worldwide.

The James Reserve application targets the investigation of forest plant growth [CEH+01, SOP+04]. Here different types of wireless sensor network nodes and sensors are deployed in patches to collect ecosystem and environmental data. Organized in hierarchies, different protocols and communication paradigms are used as well as local computation and data refinement on cluster heads.

Wildlife Habitat Monitoring

In the Great Duck Island habitat monitoring deployment [MCP+02, SPMC04, SMP+04] about 150 sensor nodes were distributed on an island to observe the nesting behavior of birds. Nodes are located both inside underground burrows to observe motion patterns of the birds and the micro-climate as well as outside as weather stations. Data is transmitted in streams over a multihop network to an access point where it is buffered and uploaded to a central database repository off the island using a satellite link.

ZebraNet is an application for monitoring and tracking wildlife in a spacious outdoor habitat [JOW+02]. All nodes are constantly mobile, fixed to a collar around a zebras

neck and relay behavioral data as well as global positioning system (GPS) position fixes to a mobile data collection unit using flooding in a mobile ad hoc network [LSZM04, ZSLM04].

Building Management and Factory Automation

Different applications ranging from security applications [FRL05a, FRL05b], structural health monitoring [XRC+04] to building control [ACH+01, HHS+99, SDFV05] and factory automation [NKA+05] have been both proposed and built in prototype systems.

Military Applications

In a military vehicle tracking application wireless sensor network nodes are dropped from an unmanned aerial vehicle (UAV) [Hil04]. The nodes form an autonomous network and collaboratively detect vehicle movements using built-in sensors. An occasional visit by the UAV collects and transports the resulting data to a command post behind enemy lines.

An application to locate snipers uses acoustic measurements of distributed sensor nodes to compute the trajectory and origin of bullets using the muzzle blast emitted by a shot being fired [MSLS04, SBP+04]. The extensive computational tasks are performed on auxiliary hardware and allow to compute a location estimate within a few seconds.

1.1.3 Design Space of Sensor Networks

While there is no formal definition of wireless sensor networks, many research papers implicitly or explicitly contain a number of similar assumptions about properties of sensor networks. These commonly found assumptions could be interpreted as a de facto definition of what a wireless sensor network is. In particular, sensor networks are typically considered large-scale (thousands of nodes, covering large geographical areas), wireless, ad hoc, multihop, unpartitioned networks of homogeneous, tiny (hardly noticeable), mostly immobile (after deployment) sensor nodes that would be randomly deployed in an area of interest.

However, a reality check with existing applications of sensor networks that has been recently performed by Römer [RM04] reveals that these assumptions are often not met in practice. While there are definitely a number of applications that can be characterized in the above way, equally many applications deviate in various dimensions from the narrow definition mentioned above. In fact, the applications of sensor networks form an extensive design space with many dimensions.

- **Deployment** – one-time, incremental or as random activity
- **Mobility** – occasional or continuous performed by either selected or all nodes
- **Cost, size, resources, and energy** – very resource limited to unlimited
- **Heterogeneity** – a single type of node or diverse sets of differing properties and hierarchies

- **Communication modality** – apart from radio frequency, optical, acoustic, inductive and capacitive coupled communication have been used
- **Infrastructure** – different applications exclude, permit or require the use of fixed infrastructure
- **Network topology** – single hop, star, multihop, mesh and/or multi-tier
- **Coverage** – sparse, dense or redundant
- **Connectivity** – continuous, occasional or sporadic
- **Network size** – ranging from tens of nodes to thousands
- **Lifetime** – few hours, several months to many years
- **Other quality of service requirements** – real-time constraints, tamper-resistance, unobtrusiveness, stealth and others

In general, such an extensive design space, in contrast to the above narrow definition, complicates coordinated application development in various ways. One could argue that designing for the most restrictive point in the design space, e.g. minimum node capabilities, highly mobile, etc. might be a solution. However, often there is no such global "minimum" and it is often highly desirable to exploit the characteristics of the various points in the design space. A further important implication is that most likely no single hardware and software platform will be sufficient to support the whole design space or large portions thereof.

In many cases, an application can be associated with a single point or a small region in the design space throughout its lifetime. However, it is also quite possible that application characteristics (and thus the point in the design space associated with that particular application) could change dynamically during the lifetime of an application. For cost reasons it may also be necessary or advantageous to reuse a sensor network for different applications with different characteristics. In some cases a sensor network might even execute multiple applications concurrently. Such applications require that design decisions cannot be done statically during the design and development phases, but may have to be revised or adapted during the runtime of the system.

1.2 Design and Development of Wireless Networked Embedded Systems

It was envisioned that sensor networks could be used for complex monitoring and control tasks, where the output of many sensors with different modalities would be processed, correlated, aggregated, or fused using advanced distributed signal processing functions [Spm02]. Also, it was anticipated that the resulting information would be used to control certain aspects of the sensor network, e.g. feedback loops for adaptive sampling or even to control distributed actuators. Taking these visions into account, existing application prototypes are often surprisingly simple. The measured physical signals typically

display a low spatial and temporal variability, which makes them easy to handle. Also, distributed processing of sensor data in the network is often rather simple, e.g. averaging. In many cases, raw streams of sensor data are passed via a gateway to a back-end system as input for human users [SPMC04]. Information obtained from sensors is rarely used to control sensing or actuation.

Despite this low application complexity, the development of a running prototype with more than a few nodes is a surprisingly complex task that requires highly-skilled system programmers. In excess to the hard- and software involved, non-functional constraints have to be observed. In real-world experiments researchers have been struggling to achieve setups of more than a handful of nodes. Examples such as Great Duck Island [SPMC04], shooter localization [SBP+04] or the extreme scaling mote [JPC05] exist, but have shown hard to get to run and manage, especially due to the setup in a realistic physical environment. The construction of a running system that delivers predictable results in compliance with application requirements today relies on factors such as significant amounts of manpower [HBAB04] and money, individual skill of developers [CEH+01], many iterations of the system design and implementations [MCP+02], and also a certain amount of luck [SOP+04].

Distributed simulation, e.g. TOSSIM [LLWC03], is a valuable tool in the development process; it allows to study system-design alternatives in a controlled environment. The problem with simulation is that assumptions have to be made [KNG+04, MC03], simplifications have to take place [KNE03], or models are simply wrong [MC03]; this inevitably leads to a gap between reality and the simulated, virtual world. In addition, simulation results are known to not always be consistent across different scenarios or tool chains [CS02]. Already rather subtle differences of the simulation models and the real world can make the difference between a working and a broken implementation [HBE+01]. For example, many wireless radio simulations assume a spherical communication range. However, measurements in a network of about 100 sensor nodes revealed that the radio communication range is far from spherical in practice [GKW+02]. This can lead to asymmetric or even unidirectional communication links, which can easily break algorithms that assume symmetrical links.

1.2.1 Sensor Network Systems Design

System design typically involves to partition the complete application into a number of sub-components both in hardware and in software, which together provide the required functionality of the system. Whenever possible, it is desirable to reuse existing components. The developer is then faced with the challenge of selecting appropriate components from potentially large set of available components.

Often the necessity to embed nodes into a tightly coupled physical environment without possibilities for infrastructure access or backup facilities demand highly optimized system concepts, only provisioning the resources really necessary for a solution. Selection of appropriate system components, e.g. the sensor node hardware, then requires the estimation

of the minimum resource set necessary for correct, reliable and predictable operation. Typically, this issue is addressed today by over-provisioning additional resources [SMP+04] and functionality in the design phase that is stripped in a later phase of prototype [RAdSJ+00]. While this is certainly a valid approach to overcome a certain amount of unknown obstacles in an early development phase, it does not at all support any form of validation of an application (correct-by-design) or guarantee a concise use of resources.

Already a long-time issue in domains such as integrated circuit (IC) and embedded systems design, design-for-testability has not been an issue for sensor networks yet. As discussed in section 5.1, testing and debugging must already be considered during the design in order to provision necessary resources, e.g. memory, communication bandwidth, non-volatile storage for logs. Also, issues such as calibration and verification may have to be considered already during system design.

The design space of sensor networks presented earlier indicated that, in general, there are no one-size-fits-all solutions for particular tasks. Rather, a variety of different solutions will be needed to cover significant portions of the design space. Hence, a large set of reusable components must be expected to be available in the future for the developer to select from, e.g. different sensor nodes with different amounts of resources. Every solution would have properties that are adapted to a certain region in the design space, e.g. a time synchronization service that supports mobility but provides low precision .

It would be highly desirable to support the application developer in this often complex selection process with appropriate concepts and tools to narrow down the set of design alternatives. Promising approaches of such a semi-automatic design space exploration have recently been explored for embedded systems [KTZ05] and specific applications such as packet processors [TCGK02] and as we have proposed, also for distributed wearable computing [ABD+04]. In respect to the special requirements put forward by the sensor network case, the models used have to be refined to support unreliability in their resources such as breaking links due to radio interference or node failures due to a limited energy supply. Furthermore the collaborative nature of many sensor nodes collectively providing a common system function cannot be incorporated easily into a typical one-to-one mapping of tasks to available resources.

1.2.2 Prototype Development – Programming and Debugging

Once the application has been partitioned into functional components during the design phase, some existing components may be reusable without modification, other existing components may have to be modified, and some components may have to be developed from scratch. As far as software components are concerned, modifications and new developments require programming work.

Given the required level of abstraction, the developer should be relieved as much as possible from programming individual sensor nodes at the system level, including distribution issues by providing appropriate high-level abstractions that allow the programmer to think in terms of the application. However, the use of abstractions often introduces performance and resource overheads, which typically is a major issue in sensor networks. For

example, a commonly used design principle in sensor networks is cross-layer interaction, where knowledge about system details is used in the application and vice versa to achieve efficient implementations. Since programming abstractions effectively shield application development from system details, it is not clear whether cross-layer interaction and the use of programming abstractions can be integrated in meaningful ways.

Hence, a major challenge is to find programming abstractions which are appropriate both with respect to the developer skills and with respect to efficiency.

The task of programming is tightly integrated with testing, debugging, and profiling, e.g. identifying performance bottlenecks and resource dissipation. In particular, all these tasks should – as far as possible – be performed at a similar level of abstraction. With the "generic role assignment" abstraction proposed by Frank [RFMB04, FR05], for example, role specifications to examine what is going wrong are mapped to C code. Although a traditional source-level debugger could be used to examine this C code, the value of this abstraction would be significantly degraded by doing so. Rather, the developer should be able to examine what is going wrong at the abstraction level of roles. Little work has been done so far to support a consistent level of abstraction across programming, testing, debugging, and profiling.

1.2.3 Sensor Network Deployment and Testing

Leading to meaningful results, sensor networks need to be deployed in an actual physical environment where sensor/actor interaction can take place at realistic scale and the environment for wireless communication is matching both in dimensions and properties. Moving on from programming and debugging of the single nodes to system testing in such a deployment and testing phase, two different goals can be distinguished: (i) testing in order to scale up the number of nodes in an initial deployment and (ii) testing in an actual physical environment, i.e. field testing to incorporate the influence of the real-world.

In a first phase, a sensor network will have to be initialized, either node by node, or all at once in a synchronized fashion [KMW04]. Subsequently a "steady" or operating state will be reached where all nodes and services are running. Both phases require detailed observations of the system behavior and the interaction with the environment, inflicting only minimal impact on the sensor network and it's deployment environment using a lightweight infrastructure. This infrastructure should support on- and offline monitoring, logging, distribution of software updates and commands as well as the control of operating parameters.

Tools such as distributed simulation [LLWC03] are of great benefit in an early design phase, offer good observability, but lack in respect to a realistic environment. Simulations will not be able to attach peripherals such as sensors and use simplified physical models that cannot capture many real-world influences, e.g. broken radio links, sporadic system errors. Lessons from experimentation, such as outdoor sensor networks [MCP+02, SOP+04], university class projects [HBAB04], or large-scale testbed deployments [CE04], have all required direct and reliable access to the WSN devices involved. Out of this necessity,

many different techniques have been developed to attach temporarily or permanently to WSN devices: Layered architectures with different scales of devices and hierarchical networks [PK00, HHKK04], serial multiplexing units mounted onto a table, as a semi-mobile laptop unit, or mounted into a ceiling array [CBE03, GEC+04, CE04], direct ethernet attachments such as the MIB600 programming board for the Berkeley Motes [HC02], used in the moteLab and sMote testbeds. All of these have in common that they require wired connections, although it is common to replace the direct serial cabling by another medium to allow multiplexing. In the case of large mobile or outdoor applications, it is very hard if not impossible to connect a significant amount of devices, whereas setups like a fixed ceiling array can only offer a very synthetic environment for experimentation. This limits the applicability of these approaches to cumbersome testing scenarios where fixed infrastructure is put in place for the duration of testing and afterwards removed.

1.2.4 A Vision of Wireless Sensor Network Application Design

In order for sensor networks to become a tool that can be widely used for real-world applications, the development process has to meet a number requirements. If the cost of the pure sensor node hardware will actually drop to few Dollars or even Cents per node, the overall cost of using a sensor network will be dominated by development and maintenance expenditure. Hence, it is very important to reduce the manpower and necessary skills that are required for putting a sensor network in place and maintaining it over time. As an ideal, application-domain experts, e.g. the end-users of the WSN application would be able to develop, customize, and run a wide range of sensor network applications with only little or moderate support by system experts.

This overall goal has far-reaching implications. For example, the development process should be supported by appropriate abstractions at the application level to specify, examine, and verify application requirements and behavior. These abstractions must take into account the specific characteristics of sensor networks (see section 1.1). Also, modular reuse of existing hardware and software components must replace current practice of costly custom design and development processes. The developer should also be appropriately supported in selecting appropriate hardware and software components, with respect to application requirements, from a large available set. It is also important that predictable system behavior that is compliant with the application requirements can be achieved without tweaking and adjusting bits at the system level.

This might all sound somewhat far-fetched or illusionary. However, considering an analogy to the use of personal computers might help to put the above issues into a more realistic light. Purchasing a personal computer (PC) typically involves an application analysis, e.g. used for gaming, word processing, etc. to bound the functions and performance of the required hardware. With some help from a sales person, the client can select from a large palette of readily available hardware components, e.g. core computer, input and output devices, extension cards which can be put together relatively effortlessly even with little support from a systems expert. Only a very small fraction of PC users program at the system level, e.g. using C, C++, or other system programming languages. Rather, most

"application developers" use readily available tools or components that allow development at the abstraction level of the application, e.g. graphics renderer, symbolic math package, statistics tools, spreadsheets.

1.3 Contributions

With this thesis, we want to contribute to the design and development of wireless networked embedded systems by providing a systematic approach for the development, test and deployment of distributed applications on such devices. In order to achieve this goal we present a vertical slice of the design space concerned and discuss relevant questions encountered in different phases of the design and development process.

With the increasing development of large-scale wireless sensor network applications, the coordinated development and deployment of sensor network devices are becoming an issue of increasing importance. Independent researchers have reported that when moving away from the engineer's desktop and beyond numbers of 10–20 nodes, design, development, deployment and testing become increasingly hard, and simulation will not solve all problems encountered. While algorithms, system models, device architectures, and programming abstractions have been investigated for quite some time now, not much has been achieved in the area of deployment support or even a concerted design, development and deployment methodology that allows for stepwise refinement and reliable monitoring of systems.

Coordinated methods and tools for wireless sensor network deployment are missing today. With our approach of a deployment-support network presented in this thesis, we push the limit for large-scale prototyping from simulation and virtualization to coordinated real-world deployment.

Specifically, the contributions of this thesis are as follows:

- Functional and qualitative requirements of a location management service for wireless sensor networks based on measurements and simulation have been developed. We present one of the earliest algorithms for the distributed computation of node location.
- A novel platform for fast-prototyping of wireless sensor networks has been developed. This platform has successfully served numerous researchers, among ourselves, as an underlying infrastructure for experimentation and education.
- The BTnode platform has been used to develop multihop networks and topology control mechanisms for Bluetooth scatternets. To our knowledge the experiments presented in this thesis are the largest connected Bluetooth scatternets reported to date.
- The concept of a deployment-support network as a powerful tool for the development, deployment and validation of wireless sensor networks is presented in conjunction with experimental evidence of the feasibility of the approach.

The contributions presented in this thesis have also been published in the following refereed publications [SRB01, PEW+02, BKR03a, BKR03b, PEW+03, BKM+04, Beu04, BDH+04, ABD+04, Beu05, BDMT05, BD05, DBM05, NBD05a].

1.4 Contents and Structure

In this thesis we identify functional and qualitative requirements of applications and services for wireless sensor networks based on measurements and simulation using the example of a location management service presented in chapter 2. Chapter 3 introduces and discusses design criteria for wireless sensor network platforms. A comparison of the relevant platforms developed in the field motivates a design rationale for modular platforms which we use to develop the BTnode platform, a distributed environment for prototyping ad hoc and wireless sensor networks. The BTnode platform is subsequently used to develop multihop network topology construction mechanisms based on connection-oriented Bluetooth scatternets in chapter 4. Different approaches are discussed in conjunction with the system prerequisites and peculiarities of the platforms used, algorithms and experimental evidence. Out of the experience gained in the implementation work presented, we motivate a full life-cycle support for sensor networks and develop a methodology for the design and development of wireless networked embedded systems based on the deployment-support network in chapter 5.

2

Location Management in Wireless Communication Systems

Location management both in time and space has been identified as a key technology for the successful deployment and operation of context-aware sensor network services and applications by different visions [RAdSJ+00, KKP99, ECPS02, Com01]. The benefit of location information is not limited to the subscriber of a network or to network operators but will enable every wireless enabled device to become a meaningful instrumentation probe. Today, technicians gather spatially distributed environmental information by driving to specific sample sites and making measurements, a time-consuming and inefficient solution, or by installing costly fixed infrastructure in often inaccessible target areas. Spatially distributed sensor data without an appropriate reference context, e.g. a time stamp, and a set of multidimensional coordinates within a given reference system, is next to useless. While the global positioning system (GPS) offers good solutions for localization in an outdoor environment, no such option exists for an indoor setting or lightweight approaches such as wireless sensor networks.

In this chapter we want to give an overview of the issues connected to running a location management service on networked wireless embedded systems. This exemplary service is then used break down and illustrate the cross-layer requirements on WSN systems that arise when using local positioning algorithms as the basis for a system-wide location management service. Apart from general observations, definitions and algorithms applicable to many wireless communication systems we are interested in the peculiarities of the wireless sensor network case where simplicity, robustness and energy awareness are of primary importance as well as the implications on the design and development of WSN systems.

2.1 Location Management

In geography, location is a position or point in physical space expressed relative to the position of another point or thing. A real location can often be designated by cartesian

coordinates. On the earth, the geographic coordinate system can be used to specify the position of any location. In this sense, navigation is the determination of position and direction on the surface of the earth. The concept of location is not limited to the geographic representation of physical location with sets of latitude, longitude and altitude but also applicable to symbolic location in a non-geographic sense such as location in time, in a virtual information space or a geometric abstraction such as a data structure or the graph of a network. Postal zip codes and telephone numbers are a good example of abstractions containing designated location information.

There are several different branches of navigation, including but not limited to

- **Celestial navigation** – navigation by observation of the sun, moon and stars
- **Pilotage** – using visible natural and man made features such as sea marks or beacons
- **Dead reckoning** – using compass and log to monitor expected progress on a journey
- **Inertial navigation** – using integrating accelerometers mounted on gyroscopically stabilized platforms
- **Waypoint navigation** – using electronic equipment such as radio navigation and satellite navigation system to follow a course to a waypoint
- **Position fixing** – determining current position by visual and electronic means
- **Collision avoidance** – using radar or similar methods.
- **Radionavigation** – employing (the time of flight of) electromagnetic waves

All of these methods use to some extent fixed infrastructure or elaborate and specialized equipment which is not suitable for sensor networks.

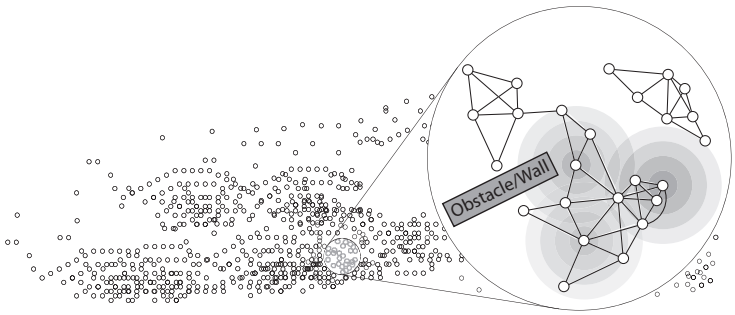

Figure 2-1
A view on a typical wireless sensor network topology scenario with highly clustered nodes, scarcely populated areas, obstructed and separated regions. The limited radio range allows only for a few overlapping links, requiring multi hop communication.

The positioning problem in wireless sensor networks can be viewed as a general distributed sensor problem, with sensors that can discover other nodes, estimate ranges between nodes and such that serve as position references. Furthermore, the maps and data-sets that are to be used in conjunction with the position information can also be viewed as a form of sensor information, only that in this case the communication network and storage resources are the means used to provide this information to context aware services. In densely populated wireless sensor networks interactions between nodes are abundant. It is therefore necessary

to extract and combine the appropriate information in a suitable way to render it usable. To a certain degree, a notion of living with errors has to be adopted, since the formal problem of distributed location is hard and resources available per node in a wireless sensor network are finite. Important requirements for positioning in wireless sensor networks are a distributed approach, that minimizes computational and especially communication overhead and is robust enough to survive unreliable situations, such as occasional network partitioning or node failures.

2.1.1 Radionavigation

Radionavigation techniques are commonly based on measuring the received signal stength indicator (RSSI), time of arrival (ToA), time distance of arrival (TDoA), carrier phase and code measurements, ultra-wide band (UWB), ultrasound and even visible light pulses of a signal or the angle of arrival (AoA) of a radio signal [Beu04]. Three or more independent signal measurements may be used to solve for a triangulation solution. While calculating direction using AoA requires additional antenna hardware that needs to be precisely calibrated, the systems based on the time of flight of electromagnetic waves require very accurate timing measurements and thus high synchronicity. Systems using this principle are abundant and have been developed a long time ago. Among them are Lorentz (Germany, 1930s), VOR (International, still used in aircraft today), GEE (British Royal Air Force, World War II), LORAN (American, World War II, long range navigation for navy) and GPS (American, 1978, satellite navigation system, see section 2.1.2.1).

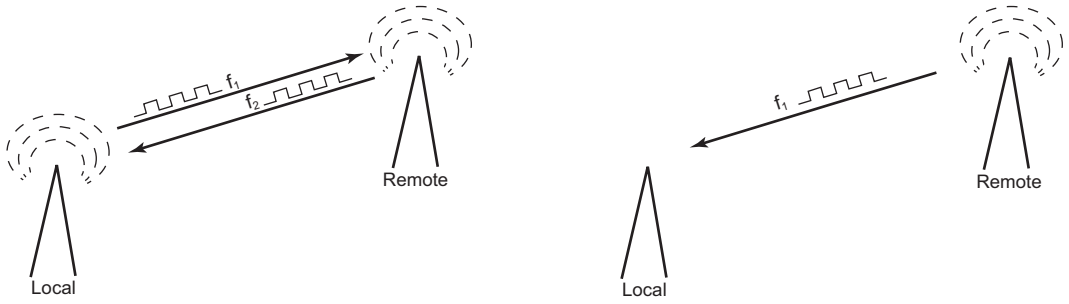

Figure 2-2
Active radionavigation uses a bidirectional relationship between the remote and local nodes. The up- and down-link signal shown by f_1 and f_2 must not be the same. Passive radionavigation systems rely on the continuous availability of timed pulses from fixed beacons.

Radionavigation can additionally be divided into two classes, active and passive radionavigation [Log92]. In active radionavigation the navigation receiver broadcasts a signal f_1 to a distant transmitter or base station that bounces the signal back at the receiver, possibly at a different frequency f_2, as seen in Figure 2-2. For example, the range between the receiver and distant transmitter in active radionavigation is computed by half the time of flight multiplied by the speed of light $c = 3 \cdot 10^8$ m/sec. In passive radionavigation a distant transmitter sends out a series of precisely timed pulses f_1. The navigation receiver picks up

these pulses, measures the time of flight and multiplies by the speed of light analogous to the previous solution.

A radionavigation system may or may not be able to communicate and thus share data used to solve for the position. Radionavigation systems that cannot communicate have to rely solely on their ability to detect and identify an electromagnetic signal. They can employ methods based on signal strength, doppler shift and AoA measurements. If communication is available, data can be modulated onto the navigation signal or be available on a secondary communication carrier. This is common for ToA and TDoA solutions. Combinations of modulated data and secondary links, for example in differential GPS, are possible too, where the secondary communication link is used for correcting and refining the solution derived by the GPS system.

In a networked environment of many navigating nodes, cell density, the distribution of nodes, the ability to share information, terrain and physical obstruction play a major role in the accuracy derived by the system.

From this perspective positioning techniques for wireless sensor networks have to be distributed, local and be based on the network. The exact implementation of each location technique depends by large on the underlying communication network and it's capabilities. The required abstraction of location must be defined from the viewpoint of a WSN application and the intended usage of the location information produced by such a service. It appears quite logical to try to integrate network services such as positioning with the data traffic in the network, as to reduce the amount of overhead connections and traffic. A hybrid, or cross-layer approach integrating networking and positioning is therefore highly desirable.

2.1.2 Related Work

2.1.2.1 Global Positioning System

The global positioning system (GPS) [Log92] was developed by the U.S. Department of Defence to be used to determine one's exact location and precise time anywhere on Earth at any time. It relies on 28 satellites orbiting on six different planes so that a minimum amount of four can be seen from any point on the planet. Each satellite transmits its exact position and its precisely synchronized on-board time in a spread spectrum signal. A GPS receiver measures the signal transit times between its point of observation and at least four different satellites with known positions to be able to solve for the four unknowns longitude x, latitude y, altitude z and time deviation Δt [SB97]. There is no reverse up-link communication necessary between the transceiver and the satellites.

Today GPS receivers are highly specialized, high performance, integrated computing devices that can be integrated into handheld devices and mobile phones. Typical performance figures of a receiver for continuous operation are less than 160 mW power consumption, 3 m accuracy, 20 ns timing precision, 41 sec cold start, 3.5 sec hot start and a 4 Hz update rate at a size (without antenna) of about 1.9 cm^3. GPS can be augmented with differential

navigation data, post processing or inertial sensors to achieve even higher performance useful for vehicle navigation, land surveying and civil engineering applications. In differential GPS the error of a stationary receiver is transmitted to a mobile receiver and used to correct it's position estimate under the assumption of similar signal perturbation at both receivers. Consequently, the resulting positioning accuracy is improved, the closer two receivers are co-located.

So far, the application of GPS to sensor networks is limited firstly because of the considerable bulk and high power consumption but mainly because of the requirement for direct line of sight to the satellites incorporated in the navigation solution. Furthermore the cost (both economical and in resources utilized) of integrating a GPS receiver is prohibitive for large scale WSN applications.

2.1.2.2 Cell-Based Positioning Techniques

Mainly driven by the U.S. Enhanced 911 mandate that requires to be able to locate mobile phones up to about 50 to 100 m, but also by emerging commercial location based services, cell-based positioning techniques have been under detailed investigation [Zha02, CS98, DMS98]. Common to all these techniques is, that they rely on infrastructure put in place and configured to be used for positioning. Similar to GPS, the wireless link to base stations can be estimated and given a sufficient amount of base stations (overlapping cells) a triangulation solution can be computed. Key issues here are the inter-operation of multiple base stations, inefficient surveying prior to deployment and scalability problems in centralized services since in most cases, the mobile cellular handsets cannot be modified.

The Active Bat system [HHS+99, ACH+01] is heavily dependent on dense ceiling mounted infrastructure that uses combined radio frequency (RF) and ultrasound ranging to achieve 3 cm position accuracy on custom mobile devices, the so-called "bats". In contrast the Massachusetts Institute of Technology (MIT) Cricket location system [PCB00] is highly decentralized and does not rely on central control and computation. The use of combined RF and ultrasound ranging and identification however is similar. Fixed beacons inside buildings distribute geographic information. AHLoS [SHS01] is a system using ultrasound and RF and does not rely on an infrastructure setting.

The approach used by RADAR [BP00] goes even further in mapping out the characteristic signal propagation in spaces with wireless local area network coverage to be used for positioning of mobile users. Individual maps per base station are then overlayed to determine a mobile units position using computationally intensive, statistical methods performed on a central resource.

The GPS-less system [BHE00] is an RF-based positioning method that requires a number of nodes to be placed at known positions that form a regular mesh and transmit periodic beacon signals containing their respective positions. This algorithm does not need any accurate distance measurements; it only uses the information if a node is in range or out of range of a certain beacon node. It has the advantage of simplicity, scalability and low power consumption. However, it is coarse grained and needs a considerable amount of

infrastructure. It has been extended to incorporate adaptive beacon placement [BHE01] and scalable, ad-hoc deployable RF-based localization [BBEH02, Bul02].

Doherty [DPEG01] proposes a method based exclusively on connectivity-induced constraints. Know peer-to-peer communication in the network is modeled as a set of geometric constraints on the node positions. The geometric constraints can be of an angular (node in/out of sight for e.g. optical transmission) or radial nature (in/out of range for RF, ultrasound or similar systems). The problem is solved using a linear program (LP), evaluating every single geometric constraint. Additionally, a method for placing a rectangular bound around the possible positions is presented.

The algorithm presented in [CHH01] uses distance measurements to estimate positions. As long as there are no anchor nodes in view, unknown nodes build up their own local coordinate systems. The first unknown node just assumes to be on position $(0, 0, 0)$, a second will be placed on the positive x-axis at position $(r, 0, 0)$. When a first anchor node comes in view of the locally built coordinate system, a translation of the local system will be required. For a second anchor a rotation and eventually a flip for the third (in three dimensions a second rotation will follow before an eventual flip) is required.

2.1.2.3 Tagging with RF-ID

Initial radio frequency identification systems (RF-ID) were meant to be passive systems confined to a very short radio range only. Powered by the signal emitted by a transmitter RF-IDs reflect a signal that can be individually identified by a usually collocated receiver. The system employed is essentially cell based localization, but on a very local level, e.g. well under a meter [NLYP03, SF03]. Cheap and easy to embed, e.g. in printable product labels or tags this technology is ideal for simple high volume applications. More complex active tags allow for data storage and simple computing operations like authentification and sensor data aggregation.

2.1.2.4 The Lighthouse Location System

The Lighthouse Location System for Smart Dust [Röm03] is based on direct line of sight between fixed infrastructure laser transmitters and the mobile unit. Each transmitter emits a laser beam that is rotated in two perpendicular axes, thus scanning a whole room. The mobile unit registers the phase and duration of the light flashes and uses this information to intersect three hyperboloids each in reference to the transmitters position.

Unique to this approach is, that high precision can be achieved with relatively low system, communication and computational complexity on the sensor nodes. This and the line of sight requirement and the extensive calibration necessary prior to usage make the Lighthouse location system an elegant but rarely applicable solution.

2.1.2.5 Abstractions

In wireless sensor networks naming data and not nodes and the organization around spatial and temporal coordinate systems require appropriate abstractions and programming models for location context to be developed. First approaches in this direction show that such

systems can be viewed as a the tuple space model, distributed databases or even loosely coupled parallel computing structures [BGS00].

2.1.3 Location Abstractions

2.1.3.1 Networks as Graphs

For the geometric abstractions that we will use we will assume the network to be a set of vertices $V = \{v_1, v_2, \ldots, v_n\}$ and edges $E = \{e_1, e_2, \ldots, e_n\}$ that describe the topology of the network by means of a graph $G(V, E)$.

In the case of range estimates being associated with a certain edge this can be done by assigning a weight $w(e)$ to every edge $e = (v_p, v_q)$ connecting the vertices v_p and v_q. For differential range estimates, where two endpoints derive individual range estimates from their own, local viewpoints, the local $w_p(v_p, e_{pq})$ and remote $w_q(v_q, e_{pq})$ range estimates can be combined using a function to assign a single weight

$$w(e_{pq}) = func(w_p(v_p, e_{pq}), w_q(v_q, e_{pq})) \tag{2.1}$$

to the edge connecting the vertices v_p and v_q. This function can be a simple averaging function or a more complicated filter based on an estimation history and requiring extensive data exchange between nodes. In a wireless sensor network the vertices of the graph represent the nodes or devices and the edges represent the wireless communication links. While this abstraction is quite suitable for link-layer abstractions, it is not always appropriate for lower layers, e.g. broadcast medium access layer (MAC). Commonly used are also visibility graphs such as a unit disk graph (UDG) that only draw an edge between two vertices if the length (weight) of the edge is below one.

Degree – The degree $\deg(v)$ or valency of a vertex v is the number of edges incident to v (with loops being counted twice).

Scope of a Node – The scope of a node is defined by the ρ-neighborhood of a vertex $v \in V$ as $\Gamma_\rho(v, G)$ where ρ denotes the maximal hop distance from v on the Graph G.

$$\Gamma_\rho(v, G) = \{w | \mathrm{dist}_\mathrm{G}(w, v) \leq \rho\} \tag{2.2}$$

When talking about physical location in the traditional way we usually view points as three dimensional coordinates (x, y, z) in a cartesian reference coordinate system. Of course many other transformations to other coordinate systems like polar coordinates are possible too but we will consider the cartesian system here. In a three dimensional cartesian system the Euclidean distance between two points v_p and v_q is defined by

$$\mathrm{dist}_\mathrm{G}(v_p, v_q) = \sqrt{(x_p - x_q)^2 + (y_p - y_q)^2 + (z_p - z_q)^2} \tag{2.3}$$

While there are also other metrics [dBvKMOS00] e.g. the manhattan distance, we will continue to use this measure as a basis for the trigonometry introduced in later sections.

Common to all notions of location is, that the individual locations are all relative to each other, meaning that they depend on a predefined frame of reference. This leads to a differentiation of the relative and absolute positioning cases.

2.1.3.2 Relative Position

Vectors linking two points give information on their position relative to each other. When no reference points are given, the solution can be rotated and mirrored through an arbitrary axis. Each position added to a system solution reduces the problem one degree of freedom at a time. A minimum of four nodes are necessary to be able to unambiguously orient a geometric position in three dimensional space (three nodes in two dimensional space) since when using one reference per axis of the coordinate system, two ambiguous mirrored solutions are possible (see Figure 2-3).

A system of many known positions that has no reference location and/or orientation is only fixed in itself, not in it's position in space: If only the position of one point and no orientation is given, the system can be mirrored and rotated through any axis leading through this point. The translatory movement is prohibited by this first known position. When a second position is introduced into the system the rotation of the system is further restricted to the axis through these two points. A two dimensional problem would thus still have two possible solutions. A third known position finally fixes the system in two and three dimensional space.

Free Node – A free node v_f has no a priori knowledge of location, but seeks to obtain a position estimate by the positioning methods described later in this section.

Settled Node – A settled node v_s initially was a free node, but has calculated a position estimate and can thus serve as an additional position reference to other nodes in the network.

A relative position can only be given in respect to other points resolving the distances and the geometric configuration, e.g. the topology (see Figure 2-3). The minimum requirement for relative topology discovery is that all nodes that are to be considered in such an algorithm must be connected and able to identify each other. Ranging and the exchange of data between nodes further allows to weight an originally unweighted topology graph.

2.1.3.3 Absolute Position

An absolute position is given in respect to an inertial system and reference points in this "inertial system". It allows to determine positioning information of disjunct systems independently, in reference to the same point in the inertial system. Points that know their position before the application of a navigation technique are referred to as anchor nodes.

Anchor Node – An anchor or beacon node v_a has knowledge of it's location, either through prior configuration, or an external reference source, such as a GPS receiver. It is important to note, that anchors do not derive position through means offered by a network positioning mechanism. They can thus serve as position references.

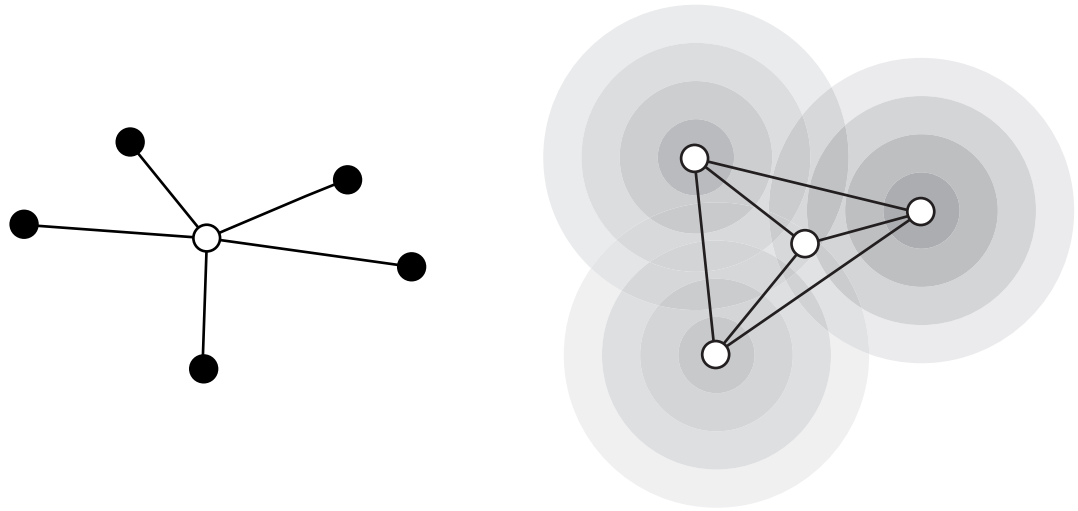

Figure 2-3
The absolute position (left figure) of a free node (white node) is defined in respect to multiple reference positions (black nodes). Relative positions (right figure) can be discriminated against other nodes in a local reference system (wire frame topology) only.

2.1.3.4 Location in Space and Time

Usually mere (x, y, z) coordinates by themselves are not meaningful for context aware system services and other information needs to be associated with these position fixes. The most straightforward extension is to introduce time as fourth dimension to be able to specify where and when a certain event took place resulting in sets of (x, y, z, t) for each complete position fix.

Such a four-dimensional fix can then be used to put subsequent or collocated events into a context frame. For this a reference frame is necessary as time and location information is useless by itself. In a relative reference frame different parts of a network would compare their data relative to each other. This enables to discriminate orientation, distance, time difference, speed and acceleration between nodes of a network but not to an external reference frame. Depending on the granularity required for the resolution of such data different requirements on the accuracy of such position fixes can be defined. This allows context aware applications to behave differently when evaluating not only the position fix but also the desired and achieved accuracy.

When using position information in reference to a geographic map or a global time reference the context information can be extended. Here already a single position fix can be put in context of this reference frame vs. a minimum of two position fixes that are necessary for the relative case. For an earth centered view different reference ellipsoids and geodetic datums that account for the specific shape of the earth [SB97]. In addition every observer of a geometric system can act as an absolute reference point in an inertial system, namely their own inertial system such as it is common on inertial platforms.

2.1.3.5 Reference Systems

Absolute positioning allows to orient the nodes of a network on a map that can be any variable or set of variables representing or assigning values to a geographic location or region, from a single point to an entire planet. This must not necessarily be a two-dimensional rendition, such as a paper road map that we are most familiar with today but can be any data structure that defines a reference frame for more than just one node.

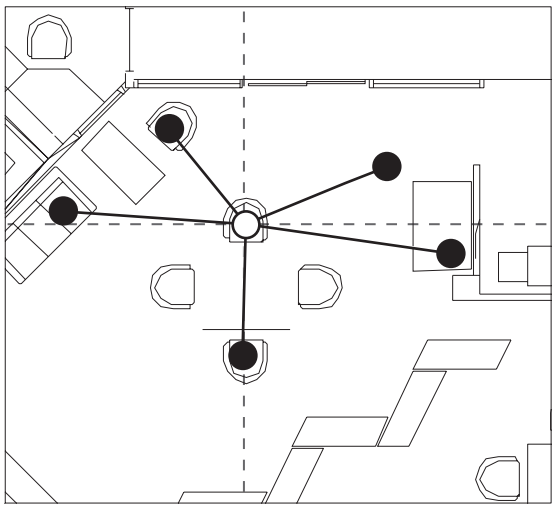

Figure 2-4
Maps as reference system: A map is a simplified depiction of a space, a navigational aid which highlights relations between objects within that space. A map can be used in conjunction with absolute position fixes (white node) and reference positions (black nodes) and so enables the contextual interpretation of location information in human readable form.

A map projection is any of many methods used in cartography to represent the two-dimensional curved surface of the earth or other body on a plane. The term "projection"here refers to any function defined on the earth's surface and with values on the plane, and not necessarily a geometric projection. Map projections can be constructed to preserve properties like area, shape, direction, bearing, distance or scale, though not all of them simultaneously.

It is thus required to be able to transform position estimates into the appropriate reference system and precision, depending on the context, in order be able to support meaningful location information. This requires access to stored data such as map datums and computational resources to perform the conversions.

2.1.4 Network Based Navigation Techniques

Positioning using navigation techniques generally consists of three components:

- Identification and data exchange
- Measurement and data acquisition
- Computation to derive location

The various approaches partition these tasks differently across their system components and in some cases no or only unidirectional data exchange between nodes is used. The identification of distinct nodes in the neighborhood with the use of the radio is usually a

task attributed to the MAC protocols and is a key requirement for a positioning system. This means, that nodes must have a distinct address as opposed to address free routing schemes proposed by Estrin et al. [EGHK99, IGE00], and capabilities to discover the addresses of surrounding nodes. Combined with storage, sorting and communication capabilities this enables a distributed set of nodes to perform a network topology discovery, i.e. they can search for neighbors and exchange this data. Based on the communication capabilities of the nodes, topologies and paths for multi hop routing of data can be set up subsequently and.

Measurements can either be performed to estimate the distance between nodes or the angle of an incoming signal (see section 2.1.1). This can be performed on active communication links or analogous to the neighborhood identification described earlier. Depending on the radio system, this can be performed in parallel to multiple nodes at once, simultaneously with data transmissions, or using a private, time-shared medium access. Moreover, different system may demand to iterate measurements multiple times, depending on the architecture and characteristics of the radio system.

The important thing to note is that there are always (i) errors in these measurements, that (ii) individual measurements are not independent of each other and (iii) individual measurements are strongly influenced by the surrounding environment and the transmission system used. The availability of any one of these physical variables depends by large on the transceiver and antenna architecture available and is beyond our scope. We will therefore refer to these nonlinear measurements as range estimates $\hat{r}$ independent of their source for the remainder of this chapter. Furthermore, we will be assuming a distributed approach where each free node is performing all tasks required to calculate a position estimate locally, i.e. identification of neighbors, range estimation and computation as opposed to centralized approaches where an external observer performs some or all of these tasks for all free nodes in a network [HHS+99, BP00].

2.1.4.1 Hyperbolic Trilateration

Using any one of the range estimation techniques listed earlier, the geometric position can be computed using hyperbolic trilateration. Here three independent range measurements with respect to globally referenced anchor nodes are used to compute the intersection of three circles (see Figure 2-5). The inputs are the coordinates of the reference nodes $v_i = (x_i, y_i, z_i)$ and the respective range estimates $\hat{r}_i$

Hyperbolic trilateration is essentially the core idea behind most methods to calculate geometric position and is used in variations in most systems, for example in the GPS as described in detail by Capkun [CHH02]. As defined in section 2.1.3 at least three range estimates are necessary to solve for all ambiguities in the two dimensional as well as in the three dimensional case. The mathematics of the overdetermined problem are explained in detail in section 2.1.5.

2.1.4.2 Triangulation

Using the trigonometry laws of sines and cosines the angles of an incoming signal $\hat{\alpha}_i$ can be used to compute a triangulation solution (see Figure 2-5) similar to the method used for hyperbolic trilateration. These angles can either be measured at the free node or at all reference locations. In the latter case they need to be oriented correctly, implicating either a precisely aligned infrastructure or supplemental measurements between adjacent references. Apart from this alignment problem, AoA measurements require extensive hardware, usually with multiple sectored antennae, making it currently unsuitable for most WSN applications.

2.1.4.3 Multilateration

In the case of dense anchor node populations a maximum likelihood (ML) estimation can be performed. The prerequisite for this multilateration technique is, that at least three reference nodes are visible at every free node when performing the position calculation.

Using this multilateration method the position of the free node $v_u = (x_u, y_u, z_u)$ is estimated from three or more reference nodes (the one-hop neighborhood) such that the difference between measured and estimated range is minimized for every range estimate $\hat{r}_{i,0}$ incorporated into the solution (see Figure 2-5).

$$\min \sqrt{(x_i - x_u)^2 + (y_i - y_u)^2 + (z_i - z_u)^2} \quad \forall \hat{r}_{i,0}; i = 0 \ldots n \tag{2.4}$$

An iterative algorithm applicable for the multi hop case of this technique is explored in [SHS01, SPS02]. The geometric constraints involved in multilateration can also be formulated as a LP such as proposed by Doherty [DPEG01] where angular and radial constraints are combined in one model. The drawback of this method is, that it has considerable amounts of communication requirements, especially as an LP scales with the square size of the network [vGR05].

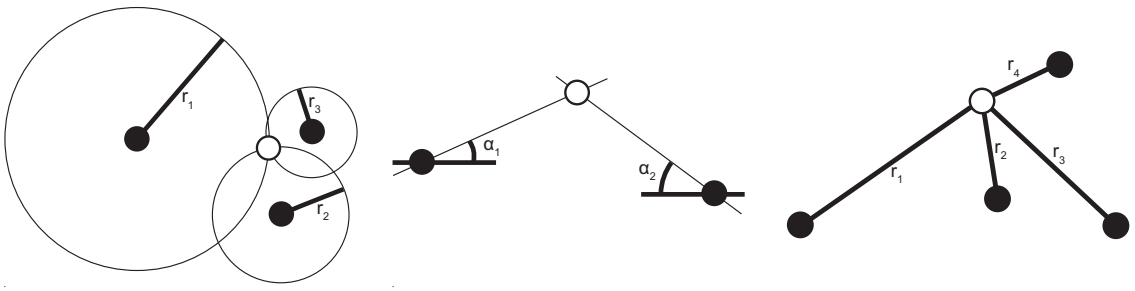

Figure 2-5
Three different network based navigation techniques (left hyperbolic triangulation, middle triangulation, right multilateration) to compute the position of an free node (white node) and their respective inputs r_i and α_i as well as the reference nodes v_i (black nodes) are shown here.

Common to all these techniques is, that they rely on distinct range or angular estimates at different distributed locations. Their approaches are compatible and a transformation of data from one technique to another can be accomplished straightforward.

2.1.5 Generalized Navigation Solution Using Trilateration

In the case of three of more independent range estimates to known reference nodes a three dimensional trilateration problem is to be solved. If these references reside at a known location (anchors or settled nodes), the absolute position can be given in reference to their inertial system, such as is done for the GPS system [Log92].

For three references a geometric approximation can be given as has been described by Caffery [Caf00]. In wireless ad hoc networks, trilateration problems are usually overdetermined, meaning, there are more range estimates incorporated into a solution than necessary to be able to solve for all unknown. Furthermore the errors in the range estimates make it difficult to solve the resulting set of linear equations suggesting techniques that use all available inputs to compute an approximation of position. The starting point here are least square methods like the minimum mean square estimate (MMSE), to find an approximate solution $\hat{\mathbf{x}}$ that best satisfies $\mathbf{Ax} = \mathbf{b}$ with agreement to the range estimates given by $\mathbf{b}$. This means that the length of the residual error $\mathbf{e} = \mathbf{b} - \mathbf{A}\hat{\mathbf{x}}$ is to be minimized.

In general, the trilateration problem can be formulated as follows: Given a set of n range estimates $\hat{r}_i$ from the i-th reference at position $v_i = (x_i, y_i, z_i)$ to the unknown position v_u, the n nonlinear navigation equations for the true ranges r_i are defined by

$$r_i = dist(v_i, v_u) = \sqrt{(x_i - x_u)^2 + (y_i - y_u)^2 + (z_i - z_u)^2} \tag{2.5}$$

with $r_i = \hat{r}_i + e_i$ and the estimation error e_i. This can be linearized by subtracting the last row resulting in a system of

$$\mathbf{Ax} = \mathbf{b} \tag{2.6}$$

with

$$\mathbf{A} = -2 \begin{bmatrix} (x_1 - x_n) & (y_1 - y_n) & (z_1 - z_n) \\ (x_2 - x_n) & (y_2 - y_n) & (z_2 - z_n) \\ & \vdots & \\ (x_{n-1} - x_n) & (y_{n-1} - y_n) & (z_{n-1} - z_n) \end{bmatrix} \qquad \mathbf{x} = \begin{bmatrix} x_u \\ y_u \\ z_u \end{bmatrix}$$

$$\mathbf{b} = \begin{bmatrix} r_1^2 - r_n^2 - x_1^2 + x_n^2 - y_1^2 + y_n^2 - z_1^2 + z_n^2 \\ r_2^2 - r_n^2 - x_2^2 + x_n^2 - y_2^2 + y_n^2 - z_2^2 + z_n^2 \\ \vdots \\ r_{n-1}^2 - r_n^2 - x_{n-1}^2 + x_n^2 - y_{n-1}^2 + y_n^2 - z_{n-1}^2 + z_n^2 \end{bmatrix}$$

There are different ways to solve the fundamental linear problem in calculus, geometry and linear algebra. The classical way to proceed is to solve the normal equation

$$\mathbf{A^T A}\hat{\mathbf{x}} = \mathbf{A^T b} \tag{2.7}$$

with the methods of linear algebra. However in order to account for system dynamics it is common to weight input data according to their reliability using a covariance matrix $\mathbf{\Sigma}$ and to use recursive and sequential filtering techniques to reduce the computational complexity on the event of system changes and the availability of new information. The weighting matrix $\mathbf{C}$ is a diagonal matrix that can be derived from the covariance matrix and extends equation 2.7 to the weighted least squares form of the normal equation

$$\mathbf{A^T C A \hat{x}} = \mathbf{A^T C b} \tag{2.8}$$

that can be solved in the form

$$\mathbf{\hat{x}} = (\mathbf{A^T C A})^{-1} \mathbf{A^T C b} \tag{2.9}$$

using QR decomposition or choleski factorization [SB97]. Other methods to solve such MMSE problems are estimation using taylor series or the householder transform. Although computationally intensive, this method scales well to scenarios that are highly overdetermined, have varying node degree and is fairly tolerant to range errors if network connectivity (node degree) is high.

2.1.6 Quality Metrics of Positioning Systems

There are different metrics that can be used to describe quality aspects of positioning systems. Firstly important are parameters that describe the network setting and the environment. Secondly important are such, that are relating directly to location and the algorithms used.

The environment is mainly characterized by it's signal propagation properties, the node densities, distribution, degree of connectivity and mobility. Of course the availability of a map is also a property of the environment that can enables absolute positioning while relative positioning focuses on topology discovery only. In certain environments the accessibility of information might be restricted to certain users or situations.

Navigation solutions can be classified based on accuracy, availability and cost (hardware, computation, storage and communication requirements, latency and consumed energy). The size of the database that is kept up to date in every node directly, update rates, and mathematical accuracy required influence the computational complexity and therefore position accuracy. In situation where one has to start from scratch the initial positioning error and the time to first fix, e.g. the duration for a first estimate to be available to an application are the most crucial.

2.2 Navigation Issues in Wireless Sensor Networks

In the following we want to point to some sensor network specific issues that can be utilized to improve the performance of navigation techniques. These are the network topology, range errors, quantization and different filtering techniques.

2.2.1 Localization Challenges in Sensor Networks

When applied to an ad-hoc sensor network, these radionavigation approaches face several new complications: sparse reference points that are not directly visible by all nodes in the network, limited accuracy in the range measurements, and the need for low-power implementation on limited resources. Anchor nodes, or nodes with a priori knowledge of their locations relative to a global coordinate system, are assumed to be sparse and randomly located. Like the other sensor nodes, their communication range is limited to their immediate neighborhood. This makes it difficult, if not impossible, for requesting nodes, or nodes attempting to estimate their positions, to acquire enough reference points to perform traditional triangulation. It is only assumed that there will be at least four anchor nodes in a connected network.

The accuracy derived through triangulation depends heavily on the geometry of the position references, the configuration of network nodes, and the accuracy of the range measurements. The short transmit ranges of only a few meters result in unacceptably high synchronization demands of 3 psec/cm of resolution, when TDoA techniques are employed. AoA approaches require costly antenna arrays on each node. These observations make these solutions unattractive, leaving the RSSI as the prime candidate for range measurements. Given a known transmission power and a good model of the wireless channel, the distance between transmitter and receiver can be estimated based on the received power. Unfortunately, the accuracy of these RSSI range measurements is highly sensitive to multipath, fading, non-line of sight (NLOS) scenarios, and other sources of interference, which may result in large errors. These errors can propagate through all subsequent triangulation computations, leading to useless information.

Fortunately, sensor networks possess two properties that may help to overcome these concerns: (i) dense interconnectivity leading to redundancy in the range measurements; (ii) limited mobility which allows for long observation times and the removal of some of the fast fading effects through integration. In the following section, we first discuss how these properties can be used to solve a local positioning problem (i.e. positioning between nodes that are within communication range). The following section will extend these techniques to a system where not all nodes are within range. Subsequently we are introducing a robust set of algorithms suitable for wireless sensor networks where not all nodes are within range of a sufficient amount of anchor nodes at all times.

2.2.2 Adaptive Network Topologies

Sensor networks offer regions with high node densities and ad hoc networking mechanisms that fundamentally allow every node to communicate with every other node. The key idea is now to make use of the redundant network connections available when viewing the network as a fully connected graph.

Formally a complete graph with n vertices has $\frac{1}{2}n(n-1)$ edges [Big89]. In an ad hoc network setting however not all nodes are visible to all others due to the reduced transmission range of a single node resulting in a structure such as seen in Figure 2-1 and 2-6. In order

to facilitate efficient routing algorithms and to conserve energy it is often practical to reduce the complexity of the available connections to an appropriate set of connections such as a planar graph or a dominating set [KWZ03, Wu02]. Geometric routing algorithms commonly use planar graphs such as a gabriel graph as shown on the left in Figure 2-6. In heterogeneous network settings the available network links might be reduced even further to so-called backbone links allowing fast long-haul data transfer on commonly used routes reducing the amount of edges that can be used for iterative positioning algorithms even further.

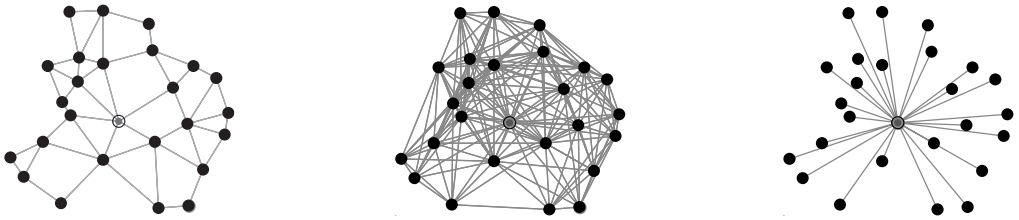

Figure 2-6
Using the wireless sensor network specific network topology for positioning allows to heavily overlay range estimates for individual trilaterations. A planar gabriel graph that is appropriate for geometric routing is shown on the left, whereas the middle shows the complete graph for a given transmission range. The right depicts all ranges visible to the center node that in this case can be incorporated into a combined trilateration solution.

A trilateration solution such as discussed earlier (see section 2.1.4) achieves the highest position accuracy when many independent range estimates to reference nodes are used. An example scenario is shown in Figure 2-6. It is easy to note that the desired topology here is quite different from the routing scenario in Figure 2-6. A high node density in a WSN setting eases positioning because the peculiarities of the overdetermined topologies can be put to use and redundant trilateration solution can be overlayed. This is exactly the reverse setting from a traditional cellular network, where mobile units only talk to a single base station. When a more mobile units crowd in a certain area, the burden on the base station increases and the additional communication links do not help in the positioning problem, since the mobile units do not communicate with each other directly.

A qualitative simulation such as the one shown in Figure 2-7 yields quite acceptable positioning errors and variances even in the case of very high range errors. The specific improvement that can be achieved for overdetermined topologies can be seen in Figure 2-12 with a characteristic nearly exponential reduction in the position error. Apart from ourselves [SRB01], independent publications have agreed on a rule of thumb of using at least five to seven nodes per trilateration to achieve an acceptable accuracy for WSN applications [SPS02, DWBP01]. From this perspective it is desirable for a positioning service in wireless sensor networks to be able to use many more independent range estimations, than offered by the routing grid alone. This could be achieved either by enforcing or switching over connections or by transceivers that can assess multiple channels (range estimates) without actually transmitting data over these channels [STGS02, SBS02, RAdSJ+00].

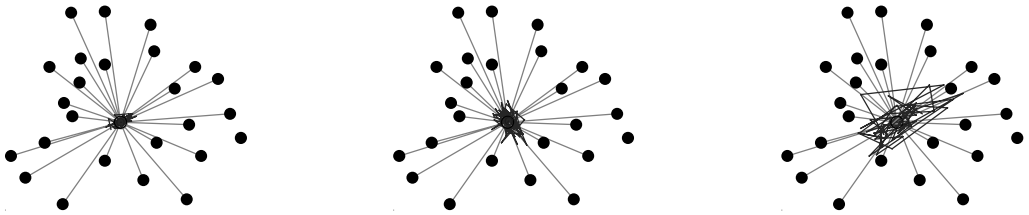

Figure 2-7
Iterative trilateration using strongly overdetermined topologies with 25 reference positions for one free node that is located in the center. Here the MMSE results are shown for successive iterations on 50 sets of uncorrelated range estimates. The three figures differentiate only in the error (left 20%, middle 50% and right 80%) that has been assumed for the range estimates.

2.2.3 Range Estimation Error and Quantization

Range estimation based on RF propagation techniques is problematic. Especially indoors the environment is not very predictable with multipath, fading, interference and shading effects abundant. Especially in the case of wireless sensor networks where lowest power consumption and therefore also low transmit power is of primary concern, obstacles in the line of sight path, noise and interference by other transmitting sources hinder the reliable estimation using a sophisticated channel model. Typical mobile radio channel models account for signal fading in the order of $\frac{1}{r^2}$ to $\frac{1}{r^4}$ but often in wireless sensor networks the attenuation is even higher. The uncoordinated placement of nodes in environments that are often harmful to radio signals adds further complexity to the estimation of ranges using radio propagation of communication signals.

Sensor nodes are anticipated to be small, simple and robust devices, down to the scale of Smart Dust [DWBP01, KKP99] carrying lightweight resources only that make it impossible to use complex transceiver structures with advanced channel estimation capability. In today's wireless transceivers, the RSSI is primarily used for the detection of a carrier signal, i.e. the signal-to-noise ratio (SNR). Internally, the baseband receiver section can then decide if it can decode a received signal or not and adapt it's parameters accordingly. In transceivers supporting transmission power control the RSSI values at the remote side can be used to reduce the local transmit power. This, however requires higher layer protocols to support this negotiation and feedback process. The link quality indicator (LQI), sometimes based on the bit error rate (BER) or a combination of SNR and BER work in a similar fashion. Without very detailed knowledge about the internal architecture and function of a wireless transceiver, it is hard if not impossible to reason about the quality and behavior of the RSSI and link quality indicator. Depending on the transmission system used and the architecture of a transceiver, a signal-to-noise ratio ratio can be measured in analog form in the vicinity of a receive amplifier or input filters (near the antenna), in the mixer stages, at the analog-to-digital converter (ADC) or in the baseband section where signal decoding and shaping takes place in digital form. Different architectures have different ways of communicating the internal values to a host controller. Systems like Bluetooth use a command/event interface supplying digitized values to the "user", others like a low-power radio (LPR) have a dedicated analog output pin, that can be either used to control

a transmit power amplifier or be converted to digital values and supplied to higher layer protocols. For instance, the Bluetooth specification only requires a Bluetooth device to tell whether the RSSI is inside, above or below the golden device power range allowing applications to discriminate analog-to-digital converter basic tradeoff if a signal and thus a connection is good or about to cut off.

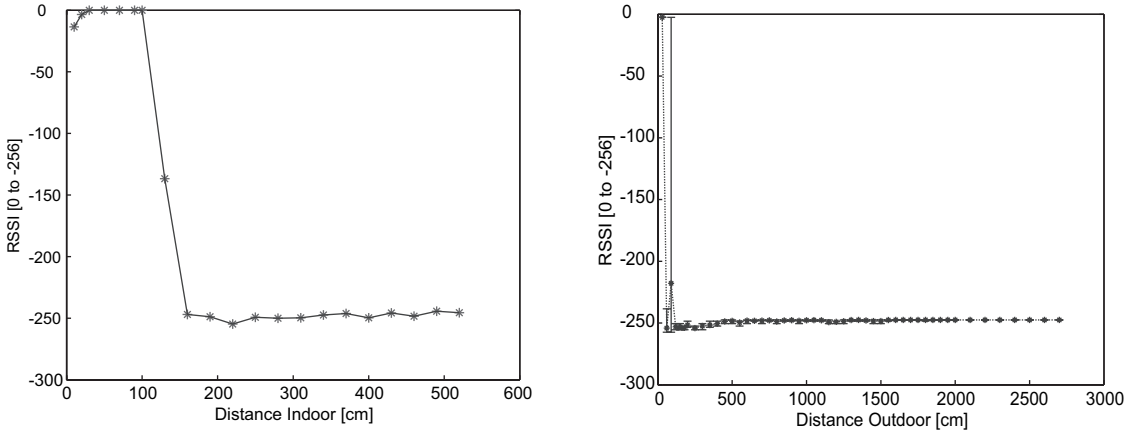

Figure 2-8
A pair of Ericsson ROK101008 Bluetooth 1.0 modules was used to measure the received signal stength indicator over distance. Two different locations, a shielded setting in an indoor absorber room (left, mean values) and an outdoor setting on a rooftop (right, mean values with variance). The RSSI resolution supported by these first generation Bluetooth devices is next to useless to discriminate more than just near and far.

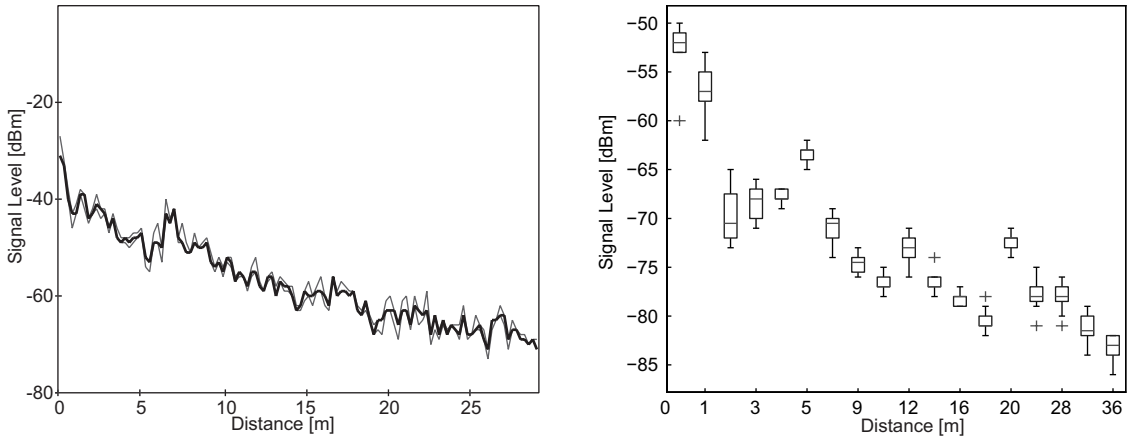

Figure 2-9
802.11b Wireless LAN and Bluetooth 1.2 received signal stength indicator measurements – A Wireless LAN card (left, minimum, mean and maximum values) and a modern Bluetooth 1.2 transceiver (right, mean values with variance) show significant improvements in their RSSI resolution. However the jitter is large and outliers quite frequent. Both measurements were performed in an indoor setting.

In order to illustrate this, some qualitative examples measured in different environments and on different types and generations of transceivers are shown in Figures 2-8 and 2-9.

Other systems such as the Chipcon CC2420 low-power radio specify somewhat linear characteristics of RSSI to input power in their datasheets, but fail to specify anything about the attenuation over distance,

It is therefore important to make the best out of the situation and pursue a strategy employing other properties than an exact channel estimation in wireless sensor networks. In the previous section we have suggested to use as many vectors as possible for the trilateration solutions. What can be done in the case of large unpredictable errors and high variance of range estimates as we have put forward? What if a sensor node can only discriminate very few discrete steps in a range estimate, what if it can only detect near and far? Quantization effects in range estimates influence positioning results, shown here for strongly limited resolution. In Figures 2-11 and 2-10 the influence of reduced quantization in overdetermined trilateration using MMSE is depicted. We show, that for large range errors the resulting position error actually increases with the quantization steps. This means, that in the case of unreliable channel estimates, it is best to not spend too much effort on range estimation but to just use the topological information for positioning.

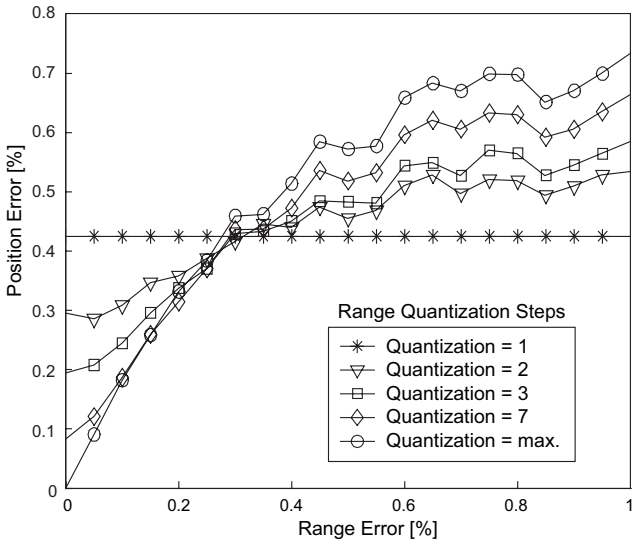

Figure 2-10
Range estimation error and quantization – Here four anchor nodes were used to compute an unknown position using trilateration. Different levels of range quantization were assumed along with range estimation errors ranging from 0-100%. With growing uncertainty of the range estimate, the resolution of the range estimate becomes less important. The case max. quantization refers to the full range estimate resolution.

To apply quantization to the measured distances, the range of possible values for distances is divided into a number of equally sized quantization intervals. After applying the quantization, all distance values inside the same quantization interval will receive the same quantized distance value. To choose this quantized distance value inside the quantization interval, a linear quantization was used for the simulation. The linear quantization is a simple quantization scheme to use. It does not take into consideration the 2-dimensional simulation setup. It instead assumes that all measured distances are distributed equally inside the quantization interval. The quantization point inside this interval is chosen in a way, that the distance from the quantization point to the upper border of the quantization interval stands in a defined ratio to the distance from the lower border of the quantization interval to the quantization point.

At low error rates, the calculated position error increases with a decreasing number of

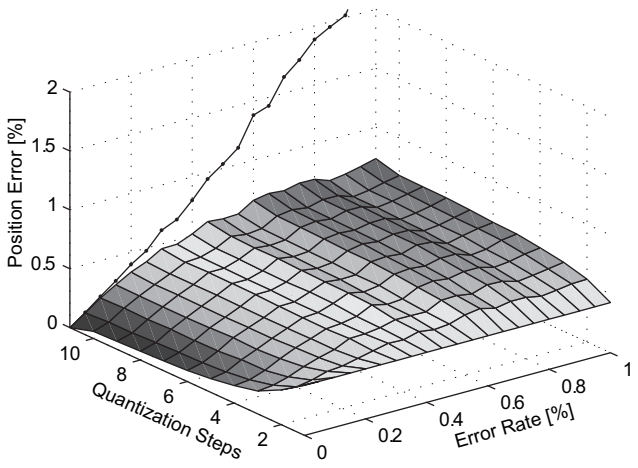

Figure 2-11
The effect of the quantization on the calculated position error: The shown graph was simulated with four anchors nodes. A position error value of one on the z-axis therefore denotes a case, where the distance from the calculated position to the real position of the simulated node equals the communication range of this node. The curve plotted on the xz-pane at the back shows the error rates obtained from calculations without any quantization or limitation.

quantization steps. However, this behavior is changing, such that for higher error rates, the calculated position error increases with an increasing number of quantization steps. Two more interesting observations are firstly that with one quantization step, the calculated position error does not depend on the error rate and secondly that the position error does not go to zero when a finite number of quantization steps are used, even when the error rate is zero.

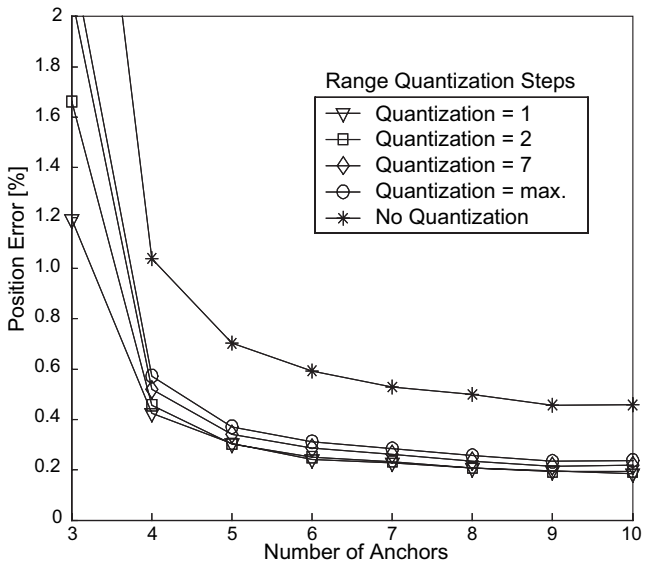

Figure 2-12
Overdetermined topologies and quantization: Using overdetermined topologies such as shown in Figure 2-7 by using more anchor nodes than actually necessary shows improved positioning accuracy using trilateration. The independent range estimation error assumed here was 50%.

2.2.4 Geometry, Border Effects and Filtering

A well known influence of accuracy in the global positioning system is the dillution of precision (DOP) [Log92, SB97]. The dillution of precision describes the geometric quality of a GPS satellite configuration in the sky. Factors that affect the DOP are, besides the satellite orbits, the presence of obstructions which make it impossible to use satellites in

certain sectors of the local sky. Especially in urban measurements, this may be limiting. We speak of HDOP, VDOP, PDOP and TDOP respectively, for horizontal, vertical, position and time dillution of precision. These quantities follow mathematically from the positions of the usable satellites on the local sky.

Translated to navigation in sensor networks, straightforward geometric analysis using erroneous inputs reveals, that certain geometric constellations of the anchor nodes result in larger position errors than others. This is easy to see when the optimum constellation is considered. Assuming similar but independent perturbation on all range estimates the best DOP is achieved, when all anchor nodes are located on a circumventing sphere in equidistant spacing from the unknown location (an equilateral triangle for two dimensions and a tetrahedron for three dimensions with the free node in the middle), and the worst DOP in the case of all anchor nodes located on a straight line. Some examples for different geometries simulated in two dimensions are shown in Figure 2-13 and 2-14.

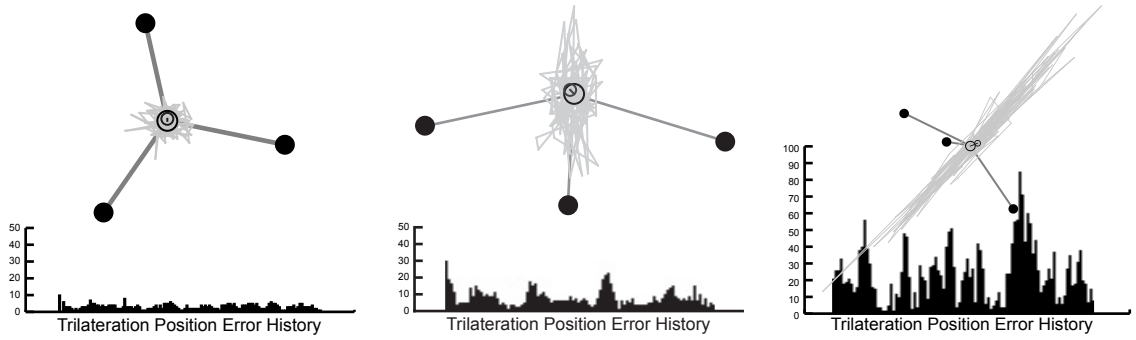

Figure 2-13
The geometry strongly influences positioning accuracy. Here, three settings with identical parameters (three anchor nodes, 30% uncorrelated range estimation error, 100 iterations) are shown for optimal (left), suboptimal (middle) and bad (right, not drawn to scale) geometries. It can be seen that for (sub-)optimal geometries, the resulting position error stays well below the range error whereas for a bad (inline) geometry the resulting position error is at least as big as the range estimation error.

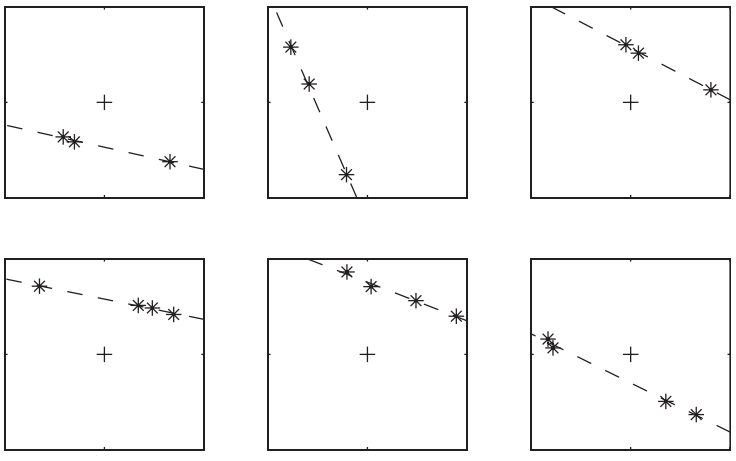

Figure 2-14
Extracted from 10000 simulation runs, these example geometries yield exceptionally high (>1000%) position estimation errors. Characteristic for this phenomenon are few range estimates (three and four anchors nodes shown here) and bad dillution of precision with all anchor nodes practically aligned.

Combining the knowledge of DOP and quantized range estimates, we have repeated the calculation shown in Figure 2-11 for different areas assuming obstacles and network deployment area boundaries. The examples shown in Figure 2-15 suggest, that border and corner areas exhibit significantly higher position estimation errors (up to 2x) than areas where anchors are located on a complete circumventing plane.

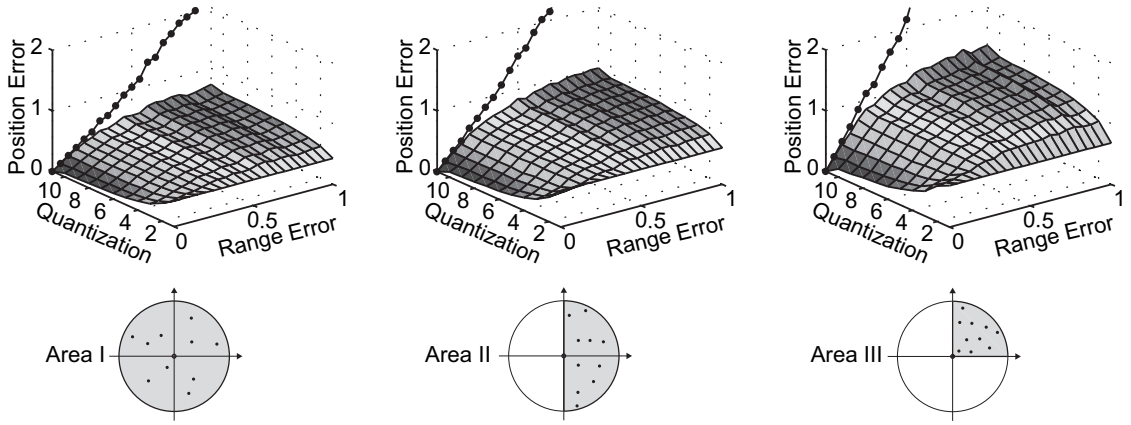

Figure 2-15
Obstacles and boundaries influence trilateration accuracy. Here three different area types are shown where the anchors are placed in a full plane (left), a half plane (middle) and a quarter plane (right) around the free node. A position estimation error up to 2x is evident for the obstructed cases with bad dillution of precision.

The implication of these observations are twofold. Firstly, edges and border regions of a network topology have to be treated differently when weighting individual trilateration solutions than regions that are surrounded by anchor nodes on all sides. Secondly, in the case of an overdetermined problem, care should be taken to select sets of equidistantly located references that are in a somewhat similar distance from the free node. If multiple such sets can be selected because of a strongly overdetermined problem, the resulting position estimates can be combined in a weighted filter subsequently. Such an approach will eliminate position estimates such as shown in Figure 2-13 that exhibit excessively large errors, mostly due to bad geometry, i.e. dillution of precision.

When using a (quasi) unit disk graph as the underlying connectivity model position estimates with large errors are not feasible in all cases, e.g. when the distance to an anchor node resulting from a position estimate is larger than the maximum radio range. Due to the fact that there are usually no exact solution possible the approximations (least squares) can produce large overshooting errors. It is thus beneficial to bound the calculated position estimates by $\mathrm{dist}(v_i, v_{i+1}) \leq r_{max}$, with r_{max} denoting the maximum transmission range. The effect of limitation on the calculated position estimate is shown in Figure 2-16. Especially in the case of a resource constrained system, checking the resulting topology for infeasible results and limiting the maximum deviation of successive iteration results is an easy to implement and efficient way of improving accuracy and robustness.

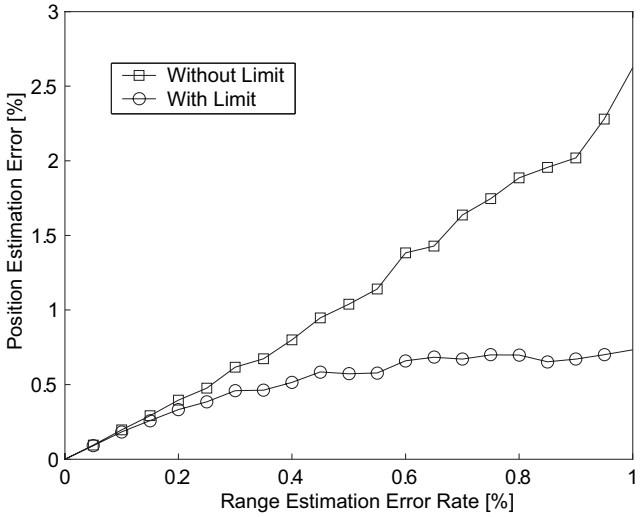

Figure 2-16
The mean of the calculated position error of the 1000 simulated trilateration iterations using 4 anchor nodes is shown. The lower curve shows these values of the calculations made with continuous measured distances with an upper limit set to the maximal communication range of the simulated node. This corresponds to an infinite number of quantization steps. The upper curve shows the values obtained from the calculations made with continuous measured distances without any limit.

2.2.5 Heuristics and Iterations

Heuristics and iterations improve the resulting accuracy, but since they depend on independent sets of inputs they add to the complexity, storage, communication and computation requirements. Sliding window and moving average finite impulse response (FIR) filters can be used to average over multiple independent sets of range estimates as well as computed position estimates. Compared to storing an extensive series of inputs and computing position estimates using time series of individual measurements, filtering over a weighted sum of individual position estimates and simple feasibility checks such as a limiter as discussed earlier is preferred for achieving a lightweight and resource sensitive solution suitable for wireless sensor networks. Of course any method using time series of data will slow down the responsiveness of a service to changes in the environment and must be carefully chosen for scenarios with high mobility.

2.3 Robust Location Management Schemes for Wireless Sensor Networks

In section 2.1.4 we have introduced the three key components of network based navigation, identification and data exchange, measurement and data acquisition and computation to derive location. In this section we introduce the hybrid approach of cooperative ranging integrating networking and positioning that offers both robust startup and precision on-demand position updates as a suitable means for wireless sensor networks. The startup phase addresses the sparse anchor node problem by cooperatively spreading awareness of the anchor nodes positions throughout the network, allowing all nodes to arrive at initial position estimates. These initial estimates are not expected to be very accurate, but are useful as rough approximations. The precision on-demand position update phase of the algorithm then uses the results of the startup algorithm to improve upon these initial position estimates. It is here that the range error and convergence problem is addressed.

2.3.1 Cooperative Ranging

The cooperative ranging scheme allows to combine the different tasks that are necessary for network based positioning to operate concurrently on many nodes. Once the neighborhood information is available at a specific node updates of ranging, updating and positioning (see Figure 2-17) that depend on each other to some extent, can be performed sequentially and out of order. Every node in the network is required to keep a database of this neighborhood information containing neighbors position estimates and the range to these neighbors. The size of this database depends on the requirements of the positioning service requested as well as on the state of the network, i.e. amount and geometry of neighboring nodes.

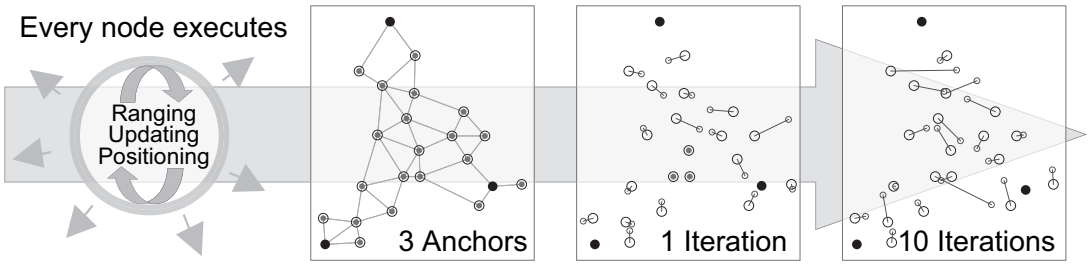

Figure 2-17
With the three phases of cooperative ranging data exchange, data acquisition and positioning operating at every node in a wireless sensor network all nodes are able to efficiently compute position estimates even when out of range of the three anchors shown above. After one iteration the nodes in the center have not been able to acquire enough data to compute a position which changes, when first order neighbors become settled node and act as position references as can be seen on the right [SRB01].

This scheme constitutes the basis of all positioning schemes and can be adapted to fit specific needs or service levels as we show for the topology discovery, startup and precision on-demand update phases in the following.

2.3.2 Topology Discovery

A node discovering it's topology by exchanging link pairs within it's scope receives range measurements to a large number of neighboring nodes. This information once again can be used to construct the topology graph of the network and to derive relative location

The Assumption Based Coordinates (ABC) algorithm [SRB01] determines the locations of unknown nodes one at a time in the order that they establish communication, making assumptions where necessary, and compensating for errors through corrections and redundant calculations as more information becomes available. These assumptions are needed at first in order to deal with the under determined set of equations presented by the first few nodes. This description of the general algorithm assumes the perspective of node v_0 and can be solved successively reducing the amount of necessary computations.

The algorithm begins with the assumption that v_0 is located at the origin $(0, 0, 0)$. The first node to establish communication with v_0, v_1, is assumed to be located at $(r_{01}, 0, 0)$,

where $r_{01} = \mathrm{dist}(v_0, v_1)$. The location of the next node, v_2, can then be explicitly solved for, given two assumptions: the square root involved in finding y_2 is assumed to yield a positive result, and z_2 is assumed to be 0.

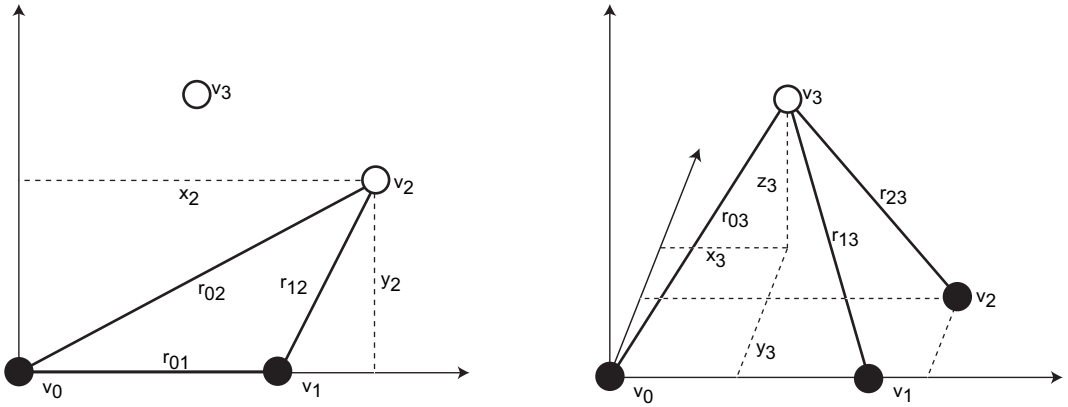

Figure 2-18
Fixing a local set of coordinates starts topology discovery as is shown for the ABC algorithm here. Adding more and more nodes to such a scheme a three dimensional reference system can be built up successively that is relative to the origin node v_0.

$$x_2 = \frac{r_{01}^2 + r_{02}^2 - r_{12}^2}{2r_{01}} \qquad y_2 = \sqrt{r_{02}^2 - x_2^2} \qquad z_2 = 0 \tag{2.10}$$

The next node, v_3, is handled much like v_2, except that only one assumption is made: the square root involved in finding v_3 is positive.

$$x_3 = \frac{r_{01}^2 + r_{03}^2 - r_{13}^2}{2r_{01}} \qquad y_3 = \frac{r_{03}^2 - r_{23}^2 + x_2^2 + y_2^2 - 2x_2x_3}{2y_2}$$
$$z_3 = \sqrt{r_{03}^2 - x_3^2 - y_3^2} \tag{2.11}$$

From this point forth, the system of equations used to solve for further nodes is no longer underdetermined, and so a standard MMSE algorithm can be employed for each new node. Under ideal conditions, this algorithm thus far will produce a topologically correct map with an orientation relative to the local node v_0. A similar approach to derive a startup configuration with a local coordinate system established at every node is followed in [CHH02]. Here a geometric transformation that is used to transform multiple local coordinate sets (LCS) into one oriented and networked coordinate system is described as well.

2.3.3 Robust Start-up Positioning Scheme

The goal here is to have a service that is available at all times on every node, no matter how small the node and independent of dedicated resources with in a network node. Tight

integration with the network transport and efficient operation are the main aspects here. For many applications such a simple, lightweight service that might not be able to give high accuracy, account for dynamics and only support relative positioning is sufficient.

The purpose of the startup phase it to solve the sparse anchor problem, which comes from the need for at least four reference points with known locations in a three-dimensional space in order to uniquely determine the location of an unknown object. Too few reference points result in ambiguities that lead to underdetermined systems of equations. For initial startup in a multi hop ad hoc network a mandatory requirement is that a network be connected and for generating a first unambiguous position estimate also a sufficient node degree. In the simple case of a single node this translates to a minimum degree of three for the unknown node (see Figure 2-13). When assimilating a WSN topology to be a wire frame with nodes acting as hinges [NN01], this can be extended to the whole connected network. But successful startup does not yet imply, that accurate positions can be computed. As we have shown earlier, different factors influence accuracy and even the convergence of the positioning problem. Especially bad DOP and large range estimate errors in multi hop scenarios can add up and cause divergence of the positioning results.

In order to support only the basics necessary for a successful startup different suggestions have been made by [SRB01, SRL02] (Hop-TERRAIN), [NN01] (DV-hop) and [CHH02] (LVS, LRG). The basic idea here is to use a hop count to the nearest references that can be derived from the local topology cache of every node to estimate an extended range from the unknown to the reference point. Usually, most nodes will start without known locations and only a few randomly distributed anchors will exist. It is therefore highly unlikely that any randomly selected node in the network will be in direct range with a sufficient number of anchor nodes to derive its own position. Hop-TERRAIN solves this problem by trading off accuracy for consistency. The startup phase will provide rough guesses of the nodes initial positions. It is shown in [SRL02] that this is good enough as an input to the second phase for refining the position estimates.

The Hop-TERRAIN algorithm works as follows: At large time intervals, each of the anchor nodes launches the Hop-TERRAIN algorithm by initiating a broadcast containing its known location and a hop count of zero. All of the one-hop neighbors surrounding the anchor will record the anchors position and a hop count of one. Then they perform another broadcast containing the anchors position and a hop count of one. This process continues until each anchors position and an associated hop count value have been spread to every node in the network, (see Figure 2-19). It is important that nodes receiving these broadcast packets only store and rebroadcast a certain anchors position if they have not received such a packet with the same or smaller hop count before.

Once a node has received data regarding at least four anchor nodes, it is able to perform a trilateration to estimate its location, (see Figure 2-5). This will of course only be a very rough estimation of the actual positions.

The DV-hop algorithm proposed by [NN01] works very similar and is used to approximate location for all nodes in an isotropic environment. It allows to orient a network on a plane with limited mobility nodes and works with few references. The claim here is that DV-

Algorithm 1 Hop-TERRAIN

```
for all v_u, v_s ∈ G do
  while receiving position packet do
    if v_a ∉ Γ_ρ(v, G) or lower hop-count received then
      store hop count
      broadcast position packet with (hopcount + 1)
    end if
    if |v_a ∈ Γ_ρ(v, G)| ≥ (dimension + 1) then
      estimate current position using MMSE
    end if
  end while
end for
```

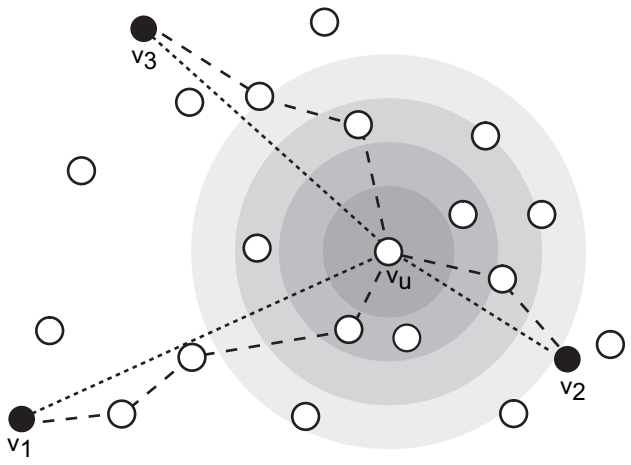

Figure 2-19
The Hop-TERRAIN startup phase uses the hop-count over all intermediate nodes to the closest anchor nodes v_i to estimate extended ranges to be used in the computation of initial position estimates for all unknown nodes v_u. The maximum radio range of node v_u is given by the shaded circle here. In this example, the resulting range estimates $\hat{r}_1$, $\hat{r}_2$ and $\hat{r}_3$ are four, two and three respectively.

hop allows distributed efficient position awareness for non-GPS enabled nodes providing an average accuracy of less than one radio hop from the true location. The difference here is, that a hop count also takes place between reference locations (anchors) and in combination with the true distance a priory known between these references an individual weighting metric for each hop is computed rather than an average hop length as is used in Hop-TERRAIN.

2.3.4 Precision On-Demand Position Updates

With the initial position estimates of Hop-TERRAIN in the startup phase, the objective of the refinement phase is to obtain more accurate positions, using the estimated ranges between nodes (see Figure 2-7). This paradigm can be abstracted as a scalable on-demand service dependent on the available infrastructure for range estimation, the environment and the available computing resources.

In the startup phase, the Hop-TERRAIN algorithm floods the anchors positions through

the network and nodes record the hop count of the shortest path to each anchor. Hop-TERRAIN also records the neighbor IDs on the shortest path. These IDs are collected in a set of potentially sound neighbors. When the size of this set reaches four in three dimensions (three in two dimensions) a node declares itself settled and may enter the refinement phase. The neighbors of the settled node add its ID to their sets and may in turn become settled, etc.

Refinement is an iterative algorithm in which the nodes update their positions in a number of steps. At the beginning of each step, a node broadcasts its position estimate, receives the positions and corresponding range estimates from its neighbors and computes a least squares triangulation solution to determine its new position. Often the constraints imposed by the measured distances will force the new positions towards the true location of the node. Refinement stops and reports the final result once updates become small.

Without any prevention, the large errors induced by RSSI measurements will propagate fast throughout the network. Therefore, a confidence metric can be included in the refinement algorithm. Instead of solving the unweighted least squares the weighted version as introduced in section 2.1.5 is solved. Each node assigns a confidence weight between zero and one to its position estimate. Anchors immediately start with a confidence value of one. Unknown nodes start with a low value and may raise their confidence after subsequent refinement iterations. Whenever a node performs a successful triangulation, it sets its confidence level to the average of its neighbors levels. In general, this will raise the confidence level. It is shown in [SRL02] that including confidence levels improved the refinement phase considerably.

A general scheme for efficient and robust precision on-demand location awareness that can be derived from these ideas is shown in overview in algorithm 2.

Algorithm 2 Precision On-demand Positioning

```
for all v_s ∈ G do
  while receiving position packet from Γ_ρ do
    establish connectivity graph, store and exchange neighborhood information
    count hops to next references
    estimate span between hops and weight the hop count
    estimate current position using MMSE
    bound and filter position estimates based on the neighborhood topology
    check error against reference positions
    if error too large then
      inc(ρ)
    end if
  end while
end for
```

Further improvements can be made by detecting that a single node is ill-connected: If the number of neighbors is less then four in three dimensional and less than three in two

dimensional space then the node is ill-connected. However, detecting that a group of nodes is ill-connected is more complicated, since some global perspective is necessary. A heuristic can be employed that operates in an ad-hoc fashion, yet is able to detect most ill-connected nodes. The underlying premise for the heuristic is that a sound node has independent references that is, the multi-hop routes to the anchors have no link in common.

When performing subsequent updates using cooperative ranging schemes such the refinement scheme described above different parameters can be adapted according to the requirements of the applications using the positioning information. To achieve the goal of high local connectivity for optimal trilateration results it is important to expand the size of the scope of each node also termed the location reference group (LRG) by [CHH02]. With a growing visibility of the network more nodes and also more references can be taken into account into each local wire frame model at the cost of higher storage and computational requirements.

2.4 Infrastructure

2.4.1 Netsim – A Positioning Simulation Environment

The existing approaches for network simulation are typically based on low-level, event-driven models of networking and physical layer functions [NS-2, GloMoSim, Opnet, etc.] and target the detailed statistical analysis of very specific protocol functions. They are often insufficient when analyzing large network topologies in varying environments or high-level services such as local positioning.

We have designed a simulation tool suite that allows extensible and parametrized exploration of positioning algorithms. Different algorithms as well as error models can be easily plugged into the framework (see Figure 2-20). It has been used to generate most of the results presented earlier.

- Designed to cope with very large multihop networks
- Every node is implemented as single instance
- Different communication mechanisms
- Support for different settings and mobility
- Detailed analysis and logging capability
- Easy to use and parametrize for fast design cycle

The Netsim tool suite uses a single instantiation per node to create a time-multiplexed distributed network simulation run. Each node instance is randomly selected and executed once per run. Successive runs can be scheduled with varying environmental parameters. The interaction between nodes is based on network messages for data exchange and different error models for the range estimation between nodes.

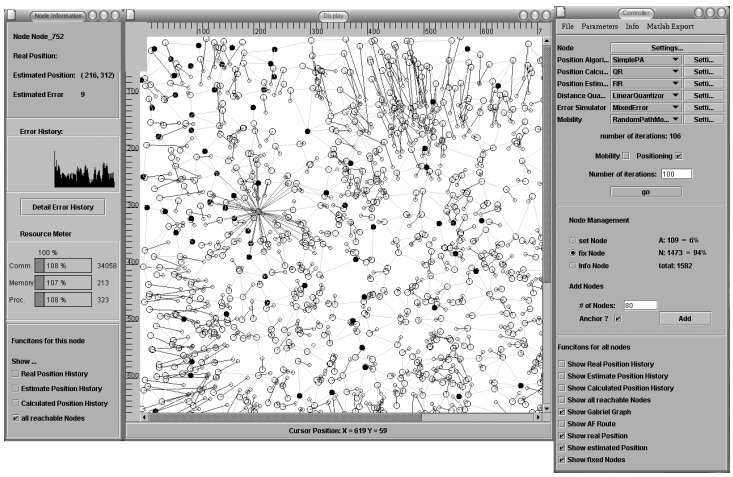

Figure 2-20
The Netsim positioning simulation framework consists of a graphical front end and a simulation back end. It supports easy to use parametrization of the simulation environment and extensive support for analysis. Shown on the left is a setup with 1582 nodes after 100 runs. Regions with high position estimation errors can be seen in the upper right and lower left corner were anchors are sparse.

2.4.2 Services for Location Management

For a location management service we can identify the following functions that need to be supported on every networked node that can be split up into communication based functions and local execution primitives at each node:

Search and Identification of Nodes

Not all radio communication systems allow to explicitly search and identify individual nodes. However, for a positioning algorithm this is a vital function. Furthermore it is beneficial if a service can use primitives that allow to identify a node type, role and/or state, i.e. to identify anchor, free or settled nodes prior to a connection setup. Service discovery, which allows to detect the capability of a certain service availability from remote is not a strict requirement for a location service but increases performance due to a reduction of the search time for a certain node/service.

Range Estimation

Although strongly connected with the capability to detect a carrier signal and demodulate symbols in a wireless communication link, not all transceiver systems allow to generate range estimates between transceiver pairs. Those that do support such features, such as RSSI or LQI typically use the data provided as a basic means to select among different links (if available). Other such as the GPS system use the radio signal to solely estimate ranges (all data transferred from satellites to receivers is used to compute range and position estimates) and do not allow private data communication in the sense of an application. A twin system of both capabilities would be most desirable.

Data Exchange

In order for location estimation schemes as proposed to work, nodes have to exchange data with each other (on a local scale). This necessitates the control of communication links (topology control, link supervision, routing), reliable data transfer and time synchronization. In dynamic environments with either varying conditions or mobile nodes, data sets exchanged cannot be brought into a common context without time synchronization, i.e. nodes must be able to transfer data sets within a given bounded interval and align them with local observations,

Data Storage

Actually a basic primitive of every computing system, positioning schemes have extra requirements of time-synchronized data storage. This should be supported both for range estimation data as well as for arbitrary events to support logging and debugging.

Computation

Local computation with or without support through dedicated hardware is required to compute filtering and MMSE results. Usually not an option in wireless sensor networks distributed execution across multiple nodes can help to offload computationally intensive tasks to nodes with more resources in heterogeneous environments.

Local Node Management Functions

Location estimation is a more complex task than the typical duty-cycled, sense–and–broadcast application found in many wireless sensor networks. Basic node management functions that need to be supported at every node to support the more complex and event driven execution in a location management service are operating system based process (thread) management, support for hardware drivers, storage and memory functions, power management, user interface modalities (control, debugging, verbosity levels)as well as status monitoring.

3

A Distributed Environment for Prototyping Sensor Networks

Although still a fairly recent field, platforms and prototype systems for wireless sensor networks have received substantial attention and been focused in numerous investigations. The classical approach to wireless sensor network devices following the vision of Smart Dust [KKP99] aims at low-power and highly integrated node hardware. One of its most prominent representatives is the UC Berkeley family of Motes [HSW+00] that has been commercialized and is widely used by researchers and industry all over the world. Other research groups have developed similar architectures, developed from off-the-shelf components and based on a low-power microcontroller and a custom radio front end chip like the Smart-Its [BG03], Particles [DKBZ05], Eyes [HPH+05], ScatterWeb [RTVS03] and Telos [PSC05] platforms. Common to most of these architectures is that all communication (baseband, medium access control and link-layer protocols) as well as the application processing is performed on the host microcontroller, which implies a meticulous design process, extensive knowledge about real-time systems, and in effect the allocation of substantial resources on the host central processing unit (CPU). Another drawback of these platforms is, that they can only communicate with devices equipped with the same type of radio and the necessary protocol processing capabilities. In contrast to this, the BTnodes can interact with any Bluetooth-enabled device without the need to integrate further hard- or software.

3.1 Related Sensor Network Platforms

3.1.1 Early Platforms

An early platform in mobile computing research was the ParkTab device developed by Want at Xerox PARC [WSA+96]. The ParkTab is a small battery powered mobile communication device that allowed to receive messages and interact with services running on

back end servers. Furthermore a simple cell-based localization service was available based on infrared, ceiling-mounted location beacons.

The InfoPad [TPB98] developed by Brodersen et al. at the University of California at Berkeley (UCB) is a portable multimedia terminal that allowed the streaming of real-time audio and video data over a wireless data link to a farm of back end servers responsible for all processing tasks. All very-large scale integration (VLSI) circuits here were custom made to optimize efficiency and battery life of the mobile units.

The back-end to the Sentient Computing system by Ward and Hopper at AT&T Research Cambridge [WJH97, ACH+01] consists of a dense ceiling array of transceiver modules connected over a system bus. Mobile, battery-powered devices, the so-called "bats", that communicate with the transceiver modules via a radio down-link and an ultrasound up-link. This technique allows for precise location determination of the active bats using trilateration based on ultrasonic time of flight and is deployed for experimentation on about 1000 m^2 of building space spanning multiple floors.

All of these early platforms have in common a fixed infrastructure that needs to be put in place, operated and maintained for the lifetime of the system. Moreover they were more or less targeted at a single application only with limited flexibility or reuse in follow up projects. Among these examples, the active bat system is the only one that has a lifetime of multiple years and is today still and being operated and used to produce current research results [HWH03].

3.1.2 State of the Art

The de facto standard in WSN platforms today, both in industry and research, are the Berkeley Motes and the TinyOS software framework by Culler et al. [HSW+00, LMP+05]. More than just a piece of operating system software and demo applications, TinyOS is a whole new component based paradigm for wireless sensor networks that assumes an ongoing evolution of hardware devices and software components based on a network-centric abstraction [CHB+01]. Introduced in 2000 it has gained significant impact over the past years, mostly in the academic and research communities. The commercialization of the standard platform for TinyOS, the Berkeley Mica Motes through the industrial partner Crossbow has not only made experimentation accessible to researchers without the systems focus or manufacturing capabilities to develop their own testbeds but also to emerging industrial applications and commercial products.

Originally following Kahn's vision of Smart Dust [KKP99] the family of Motes were developed from comodity-off-the-shelf (COTS) components [HC02] and are targeted for a broad audience. The family of Motes, based on the Atmel AVR micro-architecture and an additional radio chip ranges from early prototypes like the WeC, Rene, Rene2, Dot and Mica Motes to the widespread Mica2 and Mica2Dot and design studies such as the Spec integration of a Mote-on-a-chip (full system-on-chip) by Hill [Hil04]. Recently added members of the family pave the way to new radio architectures such as IEEE 802.15.4 (Telos, MicaZ, Imote2), Bluetooth (Imote) or lower power micro-architectures (Telos and

Eyes using a TI MSP430, Imote using an ARM7 thumb, Imote2 using an Intel PXA271). Especially the Intel Imote architecture deserves some attention here as it is closely related to the BTnode architecture. The Imote is based on an ARM7 thumb and a Bluetooth radio integrated into a Zeevo Bluetooth system-on-chip (SoC) [KAH+04]. This requires to run both the proprietary Zeevo OS and Bluetooth protocol stack as well as the TinyOS application on the same CPU resulting in considerable burden and software complexity. As a result the Imote2 [NKA+05] is again separated into two subsystems, a CPU and a radio.

Usually deployed in outdoor scenarios applications such as monitoring of seismic activity [WAJR+05], habitat monitoring [SOP+04, SPMC04], wildlife monitoring [CEH+01, MCP+02] and military applications like shooter localization [MSLS04] or wide-area surveillance [HKS+04] have used setups ranging between tens and hundreds of nodes.

A recent study and classification resulted in a proposal to extend WSN architectures to tiered architectures [HHKK04] with connection to back-end infrastructure using Stargate embedded Linux devices, medium sized nodes capable of bridging networks and supporting high data transfer rates (Bluetooth devices like Imotes and BTnodes) as well minimum complexity and thus ultra low-power sensor devices, such as the traditional Mica2 Motes.

3.1.3 Research Platforms

There are many platforms that have been developed for research purposes or as case studies only. The PicoRadio testbed by Rabaey et al. [RAdSJ+00] aims at the early investigation of system level properties and software architecture trade-offs to be used in a full SoC implementation of a PicoNode. Focus is on highly-specialized ultra low-power radio architectures [RAK+02] and SoC integration. In the first testbed prototypes a stack of boards is used to accommodate a Strongarm processor with adjacent field-programmable gate array (FPGA) for baseband processing, a selection of radio modules, input-output (IO) and power components.

Similar in architecture but with less performance and targeted at networking and application layer research are the Rockwell/WINS nodes developed by Pottie [PK00]. RadioActive networks [BWG99] drawing the strength from both software radios and active networks first described by Tennenhouse et al. [TW96, TSS+97, Ten00] are other examples of network-application integration.

The Medusa MK-II nodes by Srivastava [SS02] step up the system integration effort and versatility incorporating multiple radios and multimedia IO on a single printed circuit board (PCB). These were augmented with various sensors to serve as the I-Badge platform used in ubiquitous computing research to create a smart kindergarden [CMY+02].The many subsystems on the I-Badge have however induced considerable system complexity and also interfacing constraints that are impractical in the long run.

Research focusing on ad-hoc networking protocols and local positioning by Estrin and others [EBB+03] has made use of mixed environments of iPAQ devices and Mica2 Motes

with attached ultrasonic ranging sensors. Applications here range from beamforming using comodity-off-the-shelf components [WYM+02], and habitat monitoring [WEG+03a]. While first pioneered at the University of California at Los Angeles (UCLA), researchers at MIT developed the concept of an acoustic Mote further into a now commercialized platform, the Crossbow MCS Cricket with on-board ultrasonic rangefinder [PCB00].

The Smart It's devices were conceived for ubiquitous computing experimentations and are oriented towards a low-complexity, quick to assemble solution requiring minimum effort and learning phase. They are built from 20 MHz PIC microcontrollers using RFM low-power radios by Beigl [BG03]. Recently a whole family of sensor add-ons and different sensor–processor–radio combinations have been revised under the name Smart It's Particles [DKBZ05] targeting the lower end of system functionality and size. A variant of this idea of quick assembly and kit-based experimentation has been developed by Schmidt [Sch02] to support physical prototyping of ubiquitous computing experiments. A similar theme of interaction within a pervasive computing world is followed by Fuhrmann et al. [FKO03, HBFZ04] and their Bluewand devices.

The ScatterWeb devices designed by Schiller [RTVS03] are embedded wireless modules using an HC11 microcontroller, a low-power Radio, Bluetooth, wired ethernet and a display for educational purposes. These devices can run full TCP/IP software as well as web servers and simple application software.

The uAmps project at MIT focuses on adaptive low-power sensor networks [WMM+01] with work on energy-scalable protocols by Wang [WHSC01] and Heinzelman [HSWC00] on power-aware wireless micro-sensor networks by Min [MCB+02]. Furthermore, an application-specific protocol architecture for wireless micro-sensor networks was proposed by Heinzelman [HCB02].

Researchers at the University of California at Irvine (UCI) have developed platforms for different applications, namely the Eco [PLC05] and DuraNode focusing on power utility maximization for multiple-supply systems by using a load-matching switch [PC04] and the Itsy platform [LC04] focusing on combined architecture-operation trade-offs in dynamic voltage scaling, partitioning and power-failure recovery.

The Consensus platform [RPW+04] makes use of two different radios (Bluetooth and 802.11b) with strongly differing characteristics to investigate power–performance trade-offs in embedded microserver applications. Albeit similar to many architectures found in todays personal digital assistants (PDA), this platform allows for in-situ profiling and analysis not possible on it's commercial counterparts.

The electronic shepherd is a device encompassing a GPS, GPRS communication and an UHF transceiver for tags used in animal flock monitoring by Thorstensen [TSBW04].

Bellis et al. [BDO+05] have proposed yet another stackable platform consisting of 25 mm^2 sensor cubes. Each cube supports a single function (radio, compute, FPGA, sensor, interface etc.) and is connected to other cubes using a global, vertical inter-cube bus system.

The Gnomes testbed for low power heterogeneous wireless sensor networks devised by Welsh [WFF03] is based on lower power COTS dust devices using an MSP430 microcon-

troller powered from nickel metall hydride (NiMH) cells. An operating system (OS) with additional process management and scheduling has been modeled after TinyOS.

3.1.4 Commercial Platforms

The family of Berkeley Motes is successfully marketed by Crossbow that manufactures the devices, sensor boards and developer kits as well as offering training and consulting services. Several variants differing in radio and sensor capabilities have been produced as well as auxiliary sensor board, programmer and microserver hardware (Stargate) and appropriate software tools. Within a larger DARPA project the extreme scaling mote (XSM) has been developed and is currently being deployed with about 10'000 nodes [Ros04, DGA+05]. Here, Crossbow takes the lead role in the industrialization and product management based on the original UCB open source hardware designs and software projects.

A long time contender in the sensor network arena, Sensoria spun out of UCLA and is mainly involved in full-custom military applications. Here, due to the commercial nature of the projects, functionality and robustness is of higher importance than small size or budgetary constraints. So many of the applications are actually not light weight at all, as documented by Elson [Els03].

There are quite a few young companies in the field that focus on scalable, secure, and robust mobile mesh networking for observation and security applications such as PacketHop or Millenial Net who are producing building and industrial automation applications based on sensor networks. Others started out by producing devices such as the TACT-433 by IP01, the TinyNode584 by Shockfish, the M1010 motes from DustInc or the Tmote Sky by Moteiv [PSC05] and are offering more and more system integration as well as software services.

Industrial embedded modules, like the series of radio modules offered by Panasonic allow easy integration of wireless communication without specialized skills (RF engineering). In the case of the Zeevo equipped Bluetooth modules with integrated ARM core and memory, these modules could be used as stand-alone platform, i.e. like the Imote, without further modification. However, (user-) interface, power management and sensor components are missing on such modules.

3.1.5 Specialized Low-Power Architectures

The experiences of Raghunathan [RPW+04] using a low-power wireless mobile computing platform show that use of specialized, low-power individual components alone is not enough to achieve a system goal, but that that the entire system has to be designed and operated in a low-power aware manner from the ground up, carefully orchestrating the transitions of the various components to and from their low-power states such as proposed by Lu [LCS+00] and others.

The PicoBeacon is a recent, low-power design study of a self powered radio circuit using an integrated solar cell, minimal charge buffering and power control circuits [ROC+03].

The WiseNet architecture proposed by Enz [EEHDP04] uses a specialized, custom MAC protocol using minimized preambled to achieve ultra-long lifetime requirements using single cell battery power supplies and so therefore not compatible to other devices and protocols. In contrast to WiseMAC [EHD04], MAC layer protocols such as S-MAC [YHE04], B-MAC [PHC04], T-MAC [vDL03] and BitMAC [RR05] operate on standard comodity-off-the-shelf hardware and are thus widely accessible.

3.1.6 Operating System Software for Sensor Networks – TinyOS

Most of the hardware platforms discussed in the previous sections use proprietary custom software and only provide a thin hardware abstraction layer, on top of which applications are executing [RTVS03]. One particular exception is TinyOS [HSW+00], a component-oriented operating systems architecture designed for the Berkeley family of Motes. TinyOS [CHB+01] pursues a network-centric approach to embedded software systems. Similar to our approach presented in section 3.4.3 it provides asynchronous events as a basic programming abstraction. System services (such as multihop routing) as well as applications are implemented as a set of components with well-defined interfaces. A simple declarative language is used to connect these components in order to form a complete application. TinyOS applications are programmed in this custom C dialect [GLvB+03], requiring special development tools, i.e. a nesC compiler. This is in contrast to our platform, which uses standard C and an unmodified GNU tool chain.

Today the TinyOS software repository contains numerous additions and contributed demo projects that can be adapted and facilitate the development of custom applications. In retrospective of the past years, the community effort generated by TinyOS and it's open-source policy has shown significant impact. The software implementation of the core TinyOS-1.x is well maintained and contains a time tested and versatile set of functions. Although well perceived and popular throughout the community, it must be noted that the software and its details are demanding on the developer and ask for meticulous care and profound knowledge to be useful. Aside from concepts and paradigm, the reality and day to day routine of developers consist of just as many bug-fixes and workarounds as in most other software systems and is not rocket science at all but serious engineering work. The ongoing work to create a TinyOS-2.x based on unified models and abstractions [LMG+04, HPH+05] look promising to date but will need extensive further development.

Default TinyOS-1.x applications use a similar approach and structure as the first generation BTnode System Software (see section 3.4.3). It's main application model followed by the majority of developers assumes a duty-cycled stream-oriented application that is very suitable when assuming the processing of sensory data and protocol data on the communication links. Approaches to interleave such streams at different rates exist, but things get very complex and unpredictable when control flow dominates in application or mixed datastream and control flow applications need to be debugged.

3.1.7 Support Middleware

Various projects aim to provide a service infrastructure or middleware which supports the development of complex sensor network applications. TinyDB [MFHH02] interprets the sensor network as a distributed database being constantly filled with sensory data such that a simple SQL-like query language can be used to query the sensor network. SensorWare [BHS03] uses an agent-like approach to program sensor networks for complex tasks. A scripting language is used to define small programs which are then distributed to the nodes of the network. Later on, executing scripts can migrate from node to node along with their state information. DSWare [LSS03] uses a real-time variant of an event notification system, which allows to subscribe to specific events generated by sensor nodes.

3.2 Metrics of Wireless Sensor Network Platforms

Sensor networks have been built for a number of different, widely varying applications and environments. Although unified requirements and characteristics to suit the needs of a majority of applications cannot be established there is a certain consensus about the state-of-the-art in applications and the typical characteristics of an optimal system [RM04].

The standard wireless sensor network application usually assumes spatially distributed nodes with simple, individual sensors that transmit these sensor values to a common sink using a wireless (multihop) network. In order to achieve a cost-effective and minimum-obtrusive solution nodes are generally thought to be of small physical dimensions, low-cost and operating in a power-efficient (duty-cycled) fashion from their own power supply. Typical application examples here are in environmental monitoring [MCP+02] object tracking [SBP+04] and long-term surveillance [LSZM04]. Further general requirements for WSN devices are a small form factor, ubiquitous usage, unobtrusive application, programmability, availability and affordability. Often underestimated, the appropriate development tools, sample applications, documentation and support are also playing a major role for a success of platforms and applications alike.

3.2.1 General Platform Metrics

It is difficult to compare different platforms, especially when designed for and used in different applications where unified requirements cannot be established [RM04]. Nevertheless, general requirements and metrics are important for the assessment, evaluation and comparisons of platforms, applications and implementations. Many of the criteria used to describe the performance and characteristics are conflicting, requiring careful analysis and application dependent metrics. We will be specifically interested in metrics concerning platform architecture, tools and support, and applications as discussed in the following.

Architecture

Generally, architectures of WSN devices can be partitioned into three subsystems: communication, computation and sensing/interface. Although a platform architecture is a combination of these three subsystems, it is often practical to distinguish between these

when evaluating and comparing platforms in the interest of greater transparency and detail.

Specific architecture metrics are:

- CPU speed and architecture
- Memory size and type
- IO capabilities
- User interface capabilities
- Sensors, type, resolution, sample rate, accessibility
- Programming modalities, necessary tools, number of cycles
- Radio characteristics, data rate, frequency, number of endpoints, number of channels, setup time
- Transmission characteristics, transmit power, sensitivity, interference, antenna type, range
- Radio and interface capabilities, protocols
- Power consumption in all operating states
- Energy supply, power conversion
- Cost, initial acquisition and follow up for deployment and maintenance
- Deployment effort, required know-how
- Physical properties such as casing, size, robustness
- Availability, year of introduction

Tools and Support

The availability of appropriate development tools and access to all system specific documentation is key for successful applications. Only a thorough, transparent and up-to-date documentation will allow developers to use the platforms appropriately and with the necessary care for the details, that are so important when operating on resource constrained systems. Especially when pursuing team efforts and in academic environments, the question of royalties and licenses per user and per developer kit necessary are of considerable concern. Here, royalty free tools and open-source software and documentation models are generally anticipated an so facilitates external collaboration, a model that is becoming more and more important in commercial application domains as well.

Specific tool metrics are:

- Availability of development tools
- Availability of system libraries and example applications
- Cost of tools and developer kits per user
- Availability and access to documentation
- Debugging support and tools

Applications

Many applications have been proposed (see section 1.1.2) spanning a wide variety of properties and goals. Today, the design space of sensor network does not follow a single common denominator as we have discussed in section 1.1.3. It is thus difficult to characterize applications, especially with the use of a single set of standardized metrics.

Some commonly used metrics for applications are:

- Application, system and single node lifetime
- Persistence of the application
- Peak and average load
- Memory requirements
- Data traffic and patterns
- Data processing, storage and aggregation
- Modalities and frequency of user interactions
- Startup behavior and duty-cycle
- Mobility
- Homogeneity/Heterogeneity of nodes
- Coverage area
- Availability, access to the application
- Redundancy, resilience to failures
- Dependence on infrastructure

3.2.2 State of the Art Platforms Compared

For a concrete platform comparison example, characteristic figures have been compiled for the Crossbow Mica2 and Mica2Dot, the Moteiv Tmote Sky (aka Telos) and the Intel Imote (see tables 3-1, 3-2, 3-3 and 3-4). The Mica2 and Mica2Dot represent the de facto standard platform for sensor networksdiffering only in form factors and slightly different core resources. The Tmote Sky is a modern, ultra low-power architecture using standard components while the Imote represents a completely different approach of a high-performance, custom node-on-a-chip. All of these platforms are designed to run custom TinyOS WSN applications (see section 3.1.2 for details) without the capability to interface to other wireless enabled devices. The data presented in the tables are derived from device datasheets in the case of the commercial Mica2, Mica2Dot and Tmote Sky and the relevant publications in the case of the Imote [KAH+04, NKA+05]. Especially in the case of the power consumption figures it is hard to derive objective and true figures since the conditions under which these figures are attained vary and often, the platform documentation cites subcomponent datasheets only (for the Mica and Mica2Dot), as opposed to measurements on the whole, live system (Tmote Sky and Imote).

In this section, the four selected platforms are compared against each other. For the sake of completeness, the platform comparison further includes the BTnode rev3 that is introduced in detail later in this chapter in section 3.4. The comparison is based on the relevant system core features (see tables 3-1 3-5 and 3-8) presented in Figure 3-1, the radio system properties (see tables 3-2, 3-3 and 3-6) in Figures 3-2 and 3-3 as well as the power consumption and power supply properties (see tables 3-4 and 3-7) in Figure 3-4. In the case of the dual radio BTnode platform a distinction is made between the low-power radio and the Bluetooth radio. We will use this comparison to motivate and illustrate the design choices made for the BTnode platform in section 3.3.

3.2.2.1 System Core

Table 3-1: State of the art platform comparison – system core features

	Mica2[a]	**Mica2Dot**[a]	**Tmote Sky**[a]	**Imote**[b]
Microcontroller	ATmega128l	ATmega128l	MSP430F	ARM7
Architecture	8-Bit	8-Bit	16-Bit[c]	32-Bit
Speed	7.3728 MHz	4 MHz	8 MHz	12 MHz
Program Memory	128 kB	128 kB	48 kB	512 kB
Data Memory	4 kB	4 kB	10 kB	11 kB[d]
Configuration Mem.	4 kB	4 kB	–	–
Storage Memory	512 kB	512 kB	1024 kB	–
ADC Resolution	10-Bit	10-Bit	12-Bit	–
External IO	51	18	16	30
On-Board Sensors	2[e]	2[e]	5[e]	–
UI Components	3 LEDs	1 LED	3 LEDs, 1 Button	1 LED
Programming Modes	ISP, JTAG, Bootloader	ISP, JTAG, Bootloader	USB, JTAG	JTAG
Re-Prog. Cycles[f]	≤10'000	≤10'000	10'000+	≤500
Size	1856 mm^2	492 mm^2	2621 mm^2	900 mm^2

[a] Typical datasheet values.
[b] Typical datasheet values and as reported by Nachman et al. [NKA+05].
[c] 16-Bit RISC with hardware multiplier.
[d] 64 kB total with 11 kB free for applications on top of the mandatory Zeevo OS and radio stack.
[e] These values include an integrated battery monitor.
[f] This values specifies the amount of cycles that the Flash program memory supports.

The system core is evaluated so that CPU architecture and speed, memory sizes, external IO and on-board sensors are all to be maximized whereas the system size is to be minimized to derive the optimal architecture (see Figure 3-1). The Mica2 and Mica2Dot Motes show conservative memory and processing capabilities combined with a default sensing and storage capability. They differ only in respect to size, CPU speed and external IO. The Tmote Sky supports increased processing capabilities, many on-board sensors combined

with ample storage capabilities but lacks clearly in terms of size and especially memory for processing. It is thus most suitable for lightweight applications with long inactive periods and bursty processing demands. The Imote is harnessing considerable processing power and program memory on a small form factor but it lack in terms of integrated sensors, storage and data memory. Especially the restricted amount of re-programming cycles limit the use of the Imote for heavy development work. All four platforms suffer from very scarce data memory resources requiring extreme care during the design and meticulous effort in developing applications that fit into such limited memory footprints of a few kilobytes only.

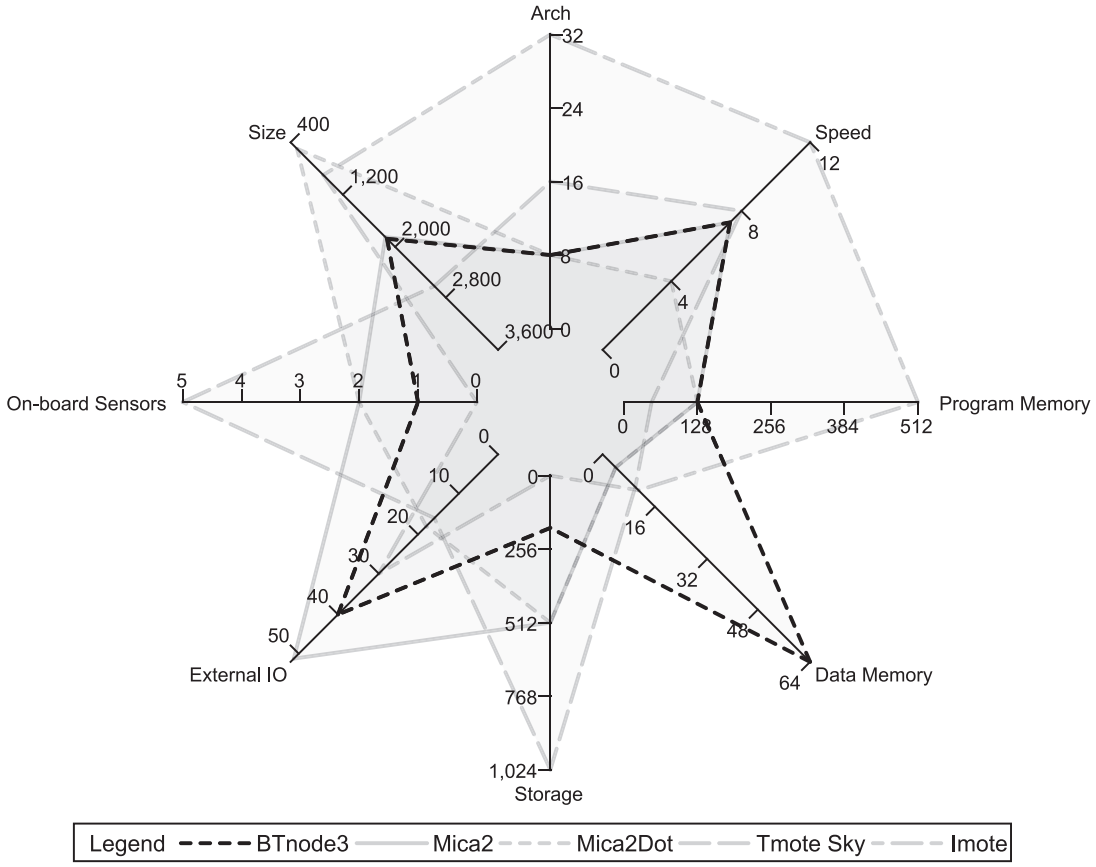

Figure 3-1
Platform comparison – system core features: While the Imote is clearly harnessing considerable processing power and program memory on a small form factor, it lacks in term sensors and storage. The Mica2 and Mica2Dot differ only in size, speed and the amount of external IO. The BTnode rev3 (see tables 3-5 and 3-8 for details) supports ample memory and IO but only one sensor, clearly distinguishing it from the minimal memory support and large variety of sensors provided by the Tmote Sky.

3.2.2.2 Radio Systems – Physical Properties

Whith the system core features given as objective facts, the radio features already pose considerable problems and room for interpretation. The cause for this is mainly the broad

range of options and configurations possible on most of these radio devices. Although designed for standardized interoperability, a multitude of protocols and operating schemes are available and the performance of a single solutions is so highly dependent on the requirements of the underlying application. While the physical parameters such as frequency, modulation, transmit power and sensitivity are more of interest to the radio designer the data rate, radio range (cell size), amount of distinct channels and setup time are very relevant to application design and performance. Furthermore the interface modalities under which a radio is connected to a host CPU and the services offered by the radio are of concern (see also section 3.2.2.3).

Table 3-2: State of the art platform comparison – radio physical properties

	Mica2[a]	**Mica2Dot**[a]	**Tmote Sky**[a]	**Imote**[b]
Radio	Chipcon CC1000	Chipcon CC1000	Chipcon CC2420	Zeevo TC2001
Standard	ISM	ISM	802.15.4	Bluetooth 1.1
Frequency Band	315-916 MHz	315-916 MHz	2.4 GHz	2.4 GHz
Data Rate	38.4 kbps	38.4 kbps	250 kbps	723.2 kbps
Setup Time	<50 msec[c]	<50 msec[c]	<1 msec	<500 msec
TX Powerctrl	30 dB[c]	30 dB[c]	24 dB[c]	–
TX Power[d]	–/+10 dBm[c]	–/+10 dBm[c]	-3/+0 dBm	+0.5/+4 dBm
Sensitivity	-101 dBm[c]	-101 dBm[c]	-94 dBm	-80 dBm
Modulation	FSK	FSK	DSSS-QPSK	FHSS-GFSK
Int. Antenna	–	Wire	embed. PIFA	GigaAnt
Ext. Antenna	MMCX conn.	–	SMA conn.	U.FL conn.
Outdoor Range	150 m[c]	150 m[c]	125 m	30 m
Indoor Range	40 m[c]	40 m[c]	50 m	30 m
Channels	4	4	16	79
Max. Endpoints	MAC specific	MAC specific	16-bit	7S, 4M

[a] Typical datasheet values.
[b] Typical datasheet values and as reported by Nachman et al. [NKA+05].
[c] Values are frequency (and data rate) dependent.
[d] Typical/maximal datasheet values.

Of the seven properties depicted in Figure 3-2, all but the setup-time, which is to be minimized are to be maximized for best performance. Although the Mica2, Mica2Dot and BTnode rev3 with low-power radio (BTnode3 LPR) are using the same radio they are typically specified for different protocol variants resulting in different data rates and radio ranges. Clearly situated at one extreme of the design space when compared to the link-oriented 802.15.4 and Bluetooth radios used in the Tmote Sky, Imote and BTnode rev3 with Bluetooth (BTnode3 BT), the broadcast-oriented CC1000 ISM radio supports only few distinct channels resulting in a low capacity. When operated at high transmit power levels the radio ranges achieved are also high, further limiting the capacity of a network. The Tmote Sky shows a different approach with lower transmit power, higher sensitiv-

ity and bandwidth, a number of dedicated channels and extremely low setup-time. The Bluetooth radios on the Imote and BTnode rev3 Bluetooth are situated at the other end of the design space supporting a very high data rate, the highest amount of channels and so resulting in considerably higher capacity. It must be noted however, that the setup-times displayed here are for the hardware resource alone and do not encompass the time necessary to set up a communication link and successfully transmit data.

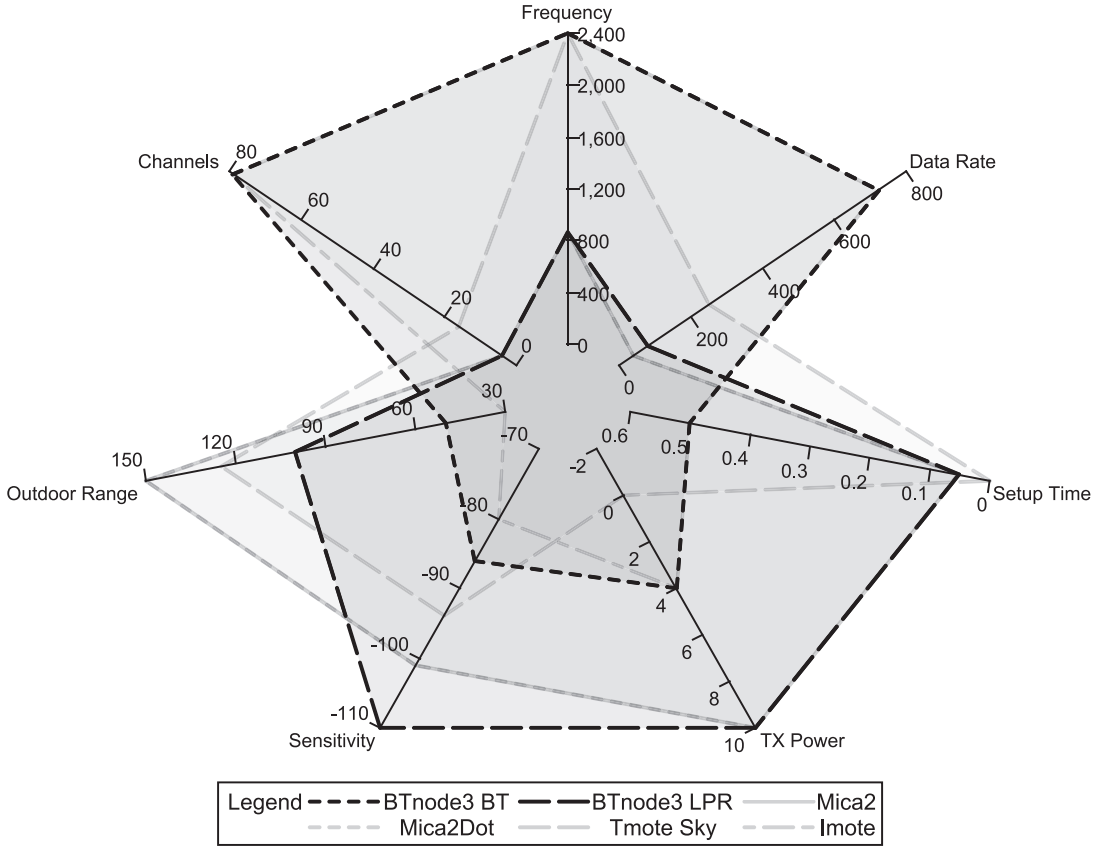

Figure 3-2
Platform comparison – radio physical properties: The BTnode rev3 Bluetooth radio (see table 3-6) and Imote cover an almost identical region supporting high bandwidth and capacity but low range and long setup time (top shaded area). The Mica motes and the BTnode rev3 low-power radio follow a different approach that spans the bottom half of the diagram (bottom shaded area). All are using the same chip but specified values and test environments are different resulting in the variation presented. The Tmote Sky supports considerable datarates, very low setup time and low transmit power yielding.

3.2.2.3 Radio Systems – Communication Interface Abstractions

The five platforms selected for this comparison use four different radios of which the Imote and BTnode rev3 Bluetooth radios differ only slightly. We are only discussing the ZV4002 here as the Zeevo TC2001 and ZV4002 are follow-up products based on the same core design. A key difference yet to be observed stems from the communication interface, services and abstractions offered by the different radios. Here, three different distinctions can be

made: While the Chipcon CC1000 is a simple RF transceiver, using a bit-stream interface, lacking support for packet and address detection as well as error correction and acknowledgments, the CC2420 and especially the Bluetooth devices follow a different paradigm (see section 4.1 for an in-depth presentation of Bluetooth). The interfaces on the CC2420 and ZV4002 offer considerable internal memory for the buffering of whole packets, and in the case of the Bluetooth radios, a comfortable asynchronous interface based on commands and events. Many of the key baseband functions are available internally and are executed transparently with minimal intervention required by the host CPU. This results in different driver and protocol stack requirements that impact design decisions and lastly the capabilities of the applications themselves. The CC1000 radio interface is basically made up of two bit-streams of the date being transmitted and received over the RF front end while the CC2420 and especially the Bluetooth radios offer packet or even link oriented abstractions. The drivers and protocol stack for the CC1000 have to provide most baseband (packet framing, address and error detection) and all MAC layer functionality on the host CPU. While the Bluetooth radios are clearly at one end of the design space with link orientation and a high-level interface supporting a multitude of protocol functions, the ISM radios such as the CC1000 are at the other end with all low-level, bit-stream and protocol processing, above the (de-)modulation and analog-to-digital conversion being offloaded onto the host controller (see Figure 3-3).

Table 3-3: State of the art platform comparison – baseband and interface abstraction

	Chipcon CC1000	**Chipcon CC2420**	**Zeevo ZV4002**
Packet Detection	no	yes	yes
Address Decoding	no	yes	yes
Encryption Support	no	128-bit AES	128-bit SC
Error Detection	no	yes	yes
Error Correction	no	2-bit FEC	config. FEC
Acknowledgments	no	yes	yes
Host Controller Interface	synchronous byte	synchronous packet	asynchronous command/event
Internal Buffering	1 byte	128 byte	1x256 byte HCI, 5x339 byte ACL
Time Synchronization	SFD/byte	SFD	internal
Link Quality Indicator	no	yes	yes
RX Signal Strength	yes	yes	yes

3.2.2.4 Power Consumption

Table 3-4 gives values for the typical energy supply and power consumption in characteristic operating states that are common across all platforms. These are a deep sleep mode where the CPU is running a timer only and the radio is turned off (CPU sleep, radio off), a mode where solely the CPU is operating (CPU on, radio off), an idle-listening mode

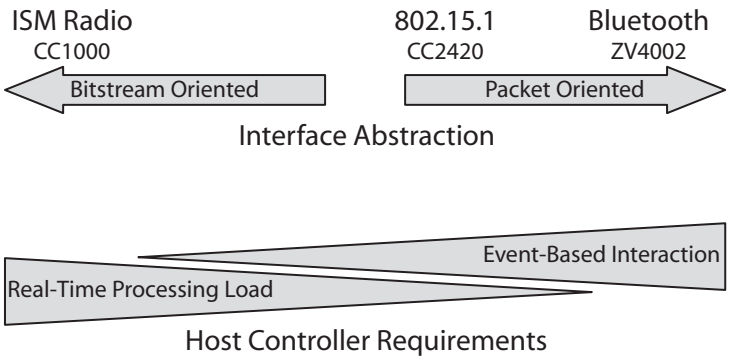

Figure 3-3
Platform comparison – communication interface abstractions: ISM radios such as the CC1000 require extensive real-time processing on a host controller while packet-oriented radio systems such as Bluetooth support high-level event driven interfaces.

where both the CPU is on and the radio is ready to receive data (CPU on, radio listen), a mode with data transmission at the nominal data rate (CPU on, radio RX/TX) and finally a worst case mode (Max. Power).

Table 3-4: State of the art platform comparison – power supply and consumption

	Mica2[a]	**Mica2Dot**[a]	**Tmote Sky**[b]	**Imote**
Battery Supply	2 AA cells	1 coin cell	2 AA cells	2 CR2 cells
Minimum Vcc	2.7 V[c]	2.7 V[c]	2.1 V[d]	3.0 V
Battery Capacity	2000 mAh	560 mAh	2900 mAh	1600 mAh
Regulated Supply	–	–	–[d]	yes
CPU sleep, Radio off	0.054 mW[e]	0.054 mW[e]	0.0153 mW[e]	9 mW[f]
CPU on, Radio off	36 mW	36 mW	5.4 mW	27 mW[f]
CPU on, Radio listen[g]	66 mW	66 mW	65.4 mW	62.1 mW[f]
CPU on, Radio RX/TX	117 mW	117 mW	58.5 mW	112.5 mW[h]
Max. Power	165 mW	165 mW	69 mW	195 mW[h]

[a] Typical datasheet values. Power consumption values computed by summation of individual power consumption of system core, flash memory (Flash) memory and radio components given in datasheets at Vcc=3.0 V.
[b] Typical datasheet values as reported for the whole system. Power consumption computed at 3.0 V.
[c] Operation from rechargeable cells (Vcc=2.4 V) unreliable.
[d] The system is fed directly from batteries with a regulated power supply at 1.8 V available for the radio transceiver only. The CPU requires a minimum Vcc=2.1V.
[e] Wakeup possible from internal timer only.
[f] Power measurements on live system at Vcc=3.0 V as reported by Nachman et al. [NKA+05].
[g] Radio listening typically assumes a worst case of CPU active and radio RX at the nominal datarate.
[h] Datasheet values at Vcc=3.0 V for radio SoC only (no peripherals).

The values for the Mica2 and Mica2Dot are derived from the Crossbow datasheets and are given for an operating voltage of Vcc=3.0 V which is equivalent to two new AA cell batteries. The Tmote Sky is either powered directly from 2 AA cell like the Mica2 or powered from a USB interface using an internal regulator. Since the primary interest here is in the supply from batteries, the system power consumption values are given for the operating voltage of Vcc=3.0 V. The power supply for the CC2420 radio is derived from

an internal 1.8 V regulator integrated in the radio transceiver. On all three systems the energy consumption is reduced when lower operating voltages are used. The table lists the lowest rated values for the operating voltages. Of course, on has to keep in mind, that a lower voltage level on a battery is also an indicator for the remaining available battery power. It is up to the system and application designer to assure a termination of the application on low battery levels if this can cause malfunction or misbehavior.

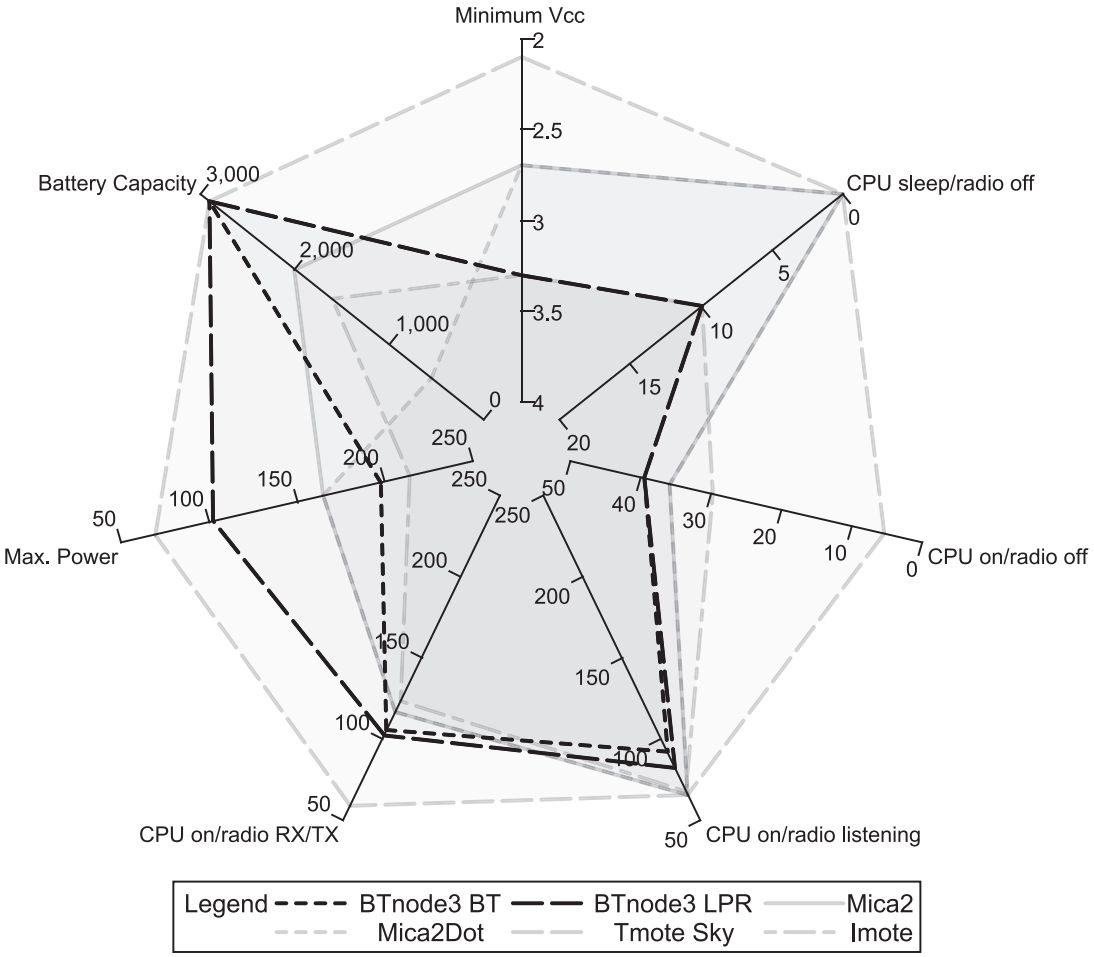

Figure 3-4
Platform comparison – power supply and consumption: being the only ultra low-power system in the comparison, the Tmote Sky clearly dominates the power consumption scenario when compared to the other platforms. The considerable power consumption in sleep mode for the BTnode rev3 (see table 3-7) and Imote can be attributed to the regulated power supply and the extended SRAM storage memory available on these devices. The Bluetooth radio systems compare quite favorably to the CC1000 based systems here.

Due to it's complex architecture, the Imote uses a different power supply concept with a regulated power supply. Here, the internal operating voltage of Vcc=3.0V is generated using a low-noise, low-dropout linear regulator. The standard power supply consists of two parallel CR2 cells at 6.0 V each. The power consumption was measured at the input of

the linear regulator at 6.0 V [NKA+05]. Since the linear regulators operational principle is to burn up all excess energy, the values given in table 3-4 were converted to the actual internal operating voltage Vcc=3.0V to give an objective comparison.

3.2.3 Problematic Platform Metrics

As mentioned already in the previous sections, objective metrics and comparisons between platforms are hard to obtain. Apart from the many variations and options possible in systems design, different design objectives lead to viewpoints and design goals specific to every single case. What seems suitable in one case may or may not be adequate in another. We will shed some light onto this chasm using two examples in the following.

3.2.3.1 Problematic Platform Metrics – Power Consumption

Of all criteria used to describe and compare platforms for wireless sensor networks, the power consumption is the most difficult to assess. The main reason for this is, that it is a key characteristics of WSN devices to be low-power, in fact to operate energy conserving and that the energy supply for every node is finite. Although proposed early on, only few have so far considered renewable energy sources [RWR03] and built complete prototype systems [ROC+03, JPC05]. The second reason for problematic and often incomparable results in power consumption is the varying settings and different architectures of the devices used and applications being compared. In most cases datasheets give detailed results about the average (typical) and worst case power consumption at a nominal operating voltage for the single device that the datasheet is referring too. Missing are further values for additional system components, such as clock generation, drivers, latches, memories, LED etc. that are required for a fully operational system. Often, different core voltages are used for internal components, while interfaces are operating at higher voltages. In the case of an unregulated battery supply, the voltage driving a system can vary considerably over time and with the respective load. Since the power consumption is dependent on the supply voltage, it is thus required to give exact values for the current consumption and the respective voltage level at which these have been measured for every operating state concerned.

Single power consumption values for steady states are easy to measure but need to be put into an application context where the power consumption varies over time, i.e. in a duty cycled application where the radio switches from transmit to receive and possibly also to sleep of off, the setup times between these states add to the overall system power consumption. For heavily duty cycled applications such as wireless sensor networks the most important parameter here is the setup time from a low power sleep state to a ready to transmit or receive state. For different radio types this setup time can vary considerably as can be seen in tables 3-4 and 3-6.

To enable a fair comparison of platforms and applications, a practice of measuring the whole system in live operation at it's energy supply, i.e. the battery, should be adopted rather than resorting to values for single subsystems or components only (cited from datasheets). The latter is more of important in respect to system design aspects, where

concrete questions that concerning single components only are being asked, e.g. the effect of different power modes on a microcontroller or different transmit power levels on a radio transceiver. Although two different architectures and operating at different operating frequencies, the power consumption values given in the datasheets for the Mica2 and Mica2Dot are the same. The values presented in publications derived from measurements [PSC05] differ up to about 300% between the two platforms and up to about 200% between publication and datasheets.

Through measurement of the system energy consumption at the energy supply, all system components can be incorporated into a comparison. To define the minimum power consumption for a system, it is usually desirable to measure currents at the lowest operating point, i.e. the minimum stable operating voltage. Often, operation below the minimum voltage specified by the datasheet is still possible and the knowledge of this threshold voltage is an important fact to know about a system. However, stable operation cannot be guaranteed nor can this argument be used as a selling point to the user.

It is also misleading to mix values for the current consumption given in mA and power consumption given in mW for a general platform comparison. The application designer usually cares about the longevity of a system and thus the energy supply and dissipation over time. So in general, for an application designer, power consumptions should be given in mW.

Architectural differences are another source of inaccuracies. Not all systems require stabilized power supplies where large amounts of energy are typically dissipated in the converters. Operating states on platforms differ, as not all system offer the same modalities of operation and states.

Elaborate platform characterization like for the Mica2 [SHC+04], the Tmote Sky [PSC05] and the Bluetooth part of the BTnode rev3 [NBD05b] are not very common in the literature but give detailed insight into the power performance and behavior of systems. They reveal interesting characteristics and properties, such as considerable power consumption for running LED (2 mA @ 3.0 V) and memory access (up to 10 mA @ 3.0 V), that are often overseen. They lay the foundation for precise system models useful in simulation (e.g. PowerTOSSIM [SHC+04]) or power-adaptive protocols (e.g. Negri [NBD05b]).

3.2.3.2 Problematic Platform Metrics – Capacity

Radio systems are predominantly characterized by bandwidth, data rate, range and power consumption. Often forgotten at first is the system capacity, i.e. the ability of a system to actually make use of the resources available and so the limits of any application. The theoretical background here is generally considered as landmark publications [GK00] but is only valid in a general context. Theory usually takes into account perfect conditions, e.g. asymptotically large ad hoc networks, perfect scheduling (no collisions), perfect knowledge of conditions and neighbor states, etc.

The term capacity, in the sense of Shannons channel capacity denotes the theoretical upper bound for a given channel. While these results are clearly of general nature, they need to be adapted when applied to real networks. Here, a networks capacity is also limited by the

Shannon bound but also by the capacity of the protocols used, that do not only consist of the raw data transmitted over a given channel. Multiple access schemes used divide the available medium in dedicated channels of a defined bandwidth. In the case of a time shared access, multiple users have to negotiate to use a single channel, i.e. with an increasing number of users producing traffic the channels get jammed. Since a transmitter covers a certain area in space for the time it is transmitting (i.e. a cell), single channel radios using broadcast primitives such as available on the CC1000 radios support much less capacity in a given area than radios with dedicated links such as the CC2420 or especially Bluetooth radios. In the case of a Bluetooth radio the increase in resource consumption is payed off by a considerable aggregate bandwidth of theoretically up to 79 channels per area at a given time. Each channel in Bluetooth (up to standard v1.2, see section 4.1) supports links of up to 723.2 kbps allowing a sustained aggregate bandwidth of 57.1 Mpbs in cells of up to 10-30 m. In practice, both simulation [NWS03] and experimental results [KL01] have confirmed the feasibility of operating about 50 simultaneous channels, which leaves us with roughly 2/3 of the limit outlined in the standard. This is considerably more than what can be achieved with the CC1000 radio using 4 channels at 38.4 kbps and covering cell sizes of up to 40-150 m (Mica2).

Given these estimates based on the raw data rates without congestion and loss of data, one can easily argue in favor of one solution or the other. A straightforward conversion to energy dissipation per bit transmitted, as is often done for radio transceivers, reveals Bluetooth as the true winner, while mostly idle channels with occasional bursts favor a CC1000 radio. Depending on the application characteristics, it is not sufficient to just relate to the theoretical capacity, but metrics must be combined with application duty-cycle, traffic patterns, the number of nodes and the region covered by their transmissions (multiple access scheme and range), MAC and link-layer properties.

3.3 Prototyping Sensor Networks – A Design Rationale for Modular Platforms

Wireless sensor networks have been anticipated to become a pervasive tool that would enable detailed and unobtrusive observation of real-world phenomena, thus bringing substantial benefits to a variety of application areas. A number of advanced proof-of-concept systems have been successfully deployed, suggesting that the remaining gap between these prototypes and real-world applications of sensor networks has been significantly narrowed. We have discussed a selection of platforms supporting such applications in the previous section. Each of them was however designed for specific application scenarios, each with their own strict requirements and constraints resulting in a loss of generality and modularity. Of course, many researchers and practitioners are using these platforms for other application domains than they were originally conceived for, but today there are equally many that are building their own costly, full-custom systems. We believe that in the future the total cost of applying a sensor network will be dominated by the development process as per-device-cost are expected to drop to few Dollars. Hence, significant efforts are

needed to approach the goal of consistent and coordinated design, programming, testing, debugging, validation and deployment of sensor network applications – which is mostly non-existent today. Also, modular reuse of existing hardware and software components must replace current practice of costly custom design and development processes.

The Berkeley family of Motes and TinyOS of which we have discussed and compared three members (Mica2, Mica2Dot and Tmote Sky) in the previous section certainly target modularity and general applicability and to some extent set todays standards, yet they are inherently custom platforms. A modular wireless networked embedded system for widespread use in research and development should not be limited to a single application, e.g. through the use of custom hard- and software, or specialized on-board sensors, support ample resources,e.g. to accommodate early development code with debugging support, have an easy to use radio interface and support a widespread programming standard, e.g. the C programming language, where no custom tools need to be maintained. Most important however, is the ability to interact with other wireless enabled devices and the human user. To think of sensor networks as fully autonomous and self-contained without interaction is a contradiction of the vision put forward by sensor networks. Observation and interaction with the physical world through a sensor network requires to be able to access a sensor network through a heterogeneous set of interoperable devices.

Key properties missing on the Berkeley family of Motes for a modular and development-oriented system are (i) the interaction with other wireless enabled devices through (ii) standardized, high-level, event-driven interfaces, (iii) flexible and sufficient support of system resources and (iv) development, test and deployment support through standardized tools and methodologies.

3.3.1 Modular Platform Requirements

Event-Driven Interaction

Interaction and interoperability with other wireless enabled devices has been proposed in different flavors, such as clustered architectures [WEG+03a, WEG03b], where the clusterheads are equipped with more computing and memory resources to process and aggregate sensor data collected from the nodes in their cluster, isolated sensor network patches being linked to a base station and database for the storage and query of results [MCP+02], actuation or notification on the occurrence of an event [FRL05a, FRL05b] or user interaction scenarios where devices with well acquainted user interface capabilities such as mobile phones or PDA enable human users to interact with sensor network nodes [Sie04b]. All of these interactions with systems and applications have in common, that they are not inherently data-driven but driven by events and user interaction, which is not very well supported by the standard Mote/TinyOS architecture today.

Systems and platforms should be designed based on asynchronous, event-based programming models to support interaction and interoperability between devices and with the human users.

Standardized Interfaces

Tiered architectures constructed from multiple sets of custom device architectures have been proposed [HHKK04], but ultimately this alone will not be sufficient to support the heterogeneity of devices and systems found today. Specifically, heterogeneity should be supported through standardized interfaces and application programming interfaces that use high-level abstractions that can be used by non-experts in a similar way as wireless communication specialists. Just supporting different radios to choose from within a system framework as is the case for TinyOS is not sufficient here.

Devices must support interoperability between different classes of devices using high-level communication interface abstractions.

Flexible and Sufficient Resources

Test and development systems usually need more resources than a final production system that is highly optimized to it's designated task. The Motes are inherently ill equipped with memory for data storage requiring the utmost care in programming. This lack of flexibility considerably restricts the ease of use and the degree of freedom to try different approaches and to debug applications. The same flexibility is lost when communication resources are scarce. High data rates, high capacity and robust link-layer abstractions are key to success and can always be scaled down to be more conservative in resources at a later stage if required. Too many and too specific sensors tightly integrated with the nodes hinder flexible application design. Specific sensors can always be added according to the application requirements using external add-ons and usually need to be specifically tailored to this application.

Ample memory resources, sufficiently high data rates and flexible IO are required for many different applications alike.

Development and Deployment Support

The development of algorithms and applications for sensor networks is non-trivial for various reasons. Firstly, sensor networks are highly dynamic distributed systems, where a consistent view of the global system state is typically not available to the developer. Secondly, the behavior of the sensor network is highly dependent on the physical environment, such that problems might not be easily reproducible. Thirdly, for reasons of energy efficiency, sensor network applications do often perform in-network data processing and aggregation, where raw sensor data is already processed and evaluated inside the network in order to reduce the volume of data which has to be transmitted. To verify and debug such systems, the developer often needs access to both, the raw sensor data and the aggregated output of the sensor network [EBB+03]. Equally important as the support of debugging and deployment are standardized tools and methodologies, such as the C programming as the prevalent standard in programming embedded systems. Independence of development processes in ongoing research projects (such as nesC in the case of TinyOS) and expensive custom tools (the ARM development suite in the case of the Imote) as well as community

support through open models foster a fast learning curve and ultimately success with a platform. Many platforms today require too many custom tools, knowledge or methodologies to develop applications successfully.

Open, proven and standardized programming models with the necessary support and documentation are key to success.

The BTnode platform tries to close the gap put forward by these specific platform requirements and provide a modular and easy to use platform for prototyping wireless network embedded systems.

3.4 The BTnode Platform

The BTnode is a versatile, lightweight, autonomous wireless communication and computing platform based on a Bluetooth radio and a microcontroller.

The device is designed for fast-prototyping [3] of ad hoc and wireless sensor network applications and is well suited to investigate different protocols, operation parameter tradeoffs and radio alternatives. The following design criteria were followed apart from the requirements layed out above.

- Small form factor, low component count
- Standardized wireless interface
- Flexible and cost effective deployment of large quantities of networking nodes

First architectural considerations required a device magnitudes smaller than a personal digital assistant (PDA) but equally flexible and programmable and supporting Bluetooth. In the context of a wireless prototyping platform, Bluetooth has the advantage of being available today for experimentation, compatibility to interface to many consumer appliances and an abstract, standardized high level digital interface. The first usage profile envisioned the BTnodes to serve as a demonstration platform for research in mobile ad hoc network (MANET)s and distributed sensor networks as well as ubiquitous and pervasive computing.

The device has no integrated sensors, since individual sensor configurations are required depending on the application. Instead, with its many general-purpose interfaces, this platform can be used with various peripherals, such as sensors, but also actuators, digital signal processors (DSP), serial devices (like GPS receivers, RFID readers, etc.) and user interface components.

An interesting property of this platform is its considerably small form factor while still maintaining a standardized wireless interface.

The BTnode hardware can run TinyOS [LDB03, BD05], or the BTnut system software that is based on an Ethernut kernel and allows cooperative multi-threaded applications with standard C programming (see section 3.4.3).

3.4.1 BTnode Hardware Generations

Starting out from an all-in-one Bluetooth module, the BTnode family of nodes has matured to a full fledged dual radio wireless networking platform. The first prototypes were built using a microcontroller development kit and a Bluetooth evaluation kit that were assembled for first functional testing and software development. After that first BTnode rev1 prototypes were produced that led to the development of the BTnode rev2 (see Figure 3-5) and BTnode rev3 (see Figure 3-9).

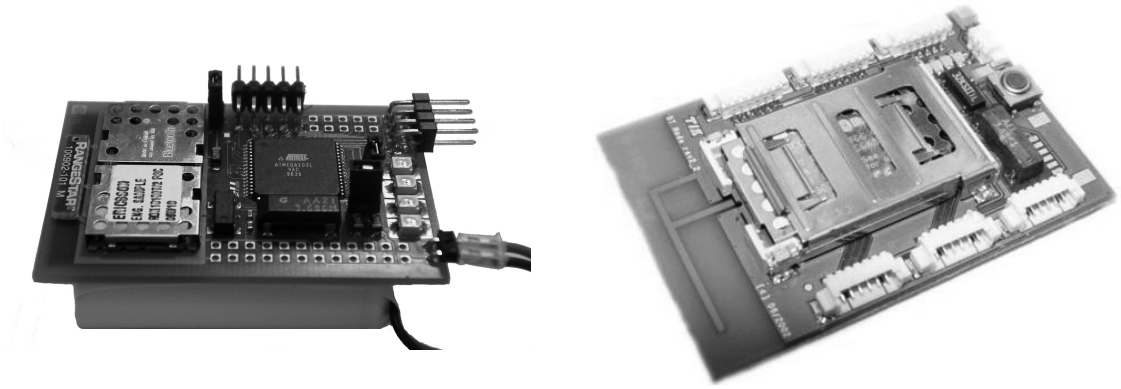

Figure 3-5
The BTnode rev1 prototype (left) and the BTnode rev2 hardware (right) that was series produced and distributed to researchers with 200 units.

3.4.1.1 The BTnode rev2

The BTnode rev2 hardware [BKR03a] is built around an Atmel ATmega128L microcontroller with on-chip memory and peripherals (see Figure 3-6). The microcontroller features an 8-Bit RISC core delivering up to 8 MIPS at a maximum of 8 MHz. The on-chip memory consists of 128 kilobytes of in-system programmable Flash memory, 4 kilobytes of SRAM, and 4 kilobytes of EEPROM. There are several integrated peripherals: JTAG for debugging, timers, counters, pulse-width modulation, 10-Bit analog-digital converter, inter-integrated circuit (I2C) bus, and two hardware universal asynchronous receiver transmitters (UART). An external low-power SRAM adds an additional 240 kilobytes of data memory to the BTnode system.

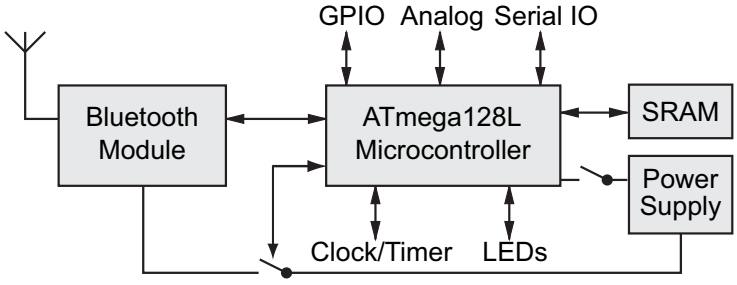

Figure 3-6
The BTnode rev2 hardware system overview shows the three basic components: Communication (Bluetooth radio), computation (Atmel system core) and IO/peripherals (sensor connectors and peripherals).

A real-time clock is driven by an external quartz oscillator to support timing updates while the device is in low-power sleep mode. The system clock is generated from an external 7.3728 MHz crystal oscillator.

An Ericsson Bluetooth module is connected to one of the serial ports of the microcontroller using a detachable module carrier, and to a planar inverted F antenna (PIFA) that is integrated into the circuit board. This is a very simple and practical antenna design when space on the circuit board is available..

Four light emitting diodes (LED) are integrated, mostly for the convenience of debugging and monitoring. One analog line is connected to the battery input and allows to monitor the battery status. Connectors that carry both power and signal lines are provided and can be used to add external peripherals, such as sensors and actuators.

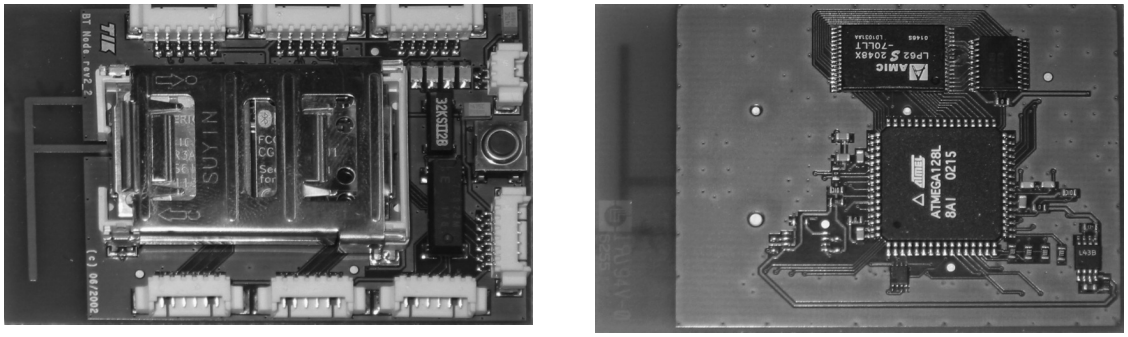

Figure 3-7
BTnode rev2 hardware contains a Bluetooth module, an antenna, user interface components and connectors for power-supply and sensors on the top (left), the microcontroller and external memory on the bottom (right).

3.4.1.2 The BTnode rev3

The BTnode rev2 has been revised, mainly (i) to make use of a new Bluetooth subsystem and (ii) to incorporate a second, low-power radio which is identical to the one used on the Berkeley Mica2 Motes. This makes the BTnode rev3 a twin of both the Mica2 Mote and the old BTnode rev2. Both of its radios can be operated simultaneously or be independently powered off when not in use, considerably reducing the power consumption of the BTnode. This new device provides opportunities to create tiered architectures with high-bandwidth nodes bridging ultra-low-power devices like the Berkeley Motes to Bluetooth-enabled gateway appliances [HHKK04], or to investigate duty-cycled multi-front end devices with wake-up radios [SBS02] or bandwidth–power–latency trade-offs [RPW+04].

The Ericsson Bluetooth subsystem on the BTnode rev2 was discontinued in 2002 making further development based on this device impossible. An assessment of the platform status and the experience gained with applications resulted in the following extended design criteria for the BTnode rev3:

- Bluetooth 1.2 subsystem supporting multiple-master scatternets

- Additional low-power radio
- Integrated, battery-powered power supply
- Board-to-board extension connector
- Availability and open documentation of radios (developer information)

The design of the BTnode rev3 resulted in a dual-radio wireless networking platform (see Figure 3-9). The low-power radio (Chipcon CC1000) is the same as used on the Berkeley Mica2 Motes. The Bluetooth radio is a Bluetooth 1.2 compliant device (Zeevo ZV4002) with radio circuits, baseband, MAC, link controller and an ARM7 core integrated on a single SoC. Both radios can be operated simultaneously or be independently powered off using microcontroller controlled power switches (see Figure 3-8).

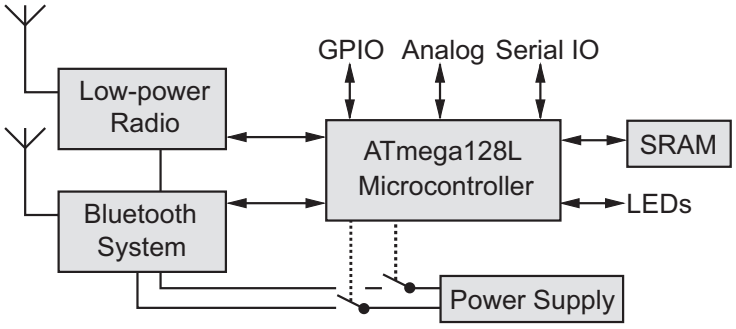

Figure 3-8
BTnode rev3 system overview with dual radios: Two independent radios can be operated simultaneously or be individually powered off when not in use. A regulated power supply operates from dual sources (battery and external DC input).

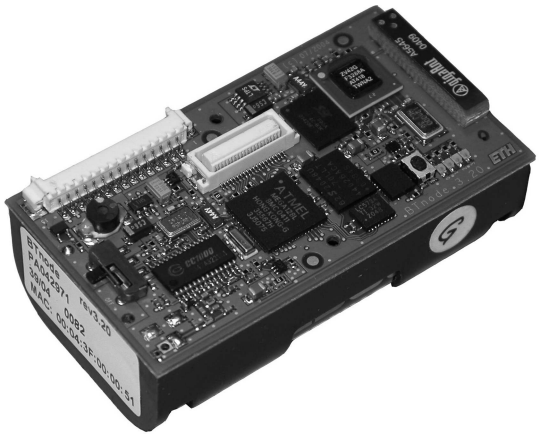

Figure 3-9
BTnode rev3 hardware: All components are mounted on a single sided assembly with the printed circuit board mounted on a 2 AA cell battery holder. The AVR system core is in the middle with the two radios on the sides: The Chipcon low-power radio is situated on the bottom left and the Zeevo Bluetooth on the upper right corner.

3.4.2 BTnode Platform Characteristics

The characterization of an embedded platform such as the BTnode is not straightforward since it already consists of a number of subsystems that are complex embedded systems by themselves. Thus the characteristics described in the following often depend on the setup and combination of the subsystems involved. In many cases simplifications have to be made. These are noted in the following tables.

3.4.2.1 System Core and Memory Capabilities

The system core consist of a microcontroller, it's auxiliary circuits like clock, IO components and memories. The memories can be separated into volatile and non-volatile as well as program and data memories. Typically the program memory consists of non-volatile Flash memory and the data memory of volatile SRAM memory, both available on-chip on modern devices. Furthermore a non-volatile configuration memory can be used for storage of configuration information and parameters specific to the device or application and a storage space for application specific data such as sensor values, logs or cached network packets. Depending on the type of program memory and the programming mode used there can be considerable limitations in the amount of programming cycles possible. For the Atmel AVR family typically in-system programmer (ISP) programming is used, but JTAG and a resident bootloader are also available (see section 3.4.2.5).

Table 3-5: BTnode family – system core features

	BTnode rev1	**BTnode rev2**	**BTnode rev3**
Microcontroller	ATmega103l	ATmega128l	ATmega128l
Architecture	8-Bit RISC	8-Bit RISC	8-Bit RISC
Speed	3.6864 MHz	7.3728 MHz	7.3728 MHz
Program Memory	128 kB	128 kB	128 kB
Data Memory	4 kB	64 kB	64 kB
Configuration Mem.	4 kB EEPROM	4 kB EEPROM	4 kB EEPROM
Storage Memory	–	180 kB SRAM	180 kB SRAM
ADC Resolution	10-Bit	10-Bit	10-Bit
External IO	UART, SPI, I2C, PWM, GPIO	UART, SPI, I2C, PWM, GPIO	UART, SPI, I2C, PWM, GPIO
On-Board Sensors	–	1[a]	1[a]
UI Components	4 LEDs	4 LEDs	4 LEDs
Programming Modes	ISP	ISP, JTAG, Bootloader	ISP, JTAG, Bootloader
Re-Prog. Cycles[b]	$\leq$10'000	$\leq$10'000	$\leq$10'000

[a] This sensor is an integrated battery monitor.
[b] This values specifies the amount of cycles that the Flash program memory supports.

3.4.2.2 Communication Interfaces

The original communication interfaces of the BTnode were Ericsson Bluetooth modules. These are Bluetooth devices operating at powerclass 1 at a maximal output power of 4 dBm. The main difference between the two types used here is the increasing flexibility for network topologies where at first only point-to-point links were available to the full fledged AFH/SFH scatternet support with 4 piconets on the Zeevo ZV4002 device in the BTnode rev3. Here an external power amplifier to increase the transmission range and with transmit power control would be possible but seemed not suitable for the BTnode rev3.

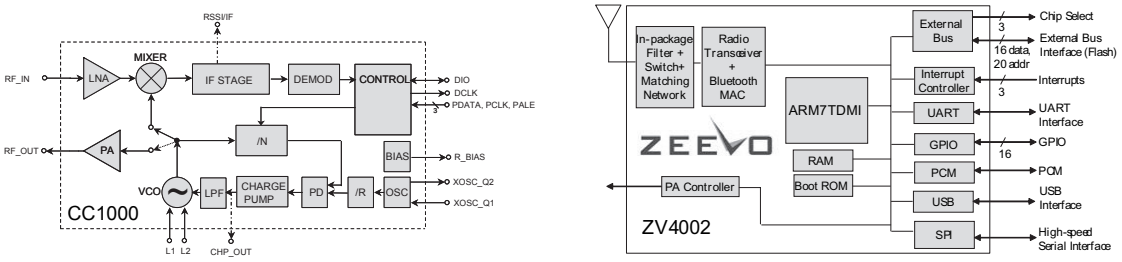

Figure 3-10
BTnode rev3 radio systems – The Chipcon CC1000 (left) is a simple wireless modem transceiver with digital IO and control whereas the Zeevo ZV4002 (right) is a complex wireless embedded system with all radio and processing components integrated and controlled by an embedded ARM7 core.

The low power radio on the BTnode rev3 is quite different from a Bluetooth subsystem. It lacks the advanced protocol processing capabilities and high level application programming interface (API) found on a Bluetooth device. It is more like a modulation engine or data pump with the microcontroller acting as the main protocol processor (see Figure 3-10). This implies that this protocol processor is capable of all baseband, MAC and link-layer processing with packet detection, error correction, retransmission, data sequencing on the serial data-stream to and from the wireless transceiver. Compared to the high-level, link-oriented interface offered by the Bluetooth host controller interface (HCI), this is quite a burden requiring meticulous care in design and implementation. Especially the interleaving of stream-oriented communication interface processing with event driven application processing becomes difficult in the case of more complex applications that do not only consist of polling a sensor and transmitting it's value on a wireless broadcast channel but of interactive and often interdependent processes. In our first implementations (see section 3.5 and chapters 4 and 5), both radios have only been used separate from each other but both a Bluetooth and low-power radio MAC protocol stack are available [RR05].

3.4.2.3 Power Management

The BTnodes are designed to be powered from standard batteries and operate on a single internal voltage level of 3.3 V. For easy in-situ power consumption analysis and flexible power management, the radios and the system core have a separate power feed each that can be switched by the microcontroller (see section 3.4.2.1 and figure 3-12). Due to the high power-quality demands of the Bluetooth radio system a regulated power-supply of $Vcc = 3.3$ V is required. On the BTnode rev1 and rev2 the standard power-supply are three AA cells whereas the BTnode rev3 utilizes a DC-DC step-up converter powered from only two AA cells or alternatively a low-dropout linear regulator powered from an external DC input (see figure 3-11. All power supplies are optimized for low noise and efficient power conversion.

An estimation of the total system power consumption P can be made by summing up the power consumption values $P_i = Vcc \cdot I_i$ measured for the individual subsystems and adding the conversion losses P_c occurring at the regulated power supplies, with $i \in \{bt, cc, core\}$

Table 3-6: BTnode family – radio systems

	BTnode rev1	**BTnode rev2**	**BTnode rev3 Bluetooth**	**BTnode rev3 LP Radio**
Radio	Ericsson ROK101008	Ericsson ROK101007	Zeevo ZV4002	Chipcon CC1000
Standard	Bluetooth 1.0	Bluetooth 1.1	Bluetooth 1.2	ISM
Frequency Band	2.4 GHz	2.4 GHz	2.4 GHz	0.3-1 GHz
Data Rate[a]	–/723.2 kbps	–/723.2 kbps	–/723.2 kbps	38/76.8 kbps
Setup Time[b]	1-2 sec	1-2 sec	<500 msec	<50 msec[c]
TX Powerctrl	–	–	–	30 dB[c]
TX Power[a]	+1.5/+4 dBm	+0/+4 dBm	+0/+4 dBm	–/+10 dBm[c]
Sensitivity[b]	-70 dBm	-70 dBm	-86 dBm	-110 dBm[c]
Int. Antenna	Rangestar	embed. PIFA	GigaAnt	Monopole
Ext. Antenna	–	–	–	MMCX conn.
Outdoor Range[c,d]	30 m	30 m	30-50 m	30-100 m
Indoor Range[c,d]	5-30 m	5-30 m	10-30 m	2-30 m
Channels	79	79	79	4
Max. Endpoints	1 S	7 S, 1 M	7 S, 4 M	MAC specific

[a] Typical/maximal datasheet values.
[b] Typical datasheet values.
[c] Values are frequency (and datarate) dependent.
[d] Typical measured value.

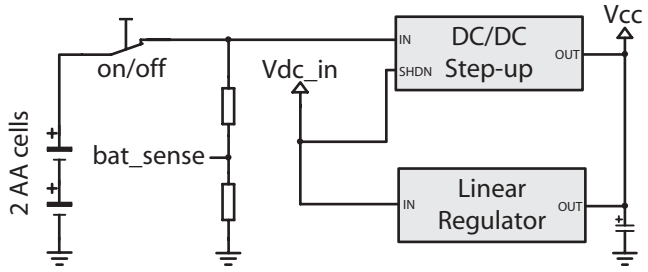

Figure 3-11 BTnode rev3 power management – The default power supply from two AA cells supports battery voltage monitoring and can be shut off on the presence of an external DC input.

and P_c depending on the available power supply and the converter type used.

$$P = \sum P_i + P_c = Vcc \cdot \sum I_i + P_c \tag{3.1}$$

$$P_c = \begin{cases} P_{c_{down}} & \text{for } V_{max} \geq V_{in} \geq (Vcc + V_d) \\ P_{c_{up}} & \text{for } V_{in} = 0 \wedge V_{min} \leq V_{bat} \leq Vcc \end{cases} \tag{3.2}$$

For a linear regulator the voltage in excess of Vcc is dissipated with the low dropout voltage V_d limiting the minimum required input voltage V_{in} [HH89] and $Icc = \sum I_i$ being the sum of the currents feeding the individual subsystems as defined earlier.

$$P_{c_{down}} = (V_{in} - Vcc) \cdot Icc \qquad \text{for } (V_{in} - Vcc) \geq V_d \tag{3.3}$$

A DC-DC step up converter is characterized by it's conversion efficiency η which is dependent on the actual current flow and the battery voltage V_{bat}.

$$P_{c_{up}} = V_{bat} \cdot Icc \cdot (\frac{1}{\eta} - 1) = V_{bat} \cdot \frac{\sum P_i}{Vcc} \cdot (\frac{1}{\eta} - 1) \tag{3.4}$$

In a simple example powered from two AA rechargeable cells and operating all three subsystems simultaneously the following values were measured: $Vcc = 3.3$ V, $I_{bt} = 35$ mA, $I_{cc} = 14$ mA, $I_{core} = 19$ mA This results in an internal power consumption of $\sum P_i = 3.3 \cdot (35+14+19) = 224.4$ mW. Using a feed of $V_{bat} = 2.4$ V and an efficiency of $\eta = 90\,\%$, the conversion loss is estimated to be $P_{c_{up}} = 2.4 \cdot (35 + 14 + 19) \cdot (\frac{1}{0.9} - 1) = 18.3$ mW and the total power system power consumption that needs to be supplied by the batteries $P = 224.4 + 18.3 = 242.5$ mW.

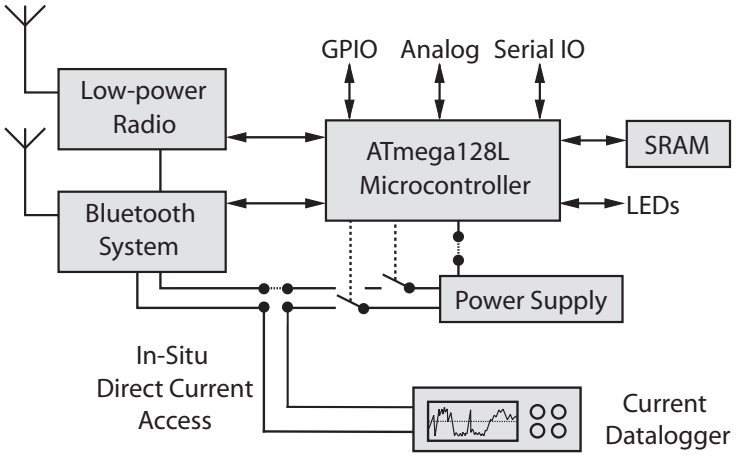

Figure 3-12
BTnode in-situ direct current access is available independently for the two radio systems and the system core by replacing a 0 Ohm resistor in the power supply feeder lines with a current-meter and datalogger. This allows to extract live data logs from the device running under realistic operating conditions.

The power consumption profile given in table 3-7 was measured in-situ on the respective subsystems. For this purpose, three direct current access points are available (0 Ohm resistors in the power supply feeder lines) where in-situ measurements of the power consumption of the radios and the microcontroller core can be performed using a current-meter and datalogger (see Figure 3-12). This allows for very fine grained and subsystem-specific power consumption measurements in the live system under standard operating conditions as opposed to an artificial lab setup with developer boards only. An additional trigger signal from the microcontroller to the datalogger enables to sample values at a given instance in time. Figure 3-13 shows a qualitative example measurement taken from the Bluetooth radio subsystem alone while connected in a point-to-point link.

3.4.2.4 Physical Setup

The physical setup of the BTnode devices is a circuit board with standard surface mount components (SMD/BGA). The design was targeted at low cost, industry standardized

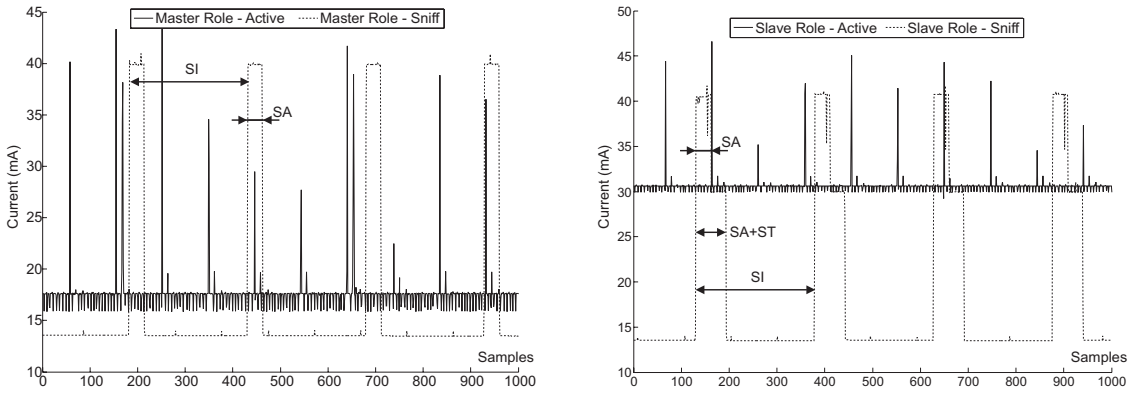

Figure 3-13
Bluetooth power dissipation details: Current dissipation on a Bluetooth point-to-point link in active and sniff mode measured on a master (left) and slave (right) role (sniff mode parameters: SI=5.12 sec, SA=ST=0.64 sec) [NBD05b].

Table 3-7: BTnode family – power consumption

	BTnode rev1	**BTnode rev2**	**BTnode rev3**
DC Supply Input Range	3.6-16 V	3.6-16 V	3.6-5 V
DC Input Dropout Voltage	90/180 mV[c]	90/180 mV[c]	150/180 mV[d]
Battery Supply	3 AA cells	3 AA cells	2 AA cells
Step Up Conversion Efficiency	–	–	90/96 %[e]
CPU power down, radios off[a]	–	0.5 mW	9.9 mW
CPU standby, Radios off[a]	9.9 mW	12 mW	23.1 mW
CPU idle, Radios off[a]	26.4 mW	30.9 mW	39.6 mW
CPU idle, Bluetooth listening[a]	–	–	92.4 mW
CPU idle, Bluetooth Tx/Rx[a]	108.9 mW	67 mW[b]	105.6 mW
CPU idle, Bluetooth Inquiry[a]	148.8 mW	160 mW[b]	198 mW
CPU idle, CC1000 listening[a]	–	–	82.5 mW
CPU idle, CC1000 TX/RX[a]	–	–	102.3 mW
CPU idle, Radios both listening[a]	–	–	135.3 mW
Max. Power	–	–	260.7 mW

[a] All values are typical values measured on live system at 3.3 V.
[b] The Bluetooth module used in rev2 is a different generation Ericsson module compared to rev1 resulting in different power characteristics.
[c] Typical datasheet values at 75/200 mA load current.
[d] Typical datasheet values at 50/100 mA load current.
[e] Typical/maximum datasheet values at 100 mA load current.

surface mounted production and follows the good automated manufacturing practices (GAMP). Standard connectors with good availability were used to connect auxiliary sensor boards or programming/debugging hardware to the BTnode system. The BTnode rev3 uses a single sided assembly that is fitted onto a two AA cell battery holder (see figure 3-14). Two screw-on mounting holes are also available.

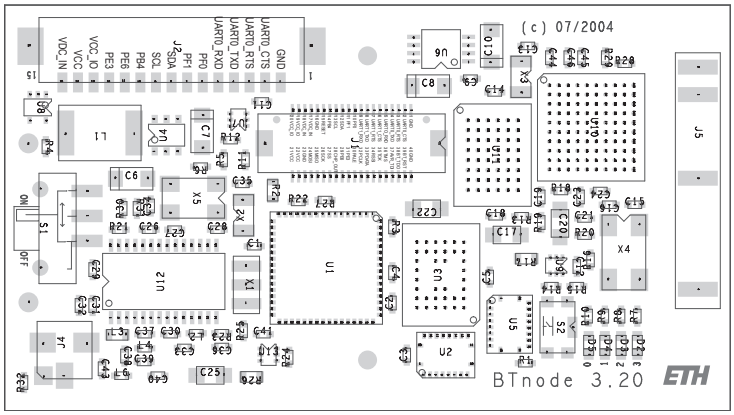

Figure 3-14
The BTnode rev3 consists of a four layer printed circuit board with all surface mount components mounted onto a single side. There are two ways to physically connect to a BTnode. The extension connector J1, or the debug connector J2.

The area breakdown on the topside can be separated into system core, Bluetooth radio, low-power radio, power management, external IO and other areas and results in quite similar partition sizes for all these subsystems. This suggests that compared to the rather complex functionality of the radios and the system core the external IO and power management are crucial design points when it comes to high system integration. This is was even more visible on the BTnode rev2 where 52.6% of the overall area were used for external IO, power management and other areas compared to 47.4% for system core and radio (see Figure 3-16).

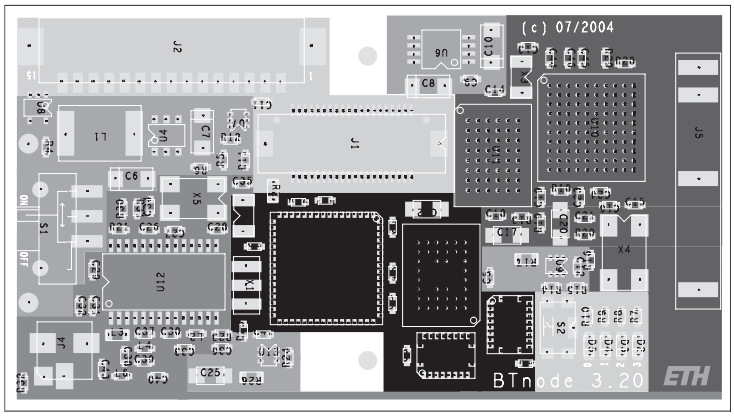

Figure 3-15
An area breakdown of the 1890 mm^2 of the BTnode rev3 into its subsystems results in system core 15.6%, Bluetooth 22.4%, low-power radio 14.1%, power management 16.4%, external IO 17.6% and other areas 13.5%.

Table 3-8: BTnode family – physical setup and commercial figures

	BTnode rev1	**BTnode rev2**	**BTnode rev3**
Size	60x40 mm	60x40 mm	58.15x32.5 mm
Components	45	50	109
Year	2001	2002	2004
Availability	prototypes only	prototype series	yes
Unit Cost	USD 170	USD 190	USD 215
Dev Kit Cost	USD 30-100	USD 30-300	USD 300
Tool Cost	–	–	–
Open Platform	yes	yes	yes

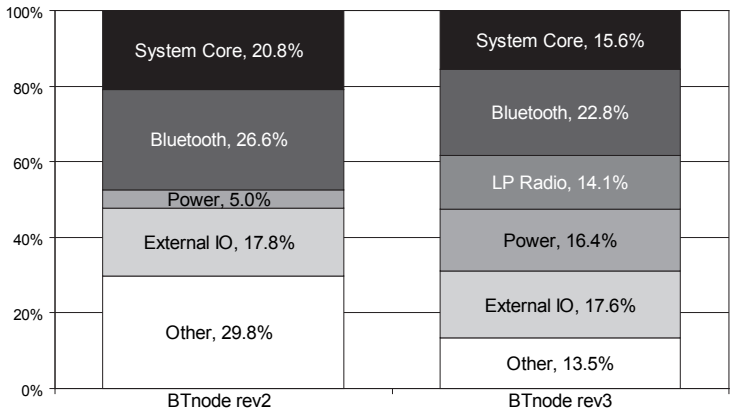

Figure 3-16
The BTnode area breakdown as measured on the circuit boards for the BTnode rev2 and BTnode rev3 compared. The percentages shown show that with increasing integration, external IO, power management and other areas dominate the area cost over the actual core components.

3.4.2.5 Programming

An embedded platform for deployment either in large numbers or in remote embedments in the physical environment has requirements on the programmability that exceed those of a traditional embedded platform. The program memory of the microcontroller in the BTnode system core can be programmed using a traditional ISP or a JTAG in-circuit emulator. With the help of a simple command line tool or a more sophisticated integrated development environment (IDE) the processor is configured by setting the microcontroller fuse bits, memories can be erased, read, written and verified. Furthermore the joint test action group or IEEE 1149.1 boundary-scan (JTAG) in-circuit emulator allows to set breakpoints for low-level debugging at runtime.

For quick desktop reprogramming and code dissemination over a wireless network the bootloader feature of the ATmega128l microcontroller can be used. Here, a bootloader remains resident in a reserved section of the program memory (max. 4096 kilobytes on the ATmega128l). When executed on a system reboot, it detects a program upload via the BTnodes serial port or a resident firmware in the storage memory (SRAM). The bootloader finally transfers this executable to the program memory (Flash) and starts it. For network reprogramming, the application currently running on the system needs to receive the new executable, save it to storage memory, and then reboot the system in bootloader mode.

The ZV4002 Bluetooth system carries it's own firmware that needs to be initially programmed during factory testing. This is done only once on the current BTnode, but an exchange of the firmware and development of embedded applications on the ZV4002 itself would be possible using the appropriate solution developer kit (SDK) from Zeevo (see [KAH+04] and [NKA+05] for a similar approach on the Intel Imote).

3.4.3 Lightweight Operating System Support

The hardware resources available on such a miniaturized embedded device are quite reduced when compared to other mobile systems like modern cell phones or PDA running full-blown Windows derivatives on multi-scalar processors. The performance of the BTnode microcontroller core and OS can be somewhat compared to the first PC in the

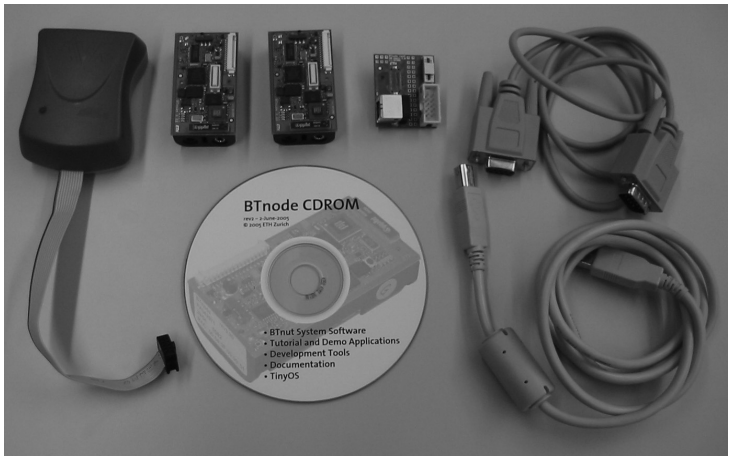

Figure 3-17
A simple BTnode developer kit to perform first steps consists of a BTnode rev3, a usbprog USB programming adapter, an Atmel ATAVRISP programmer, a serial and a USB cable, a 15-Pin Molex breakout cable for the debug connector J2 and the software, documentation and tools contained on the BTnode CDROM.

80's, without features such as dynamic memory management, dynamic program loading and execution. Thus, extreme care needs to be taken to make use of the scarce resources efficiently, predominantly a domain of classical embedded systems.

The first-generation BTnode system software (see Figure 3-18) is a lightweight OS/application framework written in C and assembly language using a standard libC and the gcc compiler suite. The drivers, which are available for different hardware subsystems, provide convenient application programming interfaces (API) for application developers. The system supports an event-based programming model and provides non-preemptive scheduling of event-handlers.Within this framework, applications are typically partitioned into a Bluetooth link-layer that is communicating with the Bluetooth front end through commands and events, keeping track of point-to-point connections with individual link-state machines, a command-line terminal for control and debugging and an application core. The terminal interface, similar to a command-line shell application or telnet allows convenient interactive control and monitoring of applications on the embedded devices using a well-acquainted and verbose user interface. The whole application is defined at compile time and then programmed into the Flash memory of the microcontroller using either an ISP, JTAG or a resident bootloader.

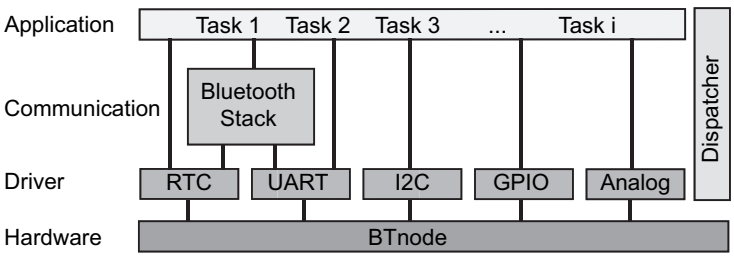

Figure 3-18
First-generation BTnode system software – A lightweight OS framework for wireless sensor network applications. A dispatcher is responsible for registering and executing event-handlers in a coordinated fashion.

3.4.3.1 Device Drivers

The drivers are designed with fixed buffer lengths that can be adjusted at compile time to meet the stringent memory requirements. Available drivers include memory, real-time clock, UART, I2C, LED, power modes, and ADC. The driver for the Bluetooth radio provides a subset of the networking functionality according to the Bluetooth specification. Bluetooth link management is performed on Bluetooth's logical link and adaptation protocol (L2CAP) layer. The RFCOMM protocol, a serial port emulation, provides connectivity to computer terminals and consumer devices, such as cameras and mobile phones.

3.4.3.2 Programming Model

The system is geared towards the processing of (typically externally triggered) events, such as sensor readings, the reception of data packets on the radio interface or user interface interaction. To this end, BTnode applications follow an event-triggered programming model, common to embedded systems and graphical user interface (GUI) programming toolkits, where the system schedules application tasks on the occurrence of events. Internally, those tasks (or event-handlers) are implemented as C functions.

In the BTnode system, events model state changes. A set of predefined event types is used by the drivers to indicate critical system conditions, such as expired timeouts or data being ready for reading on an IO device. Applications can define their own event types, which are represented by 8-bit integer values. Individual events can carry type-specific parameters for evaluation in the application task.

A central component of the system is the dispatcher, where applications have to register event/event–handler–function pairs. After the dispatcher has been started, it also accepts individual events from drivers and application tasks. Low-level drivers (interrupts) or other software components can generate an event to notify other system components to take some action. In this context, events are not processed immediately but are stored in an event queue in the dispatcher.

The dispatcher mainly consists of a first-in first-out (FIFO) queue that stores the events that occurred and calls the corresponding event handlers to process them. Of course, other, more complicated queuing disciplines are possible too. While the event queue is not empty, the dispatcher starts the corresponding handler (i.e. , the previously registered function) for the next event in the queue, passing the events individual parameters. Events are processed in the order they are received by the dispatcher. Only one task (event handler) can be active at a time and every event handler is always completely executed before the next is scheduled. So every event handler depends on the previous event handler to terminate in time. However, long tasks can be broken into smaller pieces and can be triggered subsequently by the event mechanism (e.g. by application-specific events).

3.4.3.3 Process Model

Like most embedded systems, the BTnode does not support dynamic loading and execution of applications at runtime. Only a single application is present on the system at

```
 1: #include <btnode.h>
    #define THRESH_EV (MAX_SYS_EVENT+1)
 3: static u16 conn_id = 0;

 5: int main( int argc, char* argv[])
    {
      btn_system_init(argc, argv, /* ... */ );
      /* accept conn */
      btn_bt_psm_add(101);

11:   btn_disp_ev_reg(BT_CONN_EV, conn_cb, 0);
12:   btn_disp_run();
      return 0; /* not reached */
    }

16: void conn_cb(call_t call, cb_t cb)
    {
      conn_id = (u16)(call_data & 0xFFFF);
      bt_sens_start(BTN_SENS);
20:   bt_sens_set_thresh(BTN_SENS, 30, THRESH_EV);
21:   btn_disp_ev_reg(THRESH_EV, sens_cb, 0);
    }

24: void sens_cb(call_t call, cb_t cb)
    {
      u8 error = 0;
      u8 buf = (call & 0xFFFF);
      btn_bt_send(conn_id, buf, sizeof(buffer));
    }
```

Figure 3-19
A typical BTnode program, which is waiting for an incoming Bluetooth connection. Once the connection is established, it repeatedly transmits sensor data exceeding a given threshold. During initialization, the program registers a handler for connection events (line 11) and then passes control to the dispatcher (line 12), which enters sleep mode until events occur that need processing. Once the connection is established, the corresponding handler function `conn_cb` *is called by the dispatcher (line 16). The program initializes the sensors with a threshold value (line 20) and registers the event handler* `sens_cb` *(line 21). On the occurrence of the sensor event, the associated data is sent over the connection (line 24 ff).*

a time. At compile time, applications are linked to the system software, which comes as a library. The resulting executable is then uploaded to the BTnode's Flash memory, effectively overwriting any previous application code. After uploading, the new application starts immediately. However, the BTnode system can also be reprogrammed through the network (see section 3.4.2.5).

3.4.3.4 Linux-to-AVR Embedded Emulation

In contrast to the design flow proposed for the Berkeley Motes using large simulations as a means for stepwise refinement and collaborative debugging [LLWC03] we propose a method based on virtualization and emulation of the embedded BTnode platform on a Linux system [BKM+04] prior to deployment on the embedded target device. The whole system software is designed for portability and is available for different emulation environments (x86 and iPAQ Linux, Cygwin, and Mac OS X) apart from the embedded platform itself. This cross-platform development approach uses Bluetooth devices attached to a PC to run BTnode applications natively without the need for tedious downloading to the embedded target itself and allowing for extensive debugging an monitoring due to the capabilities of the PC platform. Native compilation and execution on Linux is achieved using adapted drivers to match the host system and a virtualization of the core OS functions in conjunction with a serial Bluetooth device on a PC or even an iPAQ device.

Emulation simplifies application building and speeds up debugging since developers can rely on the sophisticated debugging tools available on desktop systems. Also, the time for

uploading the sensor-node application to the embedded target can be saved. Furthermore, different platforms, such as an iPAQ running Linux, can be used as cluster heads, reusing much of the software written for the embedded nodes and making use of the resources of the larger host platform for interfacing, extended computation or storage.

Using this "embedded Linux emulation" we can (i) make use of the unlimited resources of a PC host platform as cluster head, e.g. for computationally intensive tasks, (ii) to bridging networks, (iii) for comfortable application debugging and (iv) using the actual Bluetooth devices embedded into a real environment instead of a simplified model as is often the case in simulations [LLWC03]. The emulation has been used to perform the computational intensive operations of the Hop-TERRAIN algorithm (see section 2.3.3) in an early implementation based on XHOP (see section 4.2). Here, native x86 Linux binary libraries were linked with BTnode application code and proved the versatility of the approach.

3.4.4 Towards a Second Generation Programming Model

The first-generation, non-preemptive, event-triggered BTnode system software has drawbacks that are unacceptable in the context of networked embedded systems. In fact all event-triggered systems have these drawbacks [KR05] – not only the BTnode systems software. It strongly depends on the applications whether these drawbacks are just minor annoyances or become sources of instability and malfunction.

The key problem stems from applications that are not as simple and clearly structured as the idealized "sample and broadcast" application, often referred to as the standard WSN application. These simple applications can easily be serialized and optimized for energy efficiency using duty-cycling and state-based dynamic power management schemes. In the case of multiple, heterogeneous and asynchronous events, such as often encountered in interactive applications or in the case of the BTnode, on the Bluetooth HCI interface with complex and often interleaved control structures, a single FIFO event-handler loop such as present in the BTnode system software dispatcher in section 3.4.3 is inappropriate. Although the communication between processes is simple and based on global state variables, multiple interrupts of the same kind can lead to problems here. Too many events generated by a single source as well as long processes that prevent others from running can easily lead to buffer overflows. Extensive routines for preserving and checking the context prior to accessing shared resources are the norm and buffer overflows can occur frequently when the system load increases. In practice this results in complex, bulky and often reentrant code, that is hard to maintain and unable to support the functional performance and quality required.

Similar programming frameworks such as Contiki [DGV04], Maté [LC02] as well as TinyOS-1.x and nesC [GLvB+03] follow a comparable non-preemptive, event-triggered model and thus share most of these drawbacks with the first-generation BTnode system software. Switching to one of these systems would not remedy the problems encountered when larger, control-flow dominated applications were to be developed and deployed. Therefore, an OS option that is not following the non-preemptive, event-triggered model,

equally well-supported as TinyOS and preferably written and maintained in the C programming language, had to be considered.

This led to the choice of a multi-threaded model with each thread executing its own sequential control flow in its own state (register and memory context). Since only one thread can be executed at a time on a single CPU, a scheduler is responsible for the selection of the next thread to execute and keeps track of all threads and events during run-time. In the non-preemptive case, each thread has to decide when to allow a context switch somewhat relaxing (but not ultimately solving) the problem of too long processes as mentioned earlier. It is usually the case that priority based schedulers are being used with this process model.

3.4.4.1 BTnut System Software Integration

Nut/OS is an intentionally simple real-time operating system (RTOS) for the Atmel AVR family of microcontrollers, which provides a minimum of network oriented system services with a large and active user base. Existing application domains of Nut/OS are mostly in networking applications but user interfaces, such as terminals and displays, are also in use. It's features include:

- Non-preemptive, cooperative multi-threading
- Events
- Priorities for threads
- Periodic and one-shot timers
- Dynamic heap memory allocation
- Interrupt driven streaming I/O

The second generation BTnut system software builds upon this RTOS framework by providing BTnode specific drivers, communication protocol stacks for the Bluetooth and low-power radio radios and demo applications. A separate build environment automatically configures and integrates the Nut/OS core into this BTnut extension. The basic drivers and the application structure is intentionally kept similar to the first-generation software, although somewhat simplified in their handling through the use of the embedded multi-threaded OS.

A novel programming model for programming wireless sensor networks using attributed state machines has been proposed by Kasten [KR05] but is currently still in research phase and not ready yet for real-world applications.

3.5 BTnode Platform Success

The BTnode platform and especially the software systems, support mechanisms and tools discussed in this thesis has been jointly developed by the Computer Engineering and Networks Lab and the research group for Distributed Systems at ETH Zurich. The details of

the system software infrastructure for the BTnode platform are outside the scope of this thesis and documented in detail by Siegemund [Sie04b] and Kasten [Kas05].

The BTnode platform is not solely used in the research presented here, but truly a versatile and reliable ad hoc networking platform for fast-prototyping of applications and experiments in research and development environments. The BTnode rev2 has been successfully deployed with over 200 units. Many researchers across Europe but also in the US have been using these for their implementation work and experimentation. The BTnode rev3 has been commercialized with an industrial partner, the former ETH Zurich spin-off Art of Technology. To date, the BTnode hardware has been used by 30+ other research groups outside of ETH Zurich. This has led to a substantial gain in experience throughout the research community as well as numerous well-perceived publications both by members of the BTnode project, closely related researchers as well as third parties.

The development of the BTnode platform under an open-source policy has led to several copies of the design, especially mentioned here is a redesign and commercially available hardware replicas of the BTnode rev2 through a US based company (Cobalt Blue by Vitronics).

3.5.1 BTnodes in Education

Apart form being a popular system for student thesis projects, the BTnodes and BTnut have been introduced in a 5-session lab of a graduate course on embedded systems design held at ETH Zurich for the first time in spring 2005 (120 participants) [BBDM05].

Teaching and Demos

- Different seminars, workshops and demos
- Hands-on tutorials to related researchers
- Demo in cooperation with the computer vision and wearable computing labs at ETH Zurichs 150th anniversary
- BTnode related hardware as an example in a Bluetooth textbook [MT02]
- About 30-40 successfully completed student projects (thesis/internships)
- Bi-annual undergraduate lab using Lego Mindstorms and BTnodes [ER02]
- Graduate lab in embedded systems [BBDM05]

3.5.2 BTnodes in Research Domains

A selection of the most prominent work that has been based (in part) on the BTnodes and related publications:

Wearable Computing Applications and Case Studies

- Physical activity detection network [JLT03],
- Wearable unit with reconfigurable modules [PEW+02, PEW+03]

- A systematic approach to the design of distributed wearable systems [ABD+04]

Ubiquitous Computing Applications and Demonstrators

- Proactive furniture assembly [AMS02]
- Better avalanche rescue through sensors [MS02]
- Smart spaces using RFID tags [SR02],
- User interaction scenarios with cell phones, active and passive tags [SF03]
- Smart product monitoring using mobile phones and short text messages [Sie02, SR03]
- Context-aware communication platform for smart objects [Sie04a]
- Cooperating smart everyday objects [Sie04b]
- Handhelds and cooperating smart everyday objects [SK04b, SFV04]
- Tuplespace-based collaboration of Bluetooth-enabled devices in smart environments [SK04a]
- Proximity as a security property in a mobile enterprise application context [DNHKK04]
- Building intelligent environments with Smart-Its [HGK+04]
- Physical prototyping with Smart-Its [GKSB04]
- Innovative application development for ubiquitous and wearable computing [Mic04]
- Improving interaction with context-aware systems [Ant04]

Wireless Sensor Networks Research

- Power management, throughput measurements based on BTnode rev2 and TinyOS [LDB03]
- Prototypes and evaluations for the Hogthrob project [BBDL03, Leo04]
- Connection oriented sensor-networks [Dyd04]
- BitMAC: A deterministic, collision-free, and robust MAC protocol for sensor networks [RR05]
- Beyond event handlers: programming wireless sensors with attributed state machines [KR05]
- A distributed platform for sensor networks [BKR03a, BKR03b]
- Prototyping wireless sensor networks with BTnodes [BKM+04]
- Next-generation prototyping of sensor networks [BDH+04]
- A state based programming framework for wireless sensor networks [Kas05]
- TinyOS on BTnodes [BD05]

Time Synchronization and Local Positioning

- Tracking real-world phenomena with smart dust [Röm04b] using the lighthouse location system [Röm03]
- Determination of time and location in large-scale dynamic networks of tiny sensors [Röm04a]
- Time synchronization using BTnodes [MW05]
- Time synchronization and calibration in wireless sensor networks [RBM05]
- Time synchronization and localization in sensor network [Röm05]
- Topology and position estimation in Bluetooth ad hoc networks [Fre03]

Bluetooth Performance and Algorithms

- Early BTnode performance assessments [KL01]
- Power management of BTnodes [MFT05]
- Power consumption of bluetooth scatternets [NBD05b]
- Robust topology formation using BTnodes [Beu05]
- A practical topology control algorithm for ad-hoc networks [WZ04]
- Topology control for deployment support networks [BDMT05, DBM05]

4

Robust Multihop Networking using BTnodes

Research of the past years has brought about many new developments and proposals for wireless sensor networks, mostly in areas of algorithms and theory, backed up by extensive simulation work. In order to fully understand the complexity of these issues from a system perspective we will highlight the idiosyncracies of multihop networking implemented on real devices. In this chapter we example of how to construct connected ad hoc network topologies using Bluetooth. The goal of this work is twofold: We want to (i) form large, connected networks consisting of many devices, supporting transparent multihop transport and (ii) to understand the limits and benefits of Bluetooth technology in the context of ad hoc networking and the relevant implementation cross-layer issues.

A prerequisites for multihop networking on an ad hoc network is a connected network topology spanning all nodes in a given area of interest. Different approaches for multihop routing are known [PH00, ICP+99] that can be used to to meet different performance requirements. Popular routing protocols like dynamic destination-dequenced distance-vector routing (DSDV) [PB94], ad hoc on demand distance vector (AODV) [Per01], dynamic source routing (DSR) [JM96], location aided [KV98, YBV00], dominating set based [Wu02] or geometric routing [KWZ03] all rely on an underlying communication network infrastructure. In the case of a Bluetooth network, this underlying infrastructure is a connection oriented medium, as opposed to a broadcast oriented or connectionless medium. This means that within a given area where radio communication can be established, nodes do not communicate over a joint channel but that between any two node pairs a separate channel exists. The challenge here is not so much in the multiuser separation, collision detection and avoidance but in distributed synchronization and scheduling of the distinct channels as to deliver maximum service quality to all nodes.

A first task to be solved in multihop networking using Bluetooth is the construction and maintenance of a connected network topology. This basic backbone connectivity serves the purpose of a general deployment infrastructure on which data can be transported from node to node and which services such as a local positioning service outlined in section 2.3

can be operated. Topology construction should be simple, robust and local as to minimize the burden inflicted on neighboring nodes. On every node, such a topology control service encompasses the discovery of remote nodes, connection management (opening, closing and supervising links), as well as the construction and maintenance of a connected network topology based on the node, link and application status. Then, data-forwarding services, routing schemes and other services can make use of the network topology to create meaningful and worthwhile applications.

We have developed three different variants of such multihop network topologies based on the BTnode platform described earlier (see chapter 3), each accounting for different prerequisites and capabilities of the underlying hardware. Additionally, it was necessary to also implement an adequate data transport scheme to accompany each topology variant, enabling debugging and monitoring of the node functions. The first example (see section 4.2) is based on Bluetooth devices not capable of forming scatternets but only point-to-point piconets (Ericsson ROK101008, one master-slave connection) and uses a time-multiplexed approach to achieve multihop connectivity. The TreeNet example (see section 4.3) implements a self-healing, distributed Bluetooth scatternet formation on the BTnode rev2 platform using an improved Bluetooth module (Ericsson ROK101007) capable of rudimentary scatternet formation as described in sections 3.4 and 4.3.1.1. The third examples makes use of the BTnode rev3 with a modern Bluetooth radio system (Zeevo ZV4002) capable of forming advanced scatternets by means of parallel management of multiple connections and a much higher degree of freedom than it's predecessors to form large network topologies to be used for a deployment-support network (see section 4.4).

After a brief introduction to some basic principles and the fundamentals Bluetooth wireless communication in section 4.1, we will discuss our example implementations including devices used, prerequisites, limitations, algorithmic ideas and implementation results in the sections 4.2, 4.3 and 4.4. Much of the implementation, testing and experimental work presented in this chapter would not have been possible alone and was performed in collaboration with Matthias Dyer, Lennart Meier and Matthias Ringwald.

4.1 A Connection Oriented Medium – Bluetooth

Bluetooth, or IEEE 802.15.1 is a short-range wireless communication system conceived as wireless cable replacement by major telecommunication equipment manufacturers such as Nokia, Motorola, Ericsson, IBM and others [Haa01, Haa00, HM00]. It's main design goal is to seamlessly interconnect portable and/or fixed electronic equipment, e.g. personal computers, handhelds, mobile phones, audio equipment, cameras and other accessories. The Bluetooth standard uses a strongly layered approach, defining the radio system and data transmission on the lower layers as well as interfaces and support for the application on higher layers through Bluetooth profiles. Many features of the core specification are optional, allowing product differentiation. Today over 30 of such profiles exist for various applications, e.g. remote control, call control, object push, hands-free, local positioning personal area networking, service discovery and others.

The Bluetooth physical layer operates in the unlicensed ISM band at 2.4 GHz and uses a

frequency hopping transceiver with 1 Megasymbol per sec. Typically, a group of devices share a physical channel in a master-slave configuration called a piconet. The master provides the synchronization reference and controls the time-shared medium access in slots of 625 µsec using a frequency hopping pattern. Within a physical channel, a physical link is formed between any two devices that transmit packets in either direction between them. In a piconet physical channel there are restrictions on which devices may form a physical link. There is a physical link between each slave and the master. Physical links are not formed directly between the slaves in a piconet. The physical link is used as a transport for one or more logical links that support unicast synchronous, asynchronous and isochronous traffic, and broadcast traffic. Traffic on logical links is multiplexed onto the physical link by occupying slots assigned by a scheduling function in the resource manager. Above the baseband layer the L2CAP layer provides a channel-based abstraction to applications and services. It carries out segmentation and reassembly of application data and multiplexing and de-multiplexing of multiple channels over a shared logical link.

The system architecture is partitioned into a controller and host partition (see Figure 4-1) that communicate through a host controller interface (HCI). This allows the independent integration of the controller, i.e. the Bluetooth device or module and the host, i.e. a personal computers CPU. As the Bluetooth controller is typically assumed to be strongly resource limited in comparison to the host, the L2CAP layer is expected to carry out certain management functions, e.g. message sizes, buffer management, etc.

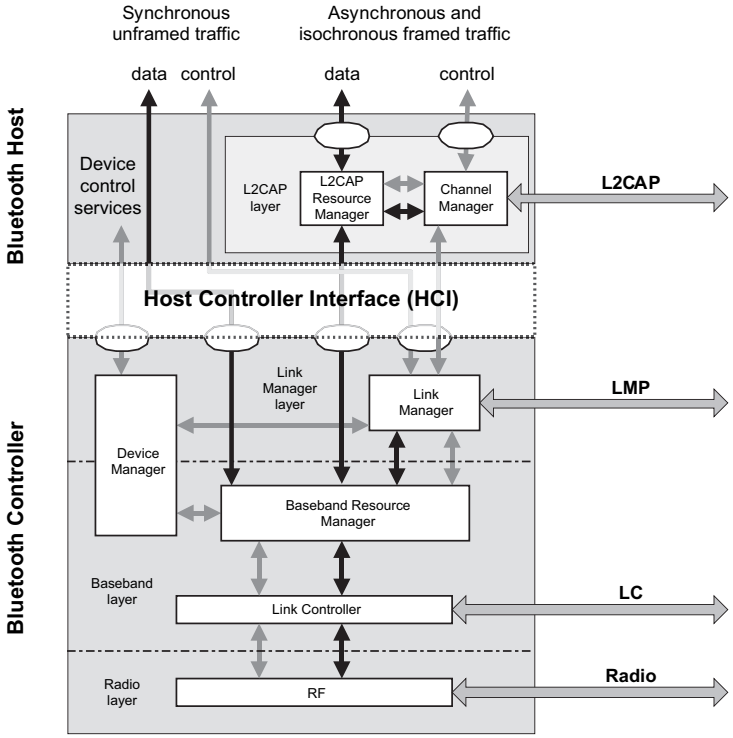

Figure 4-1
The Bluetooth core system architecture is shown except for the service discovery protocol (SDP) that is not shown for clarity. Systems are partitioned into a controller and host section that communicate via the host controller interface (HCI). Different protocols are defined on the various layers, e.g. RF and baseband (Radio), link controller (LC), link management protocol (LMP) and logical link and adaptation protocol (L2CAP). It supports both synchronous and asynchronous data traffic.

The first Bluetooth draft documents date back to 1998 and have subsequently been revised and improved.

4.1.1 Embedded Bluetooth

The generic HCI eases integration of Bluetooth into new products. These need only be capable of running a Bluetooth host protocol stack and the profiles according to the application requirements. Bluetooth devices and modules are increasingly being made available which come with an embedded stack and a standard universal asynchronous receiver transmitter (UART) port. The UART protocol can be as simple as the industry standard AT protocol, which allows the device to be configured to cable replacement mode. This means it now only takes a matter of hours (instead of weeks) to enable legacy wireless products that communicate via UART port.

4.1.2 Bluetooth Pros and Cons

++ Price (single device and design-in)

++ Layered architecture, modularity

++ High layer abstractions, HCI

++ Built in encryption and security features

++ Continuous development, maturing standard due to improvements in drafts 1.0 and 2.0

−− Power consumption, especially of early SoC

−− Limited local connectivity (max. 8 nodes per piconet)

−− Complex to handle, mostly because of connection orientated nature, quality of service and security issues

−− Initiating a connection takes a long time

4.1.3 Bluetooth Networking – Operational Prerequisites

Networking in Bluetooth is organized in master-slave configurations of up to seven active slaves that can be connected to one master at a time (piconet). Prior to opening a connection to a specific node, the initiating node has to perform an inquiry to detect other nodes in the vicinity. Multiple piconets can be interconnected by nodes taking on dual roles of slave–slave or master–slave forming a scatternet (see Figures 4-6 and 4-13). While the interconnection of nodes in these different configurations is part of the Bluetooth standard, the formation and control of multihop topologies is not governed by the standard. Also, the data transport is only defined on each single hop (from master to slave or vice versa) and not over multiple hops, e.g. from slave to slave over an intermediate master in a standard piconet or even across a scatternet formed by multiple piconets. This means that a functional layer running on the host controller must take care of general connection management and all multihop packet forwarding (see section 3.4.3).

At the lowest architectural layer, four types of physical channels are defined. Two of these physical channels (the basic piconet channel and adapted piconet channel) are used for communication between connected devices and are associated with a specific piconet. The remaining physical channels are used for discovering Bluetooth devices (the inquiry scan channel) and for connecting Bluetooth devices (the page scan channel). In order to reduce the likelihood of two independent Bluetooth transceivers tuned to the same RF frequency at the same time a mandatory channel access code is being prefixed to every packet on the physical channels. A Bluetooth device can only use one of these physical channels at any given time. In order to support multiple concurrent operations, the device uses time-division multiplexing between the channels. In this way a Bluetooth device can appear to operate simultaneously in several piconets, while still remaining discoverable and connectable. Whenever a Bluetooth device is synchronized to the timing, frequency and access code of a physical channel it is said to be connected to this channel. The Bluetooth specification assumes that a device is only capable of connecting to one physical channel at any time. Advanced devices may be capable of connecting simultaneously to more than one physical channel, but the specification does not assume that this is possible. Situated above the physical channels are physical links (active and parked links) as well as logical links and L2CAP channels. The physical links can transport synchonous connection-oriented (SCO) (bi-directional, symmetric, audio/video) and asynchronous connection-oriented (ACL) data (reliable or time-bounded, bi-directional, point-to-point) that can be parametrized with respect to up- and downlink data rates, error correction and quality of service (QoS).

Bluetooth devices use the asymmetric inquiry procedure to discover nearby devices by sending an inquiry request on the inquiry scan channel. Devices that are available to be found are known as discoverable devices and listen and respond to these inquiry requests. The procedure for forming connections is asymmetrical and requires that one Bluetooth device carries out the page (connection) procedure while the other Bluetooth device is connectable (page scanning). The connectable device uses a special physical channel to listen for connection request packets from the paging (connecting) device. This physical channel has attributes that are specific to the connectable device, hence only a paging device with knowledge of the connectable device is able to communicate on this channel. After a successful connection procedure, the devices are physically connected to each other within a piconet. While in a connection, different modes are possible: hold (reduced traffic), sniff (reduced duty cycle) and park (suspended traffic). A role switch allows to swap the roles (master and slave) of two devices connected in a piconet. The Bluetooth device is responsible to balance the different procedures, physical channel access and distribute time-shared resources to meet QoS requirements and the processing needs of ongoing logical transports.

The host controller interface (see Figure 4-1) provides abstraction and standardized access to the lower levels of the protocol stack residing on the Bluetooth controller. HCI events are used for notifying the host when something occurs. HCI commands provide the host with the ability to control baseband, link management layer, policies and status. HCI commands may take different amounts of time to be completed. Therefore, the results of commands will be reported back to the host in the form of an event. The host will receive isochronous notifications of HCI events independent of which host controller transport layer is used (currently, UART and

universal serial bus (USB) are available transport layers). When the host discovers that an event has occurred it will then parse the received event packet to determine which event occurred. For example, for most HCI commands the controller will generate the command complete event when a command is completed. This event contains the return parameters for the completed HCI command. For enabling the host to detect errors on the HCI-transport layer, there needs to be a timeout between the transmission of the host's command and the reception of the controller's response (e.g. a command complete or command status event). This amount of time is dependent on the number of commands unprocessed in the command queue and the HCI-transport layer used.

The host is responsible for opening and closing connections by issuing the required HCI commands. It is required to maintain a link state machine for every physical link and for the multiplexing of all logical links and the data transfer. For this, different ACL and synchronous SCO data packets are assembled at the sending side's host and transferred over the HCI to the Bluetooth device. The data is received and output to the receiving side's host in the same order as it has been sent.

For an in-depth discussion of Bluetooth properties in the context of ad-hoc networking either see the reference listed in section 4.1.4 or the recent thesis by Dydensborg [Dyd04].

4.1.4 Bluetooth Network Topologies – Related Work

The Bluetooth standard only describes the minimum requirements of the interconnection of piconets to form scatternets using master-slave and slave-slave bridges but no suitable algorithmic control of the network topology. This leaves room for individual ideas and researchers have embraced the topic and generated numerous proposals.

There are basic studies on the suitability of Bluetooth as a networking technology by Johansson [JKKG01] as well as on the comparison and interferences of Bluetooth and 802.11b wireless networks by Ferro [FP05a] and Chandrashekar [CCM+01].

The solutions proposed for the automatic formation of Bluetooth scatternets can be classified according to the preconditions of the algorithms and the properties of the resulting scatternet. One such precondition is that all nodes must be in each other's transmission range (single-hop) [LMS03]. The BlueMesh [PBC02] algorithm is one solution for the more general multihop case, and guarantees that a connected mesh topology is achieved. Unfortunately, the authors of BlueMesh assume that the topology of the network does not change. This algorithm can therefore not deal with link failures and leaving and joining nodes.

Many other proposed solutions refer to the problem of setting up static scatternets [PBC03, ZBC01] where many different algorithms for mesh structures either conforming to the standard [Bas02, BP02], or non-conforming with mesh [PBC02], star [PBC03], ring [FC02] or tree [ZBC01] topologies have been proposed. The performance of Bluerings has been studies by Lin [LTC03]. A recent distributed algorithm with bounds on complexity proposed by Law [LMS03] has been validated by simulations as well as through a comparative study by Basagni [BBMP04].

Following algorithmic work on the connectivity of Bluetooth scatternets [GRSV03], Guerin has studied the performance and complexity of such algorithms and comes to the conclusion that forming at least one connected topology is NP-hard due to degree constraints. However, his investigation is based on assumptions that cannot be justified on real devices, e.g. seven simultaneous master and seven slave roles [VGSR05].

Similarly to this study, the properties and performance of the most algorithmic work is determined by simulation with often unrealistic choice of parameters for these evaluations. For instance, the performance evaluation in [BBMP04] used a variable time for the inquiry operation, randomly selected in the interval from 0.01 sec to 0.5 sec. This is not realistic compared to real devices where this duration is specified to be in the range of 1.28 sec to 61.44 sec by the Bluetooth standard.

To the best of our knowledge, implementation reports of a large-scale multihop scatternet-formation algorithm are missing so far. Even theoretical analysis has just begun to address the problem of maintaining a connected scatternet for multihop topologies in a dynamic environment. Recent evaluations of scatternet-formation algorithms [BBP03, BBMP04] have also referred to the lack of such algorithms.

4.2 XHOP – Multihop Bluetooth Data Transport

Initial experiments [Fre03] based on the BTnode rev1 and BTnode rev2 equipped with the same Bluetooth devices (Ericsson ROK101008) allowed to form point-to-point connections only. These severe device restrictions require to use a time-multiplexed scheme for multihop networking. Data is sent on one point-to-point link at a time and cached on intermediate nodes. In a large, distributed context with multiple packets being transported, this results in a dumbbell-like network topology (when viewed as a momentary snapshot), with connections being created on demand only and being torn down after data transmission had finished (see Figure 4-2).

In an idle state, there are no open connections in the network. Connections will only be opened if there is data that has to be sent from one to another node. After a connection has been established, the data will be sent. If there is no pending data for an open connection it is closed again immediately. Obviously, this approach does not allow fast multihop data transmission as paging is a lengthy process (see section 4.3.2). On the other hand, it allows us to open all connections that are possible among devices, which are in range of each other.

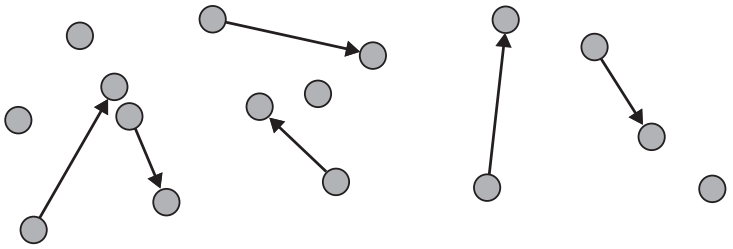

Figure 4-2
XHOP multihop data transport – Dumbbell-like Bluetooth connections allow for a time-multiplexed multihop routing across piconet borders. At any given time, only single point-to-point links are available.

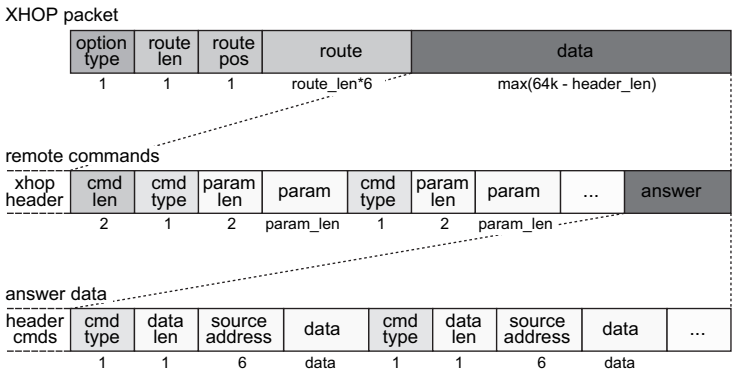

Figure 4-3
An XHOP packet consist of a predefined route of Bluetooth addresses and payload data up to the maximum length of a Bluetooth ACL data packet (max. 64 kbytes). Using this protocol, remote commands can be transported and executed across multiple hops generating answer data that is returned to the origin of a remote command.

Based on this approach, the XHOP Bluetooth multihop source routing protocol implements a variant of the popular dynamic source routing (DSR) developed at Carnegie Mellon University [JM96]. XHOP provides a mean for multihop data transport and a script like command language for remote command execution (see Figure 4-3). DSR is an on-demand routing protocol, which means that routes are not updated continuously. When a node requires sending data, it first needs to initiate a route discovery process before it can use the retrieved route to send the data with XHOP. To allow remote configuration and remote data queries over several hops in a BTnode network, XHOP was implemented using a connection manager consisting of an inquiry scheduler and a connection manager/packet forwarder.

4.2.1 XHOP Connection Manager

To facilitate data transmission between devices a connection manager has been implemented that sits immediately above the L2CAP layer and is responsible for all data transmission in the network. The connection manager consists of two parts. The first part is the inquiry scheduler that periodically performs inquiries and maintains a list of known devices that can be queried by the upper layers. The other part is the actual connection manager that accepts data packets from upper layers and is responsible for opening and closing connections and delivering the packets.

4.2.1.1 Basic Inquiry Scheduler

The inquiry scheduler performs inquiries in a periodic interval and manages a list of devices discovered. The list contains the known device's Bluetooth address, their clock offset, a time stamp of the last time the node was reported and a status byte. The routine also checks the remote's class of device (CoD) used for further identification, i.e. to decide whether the remote device is a BTnode or not. By using an unique CoD this provides a simple but effective means for identification of devices that are part of the experiment and such that are not. To account for dynamic network environments, devices are removed from the list if not found for a given timeout. The inquiry list can be queried from remote to perform a topology discovery in the network.

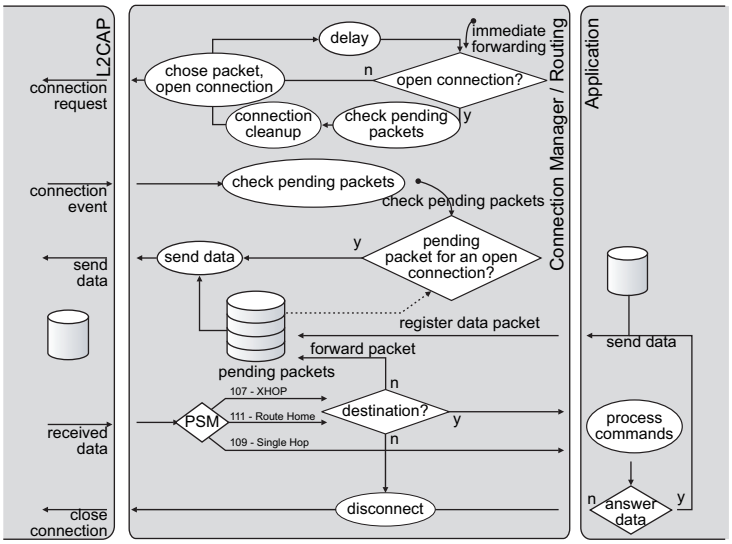

Figure 4-4
The XHOP connection manager integrates both connection management and packet forwarding on top of the Bluetooth L2CAP layer. A reduced variant of dynamic source routing supports multihop data transport. The core functionality is provided by an event-driven state machine. Packets are stored in a pending packet buffer and iteratively handled by the core state-machine. For enhanced readability, the basic inquiry scheduler is not shown here.

4.2.1.2 XHOP Connection Management and Packet Forwarding

The basic scheme for sending data in the dumbbell-like network structure works the following way. The application registers a data packet that has to be sent along a given route. The application tags the packet with a timeout constant and a maximal number of retries in case of a forwarding failure. The data is copied to buffers inside the connection manager and the function returns immediately. The connection manager repeatedly checks for open connections and for pending packets. If there is an open connection to a remote device and there is data to send to that device, then it will send it immediately. Otherwise, it chooses a packet among the pending ones and tries to open a connection to the destination device. The process chosen to select a pending packet to send next is random. However, adding a priority based scheme here would be an easy and useful extension. The connection manager is also responsible to close a connection if there are no pending packets for the device on the other end of the connection. Incoming packets are evaluated for their final destination. Packets destined for the local node get passed to the application directly, while XHOP packets not at their final destination yet are handled as described above (see Figure 4-4).

An application example of XHOP is given in Figure 5.5. Node A sends an XHOP packet to node E over the route (A, B, C, D, E). It requests node E to perform an inquiry and return the result along the reverse route (E, D, C, B, A).

4.2.2 Experiences

This Bluetooth multihop source routing prototype showed the feasibility of an integrated, scalable, cross-layer application protocol on the BTnode devices. It allows multihop data transport by routing across piconet borders and thus supporting more than eight nodes in a network. Furthermore, it allowed remote topology discovery, remote execution of commands using the command line interface and a script like command language in the payload of XHOP packets.

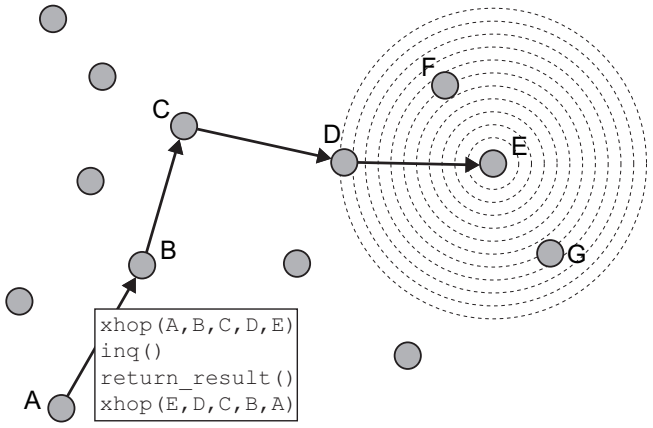

Figure 4-5
Multihop Bluetooth data transport using source routing – The XHOP protocol allows to communicate across piconet borders using a source routing approach. A skript-like command language in the XHOP packet payload allows to execute remote commands like the one shown on the left. Here node A performs a remote topology discovery by sending a set of remote commands along the predefined route (A, B, C, D, E) *to node E and receives the result* (D, F, G) *along the reverse route.*

An obvious caveat of this implementation was the high latency due to the very slow connection performance of about 1–2 sec per hop, depending on the inquiry and paging results, but nevertheless it allowed to gather first experiences, gaining an understanding of the properties and caveats of Bluetooth networking and the related implementation issues.

Prototyping first on Linux and then moving to the embedded platform (see section 3.4.3.4) proved to be a valuable tool to accelerate the design and testing cycle, and as a result fast-prototyping [BKM+04] using the BTnode platform. Another valuable technology developed in the context of the XHOP prototype was network re-programming of the BTnode program memory. This was implemented using the same basic packet format as described earlier and the remote command execution facility. As Bluetooth support error corrected data transmission, it was only necessary to implement an appropriate caching mechanism for the firmware images (up to 128 kbytes), a flash memory programming function and a bootloader. The per-hop performance achieved for 10 kbytes of data was about 0.8 sec for the data transmission and about 0.2 sec for writing to Flash, resulting in about 10 sec to transmit, write and reboot with an 80 kbyte firmware image.

At the time, the BTnode was still an early prototype, hardware components were partially untested often lacking appropriate software and driver support. While the first prototype was still implemented in a single inline application without the support of a system software it was extremely hard to interleave operations, leading to very messy code and as a result, instability of the application. With the BTnode System Software (see section 3.4.3) just in the beginning of development, many features had to be developed from scratch for this application to work, but proved valuable as a basic means of system level support.

Problems encountered with the often unreliable Bluetooth hardware can be largely attributed to the early development status of the devices used, others are properties of this specific wireless technologies where there are many possibilities for application wide failures to occur of which we give a few examples here: (i) As the command for closing a connection immediately disconnects without regarding ongoing data transmissions, care had to be taken when disconnects can

be issued. (ii) The device to which data was sent may fail to disconnect. As it is not the sender's responsibility to close connections it may be left open. (iii) A slave in an open connection is no longer able to respond to connection requests from other masters and is therefore blocked for further data transmissions. Care had to be taken that even on disconnection failures, open, unused connections will still be closed as soon as possible.

4.3 Robust Topology Formation using BTnodes

Throughout this second example of Bluetooth multihop networking, we will be using the BTnode rev2 prototyping platform introduced in the previous chapter as our underlying infrastructure. We will first talk about the Bluetooth prerequisites available on the BTnode rev2 that we can actually use as starting points to develop an appropriate multihop topology maintenance algorithm. This algorithm will be subsequently discussed in our example implementation. The Bluetooth prerequisites contain general definitions made in the Bluetooth standard and specific device capabilities found on the BTnode rev2 devices. Nevertheless, the concepts of this work can be applied in a straightforward manner to other generations of devices with differing capabilities.

4.3.1 Fundamentals of Pico- and Scatternet Formation

For the development of a topology control algorithm based on the BTnodes we will be able to use the states disconnected, master, slave and master-slave as well as the operations

- $inquiry(duration, max\_devices) \rightarrow addressList$
- $connect(address) \rightarrow link$
- $disconnect(link)$
- $roleSwitch(link, role)$
- $sendData(link, data)$

The operation $inquiry()$ is used to scan the environment for other nodes and returns the addresses found in the respective scan. The search operation either runs for a selected $duration$ or until a number of devices $max\_devices$ have been found. A $connect()$ tries to establish a link to a node with a given address, typically found by a previous $inquiry()$. In order to switch the master/slave relation a $roleSwitch()$ can only be performed on single point to point links since it changes the location of the master and thus would break the structure of a piconet with multiple slaves. A $sendData()$ operation finally sends a buffer of data across the link specified. These operations are initiated by sending an HCI command to the Bluetooth device and completion is signalled by an HCI event received by the microcontroller. Although multiple HCI commands can be queued at the Bluetooth controller, the single functions are serialized and are blocking in function until completed, e.g. only one $connect()$ can be executed at a given time, requiring care on the implementation.

4.3.1.1 Bluetooth 1.1 Scatternets on the BTnode rev2

The Bluetooth standard specifies a set of required features, of which devices typically are not required to support all. Likewise the degree of freedom in implementations is reduced considerably from that used in most theoretical studies (see section 4.1.4). In contrast to the Ericsson ROK101008 devices available in our first experiments (see section 4.2) the Ericsson ROK101007 devices available on the BTnode rev2 used here are capable of forming scatternets interconnected via nodes assuming a single master-slave dual role as shown in Figure 4-6.

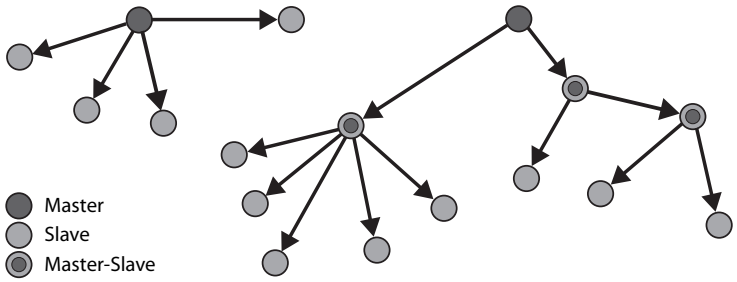

Figure 4-6
Bluetooth 1.1 organized in pico- and scatternets – Single piconets can be interconnected to form tree-shaped scatternets by the means of master-slave bridges. Only nodes assuming a master role (root nodes of a tree) remain visible to other nodes.

In our case, the capability for device discovery and connection control has hardware specific restrictions: A node can only (i) perform or (ii) answer to an $inquiry()$ or $connect()$ while in the states disconnected or master and (iii) is only visible to other nodes while not performing one of the above operations. While assuming slave or master-slave roles the devices are no longer discoverable and connectable. This means that we cannot construct arbitrary mesh topologies (we cannot construct circular structures, but only trees), and that nodes that are already part of a scatternet and not the root of the tree are not discoverable and connectable any more. Tree topologies can thus be only grown from the root and not from intermediate nodes or the leaves.

4.3.2 Non-determinism in Distributed Piconets

Since wireless channels have many variant properties there is generally no guarantee for the success for any over-the-air operation. In general, the wireless medium can be assumed to be an unreliable medium. Again, when applied to Bluetooth technology this means that operations such as $inquiry()$ and $connect()$ can exhibit considerable delays and much worse, have no a priori guarantee for success. This is especially a problem when operating in a distributed and uncoordinated environment with many peers. An $inquiry()$ of a given duration does not always return a complete list of neighbors present and a $connect()$ might not succeed even if before the $inquiry()$ resulted in exactly and exclusively this node. As the operations rely on best effort, no assumptions on the state or even the presence of other nodes can be made prior to a successful search or connection setup. A node that was visible before might not be visible anymore, be busy with an operation or subject to other random sources of interference. In practice, subsequent iterations of an $inquiry()$ result in a different amount and order of nodes found on each iteration [KL01].

In a theoretical worst-case, distributed scenario for the limited-visibility Bluetooth devices used on the BTnode rev2 discussed in section 4.3.1.1 where e.g. all nodes would try to perform

an $inquiry()$ or $connect()$ simultaneously for an equal length duration using the properties described earlier, we would not be able to detect the presence of any node, independent of the node density (see Figure 4-7).

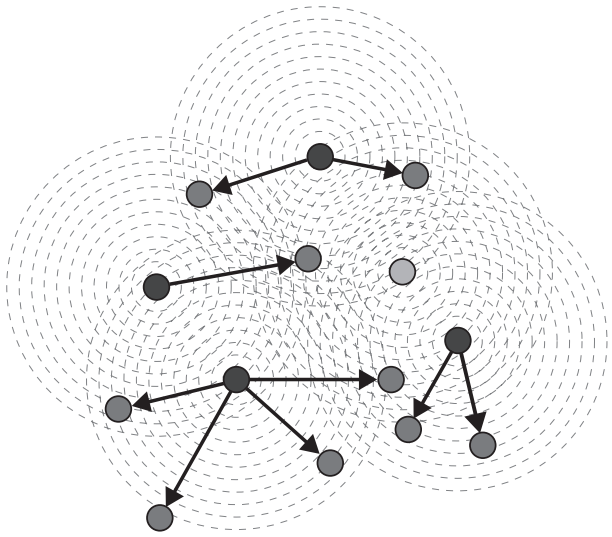

Figure 4-7
Bluetooth 1.1 theoretical worst case scenario – Synchronized $inquiry()$ of all nodes for an equal duration will not give yield node discoveries, since all nodes are busy transmitting $inquiry()$ and cannot respond simultaneously due to hardware limitations of the Ericsson ROK101007 Bluetooth module. An example of a semi-connected network (four independent piconets) and five simultaneous inquiries is shown on the left.

Traditional problems of limited visibility (hidden station problem) are further influenced by asymmetric effects on the higher layers, that are specific to Bluetooth: The inquired node does not notice the presence of the inquiring party, i.e. a response to an inquiry is not signalled to the host by an HCI event. Typically, average rates of success are defined and protocols have to take care of retries, retransmissions and back-off on erroneous behavior. Independent studies by Kasten [KL01], Murphy [MWF02] and Welsh [WMF02] have shown that the average delay for a successful $inquiry()$ and $connect()$ is in the order of multiple seconds. The recent version 1.2 update of the Bluetooth standard actually contains several improvements in this direction.

What we can learn from such insights is, that $inquiry()$ and $connect()$ are highly nondeterministic (both in timing and function) and we will have to find means to cope with all these properties to construct robust and reliable networking services.

4.3.3 TreeNet – Simple Tree Topology Construction

Learning about all these properties we can understand that we cannot construct arbitrary mesh structures in Bluetooth 1.1 scatternets. In fact the devices used allow only to construct trees and no circular structures and that we would have to deal with lot's of non-determinism and failures. So the question in this section is how to construct and maintain arbitrarily large trees in a robust and distributed way using a most simplistic scheme.

A very simple and robust distributed algorithm to be used for connection management based on a search–and–connect scheme that is executed on every node is given in the following:

Starting with all nodes initially disconnected and idle, nodes would perform an $inquiry()$ to search for other nodes and subsequently try to $connect()$ to all nodes found in this search

Algorithm 3 TreeNet – Simple Tree Topology Construction

```
loop
  while my_slaves < max_degree do
    found_nodes = inquiry();
    for all nodes in found_nodes do
      connect(node);
    end for
  end while
end loop
```

phase. Upon a successful connection between two nodes, the slave node stops inquiring and connecting (since it cannot inquire and connect anymore), while the master node continues if it has not yet acquired max_degree[1] slaves. This means that new connections are opened only between nodes that are not in a slave or master-slave role. Starting with disconnected piconets, the network topology evolves by connecting more and more piconets to each other and forming scatternet trees until, after multiple iterations, only one master node is left at the root of the final tree (see Figure 4-8). If a link fails, the root of the disconnected subtree is not a slave anymore and will thus start inquiring and connecting again. To speed up the connection process, the information obtained during inquiries (node addresses and clock offsets) is stored locally.

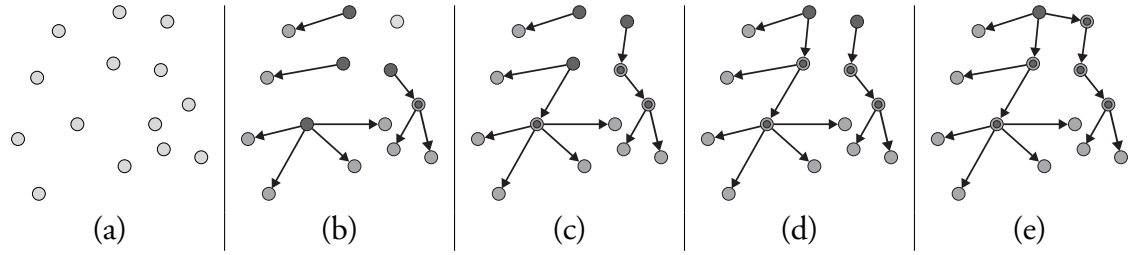

Figure 4-8
Schematic view of the TreeNet algorithm operation – (a) Initially disconnected, (b) first piconets form, (c) they interconnect to form first scatternet trees, until (d) larger scatternets are forming and the tree structure becomes visible, and (e) finally, a single tree has been constructed.

This algorithm allows the formation of large topologies in a robust, distributed and extremely lightweight fashion that is independent of any central controller, exhaustive computation or data exchange between nodes. Based on a simple search–and–connect scheme the TreeNet topology formation offers basic redundancy, self-configuration and even self-healing properties. Since this is clearly a best-effort approach there is no given guarantee for a specific topology or even optimization. If desirable, it is possible to add auxiliary topology control services to break up trees at specific connections and to optimize the topology after the initial construction phase of the TreeNet algorithm.

[1] This parameter is used to control the degree and hence depth of the final tree.

Data transport is organized based on the topology information. Upon a connect, leaf nodes (slaves) send their ID upwards in the direction of the root. Every intermediate node keeps a table of the respective subtree with only the root node (master) having a complete table of the network topology. This allows simple broadcast and unicast distribution of XHOP packets with remote commands from any node via the root node, sufficient for demo and initial deployment. However, this approach does not scale to infinitely large tree topologies.

4.3.4 Lessons Learned Through Experimentation

Of course real life in such experimentation does not come as cheap as proposed in our eight line algorithm in the previous section. Major issues that were discovered along the way from concept to final system validation were, that (i) an eight line, high-level algorithm leads to about 2000 lines of code in a functional implementation and (ii) it is very difficult to test, deploy and evaluate a large amount of devices. We will now discuss these two issues in more detail.

4.3.4.1 Code Size and Complexity

Given the constraints specific to our devices, the implementation and the highly non-deterministic behavior of the environment, extensions to the core TreeNet algorithm (algorithm 3) are close at hand to improve the overall performance.

First of all there are evident general lockup issues in the formation of the trees, that needs to be accounted for. A major restriction that is rooted in the restriction of the Bluetooth hardware to not be able to $connect()$ to slave roles is, that all nodes must be in visible range of each other for the algorithm to function and all nodes to be able to connect to a tree. Furthermore a set of nodes might not fully connect if multiple max_degree roots form because they will not be allowed to take on another slave connection due to the max_degree limitation.

In order to solve this problem and the lockup issues discussed above, a root node that can still see other nodes but has been in a max_degree state can try to disconnect one of it's slaves and try to connect to another node using algorithm 4. This is of course not a very efficient operation but it allows to gradually include all nodes into a balanced single tree using this simple and robust random back-off technique. Care has to be taken to only invoke algorithm 4 after a sufficiently long initialization phase for the whole network, e.g. when the topology control dynamics have settled down. In practice we have used timeouts of multiple tens of seconds to multiple minutes to detect a max_degree-root lock-up.

The problem of distributed $inquiry()$ and $connect()$ has been discussed earlier. A simple greedy algorithm that time-stamps operations and events can then try to $connect()$ to the last node seen on an $inquiry()$ first, expecting that node to be available for incoming connections since it has recently answered to an $inquiry()$. Also caching of all $inquiry()$ and $connect()$ attempts, their respective results and the retry count allows to use further heuristics and to adapt the periodic behavior to nodes with frequent failures. These operations are not very expensive in respect to the overall operation since they are performed completely on the host microcontroller and help to reduce especially the amount and the duration of the time-consuming and blocking $inquiry()$ and $connect()$ operations. They require to exchange the

Algorithm 4 Multiple max_degree-Root Lock-up Resolution

```
loop
  while my_slaves = max_degree do
    found_nodes = inquiry();
    if found_nodes ≠ 0 then
      current = randomly_select(open_links);
      disconnect(current);
      new = randomly_select(found_nodes \ current);
      connect(new);
    end if
  end while
end loop
```

for all $nodes$ in $found\_nodes$ **do** and $randomly\_select()$ functions in the above algorithms by a sorting and selection according to time-stamps.

An extension to sort and select nodes to connect to according to the received signal stength indicator values is certainly also desirable, but was not supported by the Bluetooth devices on the BTnode rev2. The RSSI resolution was too erroneous to be used for topology control (see section 4.4.4).

Apart from these enhancements, an application needs appropriate error handling and fall back mechanisms to handle all unsuccessful operations. The basic underlying infrastructure to be able to run such an algorithm thus additionally needs data storage, data exchange, timing functions, time-stamping of events, error handling as well as connection and link management as a minimum subset implemented on every node in the system. Note that algorithm 4 is already longer than the original TreeNet core algorithm. In total, this leads to about 2000 lines of additional code in our example implementation for the heuristically enhanced TreeNet application.

4.3.4.2 Large Scale Distributed Deployment

When it finally comes to simultaneous deployment on multiple devices, possibly away from the engineers desktop, the situation changes once more: How to debug, quantify, visualize and monitor the operation of a large number of distributed sensor devices without altering the operation of the system and without attaching additional infrastructure? Stepwise testing, deployment and validation are necessary steps in any design process.

Typical problems that appear here are the necessary cables, either for power or debugging and control, batteries, mounting, housing to shield the electronics from harmful influences of the environment, (re-)programming of all devices with software updates, debugging of a distributed concurrent system, developing for stepwise deployment, visualization and analysis of operations and the online access to nodes. Trial and error, system testing grids or brute force approaches are most common today, some examples are illustrated in Figure 4-9.

Extra functionality is necessary to enable to debug and monitor the function of every node

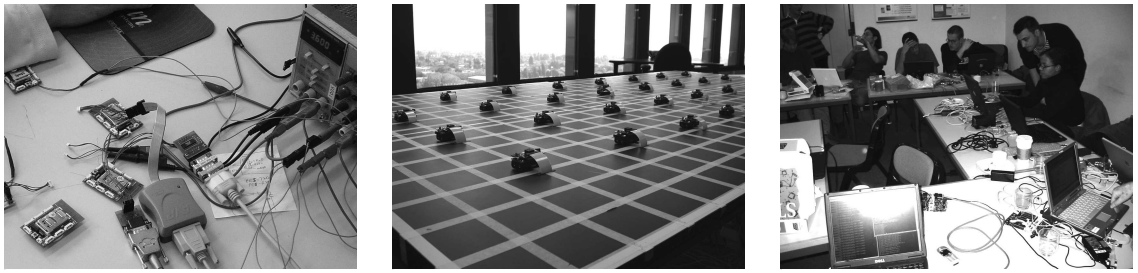

Figure 4-9
Sensor network development today – A typical engineers desktop with devices under test and programming devices (left), a static test-grid fixed onto a table using a large serial multiplexer to access individual nodes in the setup (middle) and collective debugging sessions with independent development setups (right).

until the final functionality can be verified. Today, this is typically done with a serial debugging cable and a control terminal for each node. This is of course not feasible in the field as well as with an increasing number of devices.

The TreeNet algorithm was implemented on the BTnodes and first demonstrated with over 40 devices in September 2003 at the NCCR-MICS annual project review, to that date and our knowledge the largest interconnected Bluetooth scatternet. In an initial phase the first scatternet trees appear quickly, on the order of a few seconds. It has shown that to reliably form a single tree out of all nodes is a harder task with durations on the order of multiple minutes. This is attributed to the reduced visibility of nodes as the algorithm progresses until in the final stage, there are exactly two devices left that have to match. The resulting trees are generally very stable and remain connected over hours, even in the presence of heavy interference from other Bluetooth devices and wireless local area networks. Since there are no ambiguous routes in a tree, only a simple packet forwarding mechanism was implemented that would allow a source routing type multihop data transfer. A demo application was implemented as a puzzle with light patterns being broadcast to all nodes from a control terminal for easy visualization and verification of the tree structure.

4.3.4.3 Experiences

With this work we have demonstrated the implementation of a functional, light weight application that is capable of forming robust and self-healing tree topologies in Bluetooth scatternets. We have shown that it is feasible to reliably interconnect many Bluetooth devices using simple algorithms. In our experience, Bluetooth has been easy to apply.

The wireless sensor network development reality encountered in this example has shown once again that it is very hard to deploy applications anywhere beyond 10–20 nodes. Of course it would be simple to argue that with sufficient manpower all implementation problems can be solved, but we have identified a key problem in this rather young field: Coordinated methods and tools for the complex, cross-layer development, testing, debugging, deployment and validation of systems composed of many devices are missing today.

In our example, a most simplistic eight line algorithm already bloats to well over 2000 lines

of code when implemented in a functionally satisfying way. With the additional support for stepwise refinement and debugging capabilities this increases even further, locking up extensive portions of the resources available on a platform such as the BTnode. This should remind algorithm designers to drastically reduce algorithm complexity and the demand for auxiliary functions when designing for resource constrained embedded devices.

Models and methods for the design of such systems usually do not contain abstractions for unreliable operations, e.g. link-failures, time-outs or unsuccessful operations. The integration of non-deterministic models into OS and deployment concepts is unclear today and there is no methodology for stepwise refinement and validation or large sensor networks. These are of course major issues for future work that will continue to require further trial-and-error type experimentation.

Specific to the TreeNet implementation on the BTnode rev2 we expect to improve performance considerably with the next generation devices that feature a full fledged Bluetooth 1.2 compliant front end. This will relieve many of the restrictions encountered on our present devices and allow for much more flexibility and new algorithmic opportunities. Nevertheless to achieve a functional and manageable system we believe that it is of imperative importance to go lean and lightweight in all aspects.

4.4 Scalable Topology Control for Deployment-Support Networks

Deployment-support networks have been proposed (see section 5.3.1) [BDH+04] as a non-permanent, wireless cable replacement. This approach allows to deploy and test large numbers of devices in a realistic physical scenario. The deployment-support network (DSN) is transparent, highly scalable, and can be quickly deployed. It does not disturb the target wireless sensor network any more than the traditional, cable-based approach. For the engineer, everything actually looks as if the usual cables were in place; he can thus use the same tools. The DSN nodes are attached to WSN target devices via a programming and debugging cable and form an autonomous network (see Figure 5-4). The WSN nodes can then be accessed through serial-port tunnels operated over the deployment-support network. With this tool, the limit for large-scale prototyping is pushed from simulation [LLWC03] and virtualization [GEC+04] to coordinated real-world deployment.

The BTnode is a most suitable platform for the implementation of a DSN prototype. Its Bluetooth radio perfectly meets the high bandwidth requirements for a cable replacement. Clearly, the energy consumption of a Bluetooth-based DSN node might be higher than that of the target sensor node, especially when the traffic on the Bluetooth connections of the deployment-support network is high. However, this is not critical since the required lifetime of a DSN node is comparatively small. Several techniques can be applied to improve the energy efficiency of the deployment-support network, for instance duty-cycling or using the various power modes of the microcontroller and the Bluetooth connections.

The Bluetooth standard contains no specifications for the formation and control of multihop topologies or for the data transport across multiple hops. An additional functional layer must

therefore provide these services. We chose a modular structure for this layer, i.e. topology control and data transport are independent of each other. In the following sections discusses a modular structure for the software running on each DSN node. First experiments were still performed on the BTnode rev2, with it's restriction to master-slave scatternets only, as described in section 4.2, using a new system software allowing concurrent threads (see section 4.4.1). In contrast to the TreeNet architecture, this new application architecture is based on a separate (i) connection manager and (ii) packet forwarding engine and thus well suited to be adapted to next-generation systems such as the BTnode rev3. The final implementation on the BTnode rev3 is discussed in section 4.4.2 including detailed experiments and evaluation.

4.4.1 Topology Control Prototyping

The basic operational idea of a simple, robust and completely local topology control algorithm derived from the TreeNet example discussed in section 4.3 forms the basis of a prototype topology control for deployment-support networks developed on the same hardware prerequisites as the TreeNet example (see section 4.3.1 for details). For reasons of system complexity and adaptability to new algorithms and platforms (see section 4.4.2) the application architecture was separated into two threads, each running a (i) connection manager and (ii) packet forwarding engine respectively. While supporting a clearly defined API this further facilitates the development and exchange of software components in a team without breaking the application. Initial experiments encompassing some 20–30 devices and spanning topologies of up to 10–12 hops prove the feasibility of the approach for larger experiments at a later stage documented in section 4.4.3.

4.4.1.1 Topology Control and Maintenance

The connection manager constructs and maintains a multihop network of the DSN nodes. It shall be a simple, robust, and completely local algorithm that automatically takes care of link failures and joining and leaving nodes. The basic principle of a simple, distributed connection manager algorithm is as follows: Every DSN node periodically searches for other nodes, and subsequently tries to connect to all nodes found.

The connection manager is restricted to to form tree topologies only due to the hardware constraints of the BTnode rev2 (see section 4.3.1.1). Since in a tree there is only one path between any two nodes, we are relieved from explicit route calculation; this reduces the system complexity considerably for the data transport using a packet manager, first experiments and evaluation.

Tree-building for a deployment-support network along the proposed scheme involves the detection of the main tree by assigning a unique tree ID to each subtree formed (see Algorithm 5). Upon detection of a node, the remote and local ID's are compared, and if they are not equal, a $connect()$ is performed. The larger ID is then broadcast to the respective subtree. At a later stage, this tree ID will be used to detect and eliminate cycles (see section 4.4.2.2).

Upon a successful connection between two nodes, the slave node stops inquiring and connecting (since it cannot inquire and connect anymore on the BTnode rev2), while the master node continues if it has not yet acquired max_degree slaves. This parameter is used to control the

Algorithm 5 DSN Connection Manager – Tree Construction and Maintenance

```
loop
  while my_slaves < max_degree do
    found_nodes = inquiry();
    for all node in found_nodes do
      remote_id = get_id(node);
      if remote_id ≠ my_id then
        connect(node);
        if remote_id > my_id then
          my_id = remote_id;
          broadcast(remote_id);
        else
          broadcast(my_id);
        end if
      end if
    end for
  end while
end loop
```

degree and hence depth of the final tree. This means that new connections are opened only between nodes that are not in a slave role. Starting with disconnected piconets, the network topology evolves by connecting more and more piconets to each other and forming scatternet trees until, after multiple iterations, only one master node is left at the root of the final tree (see Figure 4-8). If a link fails, the root of the disconnected subtree is not a slave anymore and will thus start inquiring and connecting again. To speed up the connection process, the information obtained during inquiries (node addresses and clock offsets) is stored locally.

This simple algorithm allows the formation of large topologies in a robust and completely distributed fashion. It does not need central control, or exhaustive computation or communication between nodes. Therefore, it can be expected to scale well to a large number of nodes. Despite its simplicity, our algorithm builds and maintains a self-healing network that tries to reconnect subtrees separated upon disconnects.

4.4.1.2 Multihop Packet Forwarding

The transport manager takes care of multihop packet forwarding. It receives information about the available connections from the connection manager and uses these connections to route packets. Note that the transport manager makes no assumptions about the underlying topology. The packet switching at every BTnode is based on virtual-circuit switching and automatically forwards traffic to the appropriate connection based on a virtual-circuit identifier.

For the transport manager, the concept of a host is important. A host is a DSN node to which a user device, e.g. a PC, is connected. Hence, a host is a source of commands to the deployment-support network and a sink for data from the deployment-support network. There can potentially be multiple hosts in a deployment-support network.

Communication is always initiated by a host, typically by opening a virtual connection to a specific DSN node. This is done by simply flooding the network with a route-request message. Each DSN node stores the ID of the connection such a message arrived on; this is the route to the host to be used on the return path. When the destination node sends its reply along the return path, the intermediate nodes assign a local virtual-circuit identifier to the connection the reply arrived on; this is the route to the destination node. After the setup is completed, packets can be transported over this virtual connection with only minimal header processing at the intermediate nodes.

If a link fails, the host and all endpoints of broken virtual connections are notified and remove the corresponding virtual-circuit identifiers from their local tables. A host application handles the retransmission of lost packets.

Multiple virtual connections are supported. If the connections are simultaneously active and if their paths overlap partially, the throughput of an individual virtual connection is obviously reduced.

4.4.1.3 Prototype Experimental Results

The following experimental setup was used for evaluation: 15–30 BTnodes are scattered randomly on a large desk. All nodes are programmed with the same software. A host PC is connected over a 115 kbps serial link to one of the BTnodes. The host PC configures the BTnode to be a host node in order to receive topology information. Each node stores connection-specific events such as new connections and link losses to a local log. The topology information and the logged events are remotely collected by a monitoring and control application running on the host PC (see Figure4-10).

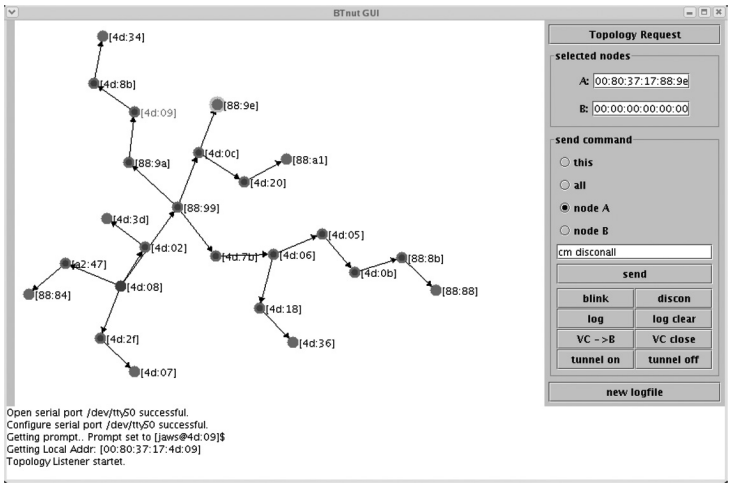

Figure 4-10
Centralized control and monitoring – A monitoring tool running on a host allows to monitor the network topology and control the deployment-support network. Here, monitoring and control is centralized with all topology information being gathered by the monitoring tool.

We will now discuss two aspects of the implemented deployment-support network: network-topology construction and the per-hop transmission delay.

Network-Topology Construction

The topology construction depends on the ability to discover other nodes and to successfully connect to them. These are highly non-deterministic operations since no a priori assumptions

about the state of remote nodes can be made. They may be inquiring, connecting, or in a slave role, which means that they are not visible to others at that time. Previous measurements have shown that the time for inquiring is a time-consuming process and in the order of tens of seconds [KL01, WMF02] for a reliable discovery of all nodes. Experimentation has shown that for our continuous and iterative connection manager, a short inquiry that is repeated often accelerates the formation of large clusters. Our experiments were conducted with the following values: inquiries last 3 sec and pauses between inquiries are chosen randomly between 5 and 20 sec.

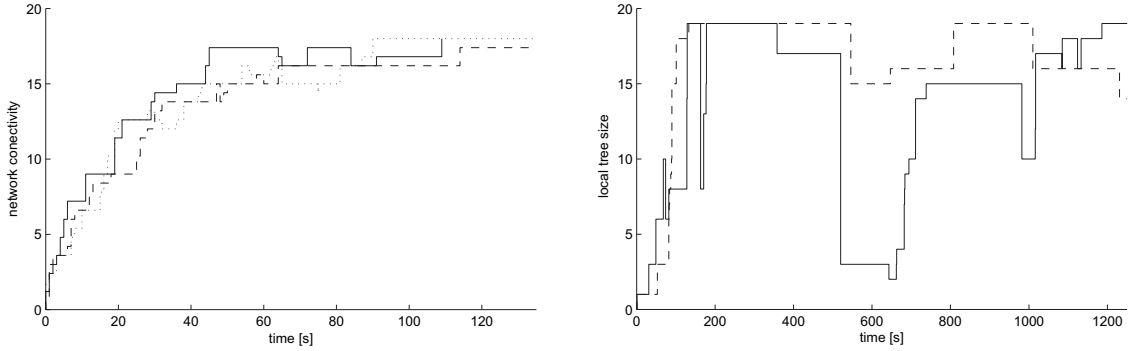

Figure 4-11
Initial network-topology construction and maintenance – Different experiments are shown here with 19 nodes. The connectivity is the total amount of connections in the network cluster (left). Upon link losses the self-healing maintenance re-establishes a connected tree topology (right).

Figure 4-11 (left) illustrates the evaluation of the initial connection events of 19 BTnodes. It shows three test runs of the initial topology construction with the total number of connections in the network with the connections are identical to the edges in the global topology graph of the network. The nodes start to form large clusters within approximately 60 sec. The self-healing property can also be seen here: connections that are dropped are subsequently repaired.

Bluetooth connections take time to be set up. Pending connection requests and lost connections that have not been detected by both endpoints are visible as steps of a half in Figure 4-11 (left). Here, the behavior of connection requests and successful connects can be seen in the quick steps in the left region. Disconnects that rely on the Bluetooth link supervision timeout that is typically set to multiple seconds have much longer half steps as can be seen on the right.

In contrast, in a tree structure without redundant connections, a failure of an arbitrary link may result in the disconnection of a large portion of the network. Figure 4-11 (right) shows two experiments with a view of the local tree size of two independent nodes connected to a host PC. Since this is the local view from a specific node, local tree sizes can vary over a large range. On subsequent connect and disconnect events, whole subtrees can be affected. Using the self-healing property of the topology control and maintenance algorithm, a recovery to a similar tree size can be accomplished within a rather short time.

Per-Hop Transmission Delay

On a virtual connection spanning multiple hops, the data has to be forwarded hop by hop. The transmission and processing delays add up along the path. We measured the delay by sending time-stamped packets to an endpoint. The packets are looped back to the sender, which measures the round-trip delay. Figure 4-12 shows the average round-trip delay divided by two. For different packet sizes we have measured and averaged the delay of 40 packets.

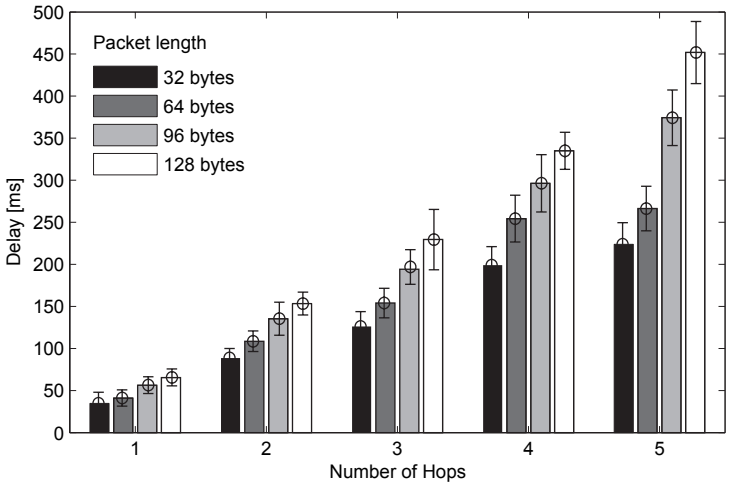

Figure 4-12
Per-hop transmission delay – Average multihop transmission delay and standard deviation for different packet lengths. The per-hop delay was measured to be 35–65 ms for the first hop and 45–90 ms for subsequent hops, depending on the packet size. For small packets, this results in approximately 50 ms per hop and per packet.

For serial-port tunneling, the end-to-end delay is of importance. The maximum tolerable delay for a serial connection depends on the application. Typically, a terminal session tolerates minutes before disconnecting, in-system programming can cope with a few seconds, and typing on an interactive user interface requires a maximal delay of 100–500 ms. Our measurements show that for the time being, we have to limit ourselves to tasks that do not require low-delay transmissions from the host to a DSN node many hops away. The local connection between DSN node and its attached WSN target node can however meet any delay demands of the target WSN target node. Delay sensitive applications like reprogramming a WSN target node can thus be done by first downloading all the data to the DSN node, and from there with negligible delay to the WSN target node.

4.4.1.4 Evaluation Based on the BTnode rev2

Looking at the overall experimental results, we can see that the concept of a deployment-support network is working out. We have successfully formed connected tree topologies spanning 20–30 BTnodes and operating multiple virtual connections with data rates up to a speed of 57.6 kbps. Self-healing tree topologies spanning 10–12 hops are autonomously constructed in tens of seconds to a few minutes. The experience from our experiments has shown that the initial tree formation is sufficiently fast and produces large network topologies. In the following discussion, we will focus on the characteristics of the deployment-support network itself and not so much on its interaction with a target wireless sensor network.

Due to the properties of our devices and the simplicity of the distributed algorithm, it is impossible to guarantee the formation of a single tree spanning all nodes in a given region. While

giving a connectivity guarantee is very hard in a random wireless environment, an optimized topology could be achieved by breaking up trees at specific connections to optimize the topology after an initial construction phase.

The tree topology was a reasonable choice for the initial implementation and the proof of concept. However, the experiments show link losses which result in the disconnection of potentially large subtrees, prohibiting long-term operation of virtual tunnels over larger hop distances.

Some of the problems encountered can be directly attributed to the rather old and sometimes unreliable Bluetooth modules we have used in these initial experiments. Devices making use of a next-generation Bluetooth subsystem, such as the upcoming BTnode rev3, will allow greater flexibility, enhanced stability, and an increased performance.

To achieve greater stability in case of link losses and reliable virtual-connection operation for days, network topologies with redundant links are clearly favorable. This would eliminate many of the limitations discussed earlier, but would require additional functionality to run on the DSN nodes. While redundancy is favorable in respect to robustness and network performance, it comes at a significant price: It will require advanced topology-shaping algorithms and of course more complex routing, which is presently reduced to simple packet forwarding.

The connection and transport manager are already designed to accommodate such functionality and many proposed algorithms for topology control and routing exist. But again success here depends on the implementation details and the concerted behavior of the nodes. Since a wireless network is not static, given that links can break and nodes can fail all components have to be able to operate in a dynamic environment with joining and leaving nodes.

Even in a simple algorithm as presented, several parameters have to be determined through practical experimentation. In our work, a major increase in performance was achieved with the selection of the correct duty-cycle parameters for the inquiry duration and period.

4.4.2 DSNtrees – Scalable Topology Control and Maintenance

The final version of the topology construction and maintenance makes use of the third generation hardware with a modern and capable Bluetooth module.

4.4.2.1 Bluetooth 1.2 Scatternets on the BTnode rev3

The improved Bluetooth device used on the BTnode rev3 uses an adaptive frequency hopping technique and advanced baseband control circuitry that enables a device to participate in four different physical channels at a given time. The Zeevo ZV4002 Bluetooth device on the BTnode rev3 is capable of interconnecting scatternets of up to four piconets simultaneously per node, with active connections to a maximum of seven slave and three master devices. Additionally, connection requests, data transfers, and inquiries can be performed simultaneously. The previous BTnodes rev2 generation imposed several constraints, most notably permitting only one slave role per node and making nodes with a slave role unable to perform or respond to inquiries or to open new connections. Compared to the Bluetooth interface used in the BTnode rev2 (see section 4.3.1.1) this allows to (i) connect not only top-down, i.e. to the root of a tree but to an arbitrary node of the network, (ii) form circular topologies using slave-slave

and master-slave roles and (iii) to stay discoverable and connectable at all times with the limitations of reduced node degree and role freedom as described. With the BTnode rev3, it is not anymore necessary for the operation of our deployment-support network that all nodes be within each other's transmission range.

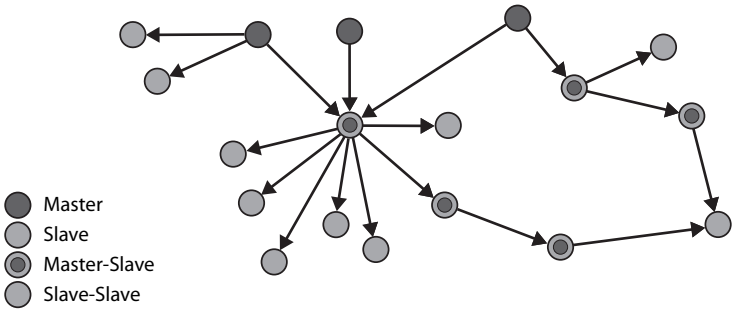

Figure 4-13
Adaptive frequency hopping scatternets on Bluetooth 1.2 allow multiple slave roles per device. This allows to use slave-slave bridging (lower right) and higher degree topologies with cycles while still being discoverable and connectable.

4.4.2.2 DSNtrees – Scalable Topology Control and Maintenance

In its current form, the connection manager forms tree topologies. Since in a tree there is only one path between any two nodes, we are relieved from explicit route calculation; this reduces the system complexity considerably, easing first experiments and evaluation. Other network topologies, e.g. planar mesh topologies [WZ04] but outside the scope of this thesis. Yet the software architecture is already set up to accommodate such adaptation.

The tree structure is constructed and maintained by two parallel threads. The inquiry thread (see Algorithm 6) periodically performs an inquiry and randomly connects to one of the discovered DSN devices.

The packet-handler thread (see Algorithm 7) processes negotiation or tree ID packets arriving from the lower layers. These packets are used to maintain the tree structure by preventing and detecting cycles in the network topology. All nodes connected in a tree share the same tree ID. When two nodes connect, they exchange negotiation packets and compare their tree IDs. If the two nodes were not in the same tree before the connection, their IDs differ and they have to establish a unique ID for the newly formed tree. This is done by agreeing on the larger of the two IDs and broadcasting it in a tree ID packet to all nodes in the subtree with the smaller ID. If two nodes that are already in the same tree connect, they will notice that they share the same ID and therefore drop the connection. If a node receives a tree ID broadcast with an ID different from its own, it adopts this new ID. If the received ID is its own ID, there is a cycle in the network and the link over which the broadcast arrived is dropped. This mechanism eliminates cycles that can arise when two subtrees are connected almost simultaneously via two different links (see Figure 4-14); in this case, the cycle prevention by negotiation does not work.

If a link is lost, the tree is partitioned, and the two subtrees must not share the same tree ID anymore. Therefore, if a node loses the link over which the current tree ID was received, the node broadcasts its unique device ID as its subtree's new tree ID.

This set of local algorithms provides self-healing topologies in a robust and completely distributed fashion. It does not need exhaustive computation or communication. The only

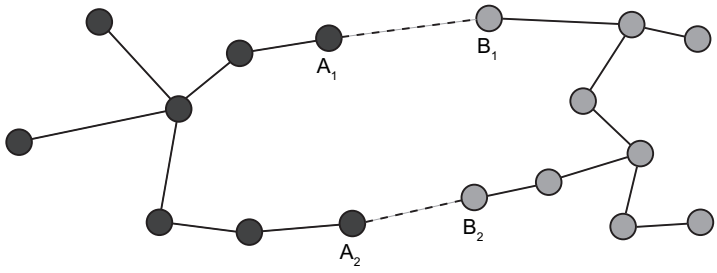

Figure 4-14
Cycles can form when disconnected trees are connected almost simultaneously at two different points. Here, A_i and B_i, $i \in \{1, 2\}$, connect. The tree-ID broadcast eliminates the cycle shortly afterwards.

Algorithm 6 DSNtrees Connection Manager – Random Inquiry Thread

```
loop
  found_nodes := inquiry();
  node := randomly_select(found_nodes);
  connect(node);
end loop
```

Algorithm 7 DSNtrees Connection Manager – Packet-handler Thread

```
loop
  packet := wait_for_packet();
  if packet.type = tree_ID_packet then
    if local_tree_ID = remote_tree_ID then
      disconnect(packet.link);
    else
      local_tree_ID := remote_tree_ID;
      broadcast local_tree_ID to my subtree
    end if
  end if
  if packet.type = negotiation_packet then
    if local_tree_ID = remote_tree_ID then
      disconnect(packet.link);
    else
      if local_tree_ID < remote_tree_ID then
        local_tree_ID := remote_tree_ID;
        broadcast local_tree_ID to my subtree
      end if
    end if
  end if
end loop
```

transmitted data that grows linearly with the number of nodes is the tree ID; since it does not change very often, it is negligible in the overall communication during the operation of the deployment-support network. The best effort search and connect with random back-off

assures that nodes will eventually connect. No expensive negotiation between neighbors and no a priory knowledge about the neighbors states (node degree, master/slave status, traffic requirements, etc.) are necessary. Therefore, it is expected to scale well to a large number of nodes.

4.4.3 Scalable Topology Control – Experimental Results

We have tested and measured the properties of our implementation in two different setups. The first one was a lab setup involving 2–40 nodes. We measured the per-hop transmission delay and observed the network-topology construction. In the second setup, we distributed 71 BTnodes on a large office floor, thus obtaining a larger, realistic deployment scenario.

In both setups, all nodes are running the same software. A host PC is connected over a 115 kbps serial link to one of the BTnodes. This node is a host in the deployment-support network and receives topology information from the other BTnodes: Each node sends information about events such as new connections and link losses to the host, and additionally stores them in a local log. The logs are remotely collected by a monitoring and control application running on the host PC (see Figure 4-15).

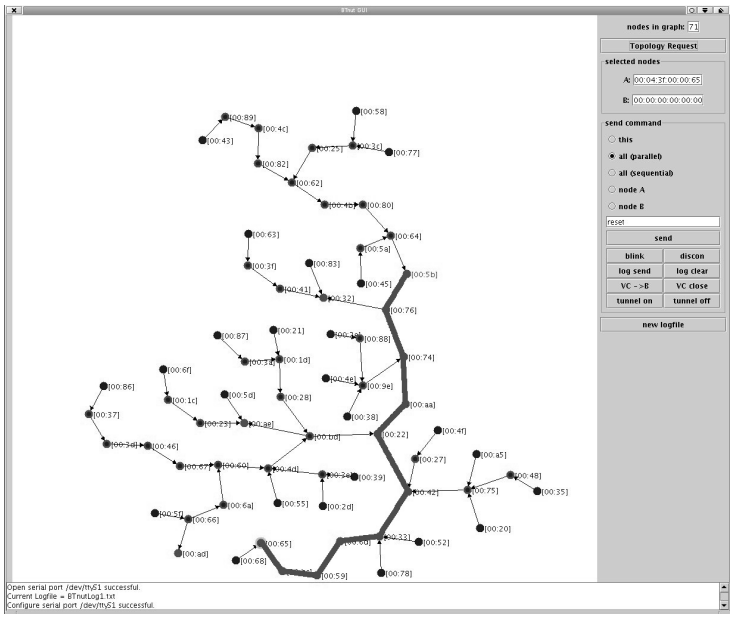

Figure 4-15
Scalable topology control – graphical user interface: The deployment-support network monitoring tool shows a scatternet tree topology with 71 nodes in a large experiment. A virtual connection from the host node `[00:5b]` over ten hops to node `[00:65]` has been established and is represented by the fattened line. Commands for remote execution at the network nodes can be entered using simple buttons or a command line interface.

4.4.3.1 Per-Hop Transmission Delay

On a virtual connection spanning multiple hops, the data has to be forwarded hop by hop. The transmission and processing delays add up along the path. We measured the delay by sending time-stamped packets to an endpoint. The packets are returned to the sender, which then measures the round-trip delay. Figure 4-16 shows the round-trip delay divided by two. For each hop count (up to 16 hops), we measured the average delay of 200 packets of two different sizes. The figure shows the expected linear behavior.

For serial-port tunneling, the end-to-end delay is of importance. The maximum tolerable delay for a serial connection depends on the application. Typically, a terminal session tolerates minutes before disconnecting, in-system programming can cope with a few seconds, and an interactive user interface requires a maximal delay of 100–500 ms. Our measurements show that for the time being, we have to limit ourselves to tasks that do not require low-delay transmissions from the host to a DSN node that is many hops away.

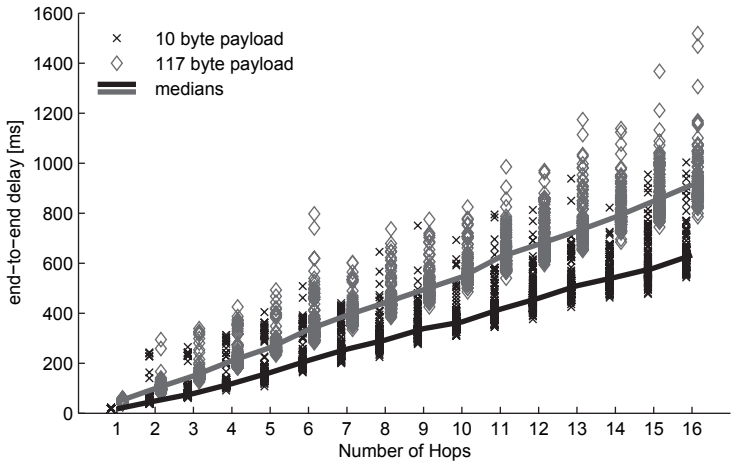

Figure 4-16
Per-hop transmission delay – The per-hop delay for the first hop is on average 17 msec for small packets and 45 msec for large packets. For subsequent hops, the average delays are 42 msec and 60 msec, respectively. The difference between the small and large packet delay is mainly the time needed for transferring more data between the Bluetooth device and the microcontroller.

4.4.3.2 Network-Topology Construction

The topology construction depends on the ability to discover other nodes and to successfully connect to them. Since a-priori assumptions about the state of remote nodes cannot be made before an actual connection, these are highly non-deterministic operations. While a node is inquiring or connecting, it might not be discovered by others. Previous measurements have shown that inquiring is a time-consuming process requiring in the order of tens of seconds [KL01, WMF02] for a reliable discovery of all nodes. Experiments have shown that for our connection manager, short but frequent inquiries accelerate the formation of large network clusters. Our experiments were conducted with the following values: inquiries last 3.8 sec and pauses between inquiries are chosen randomly (to avoid that all nodes inquire simultaneously) between 3 and 20 sec.

The scatternet-construction algorithm introduced in Sect. 4.4.2.2 is truly distributed. Since connections are established in parallel, the algorithm can be expected to scale well with an increasing number of nodes. We have verified this assumption with the following experiment.

Initially, n nodes are switched on one after the other. After all nodes are connected in a single tree, we simultaneously reset them with a broadcast command from the monitoring tool. This then provides a common time base for all nodes. All nodes log their connection and disconnection events, annotated with the time since the last reset. After all nodes are again connected, the monitoring tool retrieves these logs from all the nodes. Figure 4-17 illustrates the evaluation of the network-topology construction.

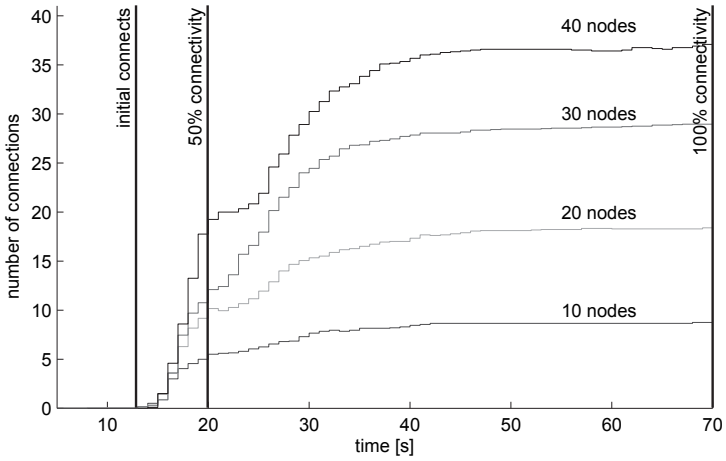

Figure 4-17
Initial network-topology construction – Each curve represents the average of ten different experiments using a constant amount of nodes. After a boot-up phase of approximately 13 sec, the first connections are established. At 20 sec, close to 50% of all the connections are established, and at 70 sec the construction is finished. These values are independent of the number of nodes involved.

All n nodes are connected in one tree if and only if there are $n - 1$ connections. In some of the 40 experiments, not all n nodes were connected in the end. A yet unresolved software error in the low-level event-handling routines occasionally caused a deadlock in the inquiry thread (Alg. 5). As a consequence, some nodes were not able to discover and connect to other nodes anymore and remained isolated.

This was the reason why the above experiments were not conducted with more than 40 nodes. If many nodes are reset simultaneously, not all of them can connect in the first iterations of the inquiry thread, probably due to radio interference. Thus, the probability that a node's inquiry thread enters the deadlock before the node is connected increases with increasing node density. This problem is not inherent to our algorithm and should disappear with the low-level software errors dissolved.

4.4.3.3 A Realistic Deployment Scenario

To test our deployment-support network in a realistic scenario, we deployed 71 BTnodes on a large office floor. We also wanted to test our hypothesis that the problem described in the previous section should disappear if we reduce the node density.

We therefore distributed the nodes as depicted in Fig. 4-18, switching them on as we went along. Being switched on one after the other, all 71 nodes joined a single tree scatternet (see Figure 4-15) without any problem. We then issued the reset command to all nodes. Within 70 sec, 46 nodes had again connected to a tree. As more time passed, the tree did however not grow beyond this size. Essentially, the problem remained as severe as in the lab setup.

The explanation for this is that there is no sufficient difference in connectivity between the lab setup and the floor deployment. This can be seen in Figure 4-18: the various connections over relatively long distances show that the average number of neighbors is still very high. The quality of the long-distance links in Figure 4-18 is probably smaller than that of the short-distance links. Furthermore, it may be desirable that the DSN nodes connected to co-located target nodes also be co-located in the deployment-support network. An improved version of

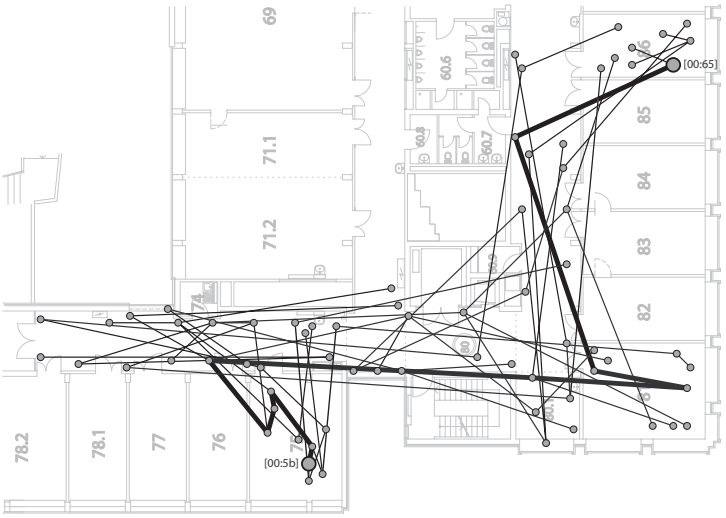

Figure 4-18
Random topology selection – real world geometry: The experiment with 71 nodes shown in Figure 4-15 was set up on a large office floor. The actual connections from Figure 4-15 are shown here; the virtual connection from node `[00:5b]` *over ten hops to node* `[00:65]` *is highlighted. This is the largest connected Bluetooth scatternet reported to date.*

our connection manager would hence prefer high-quality links.

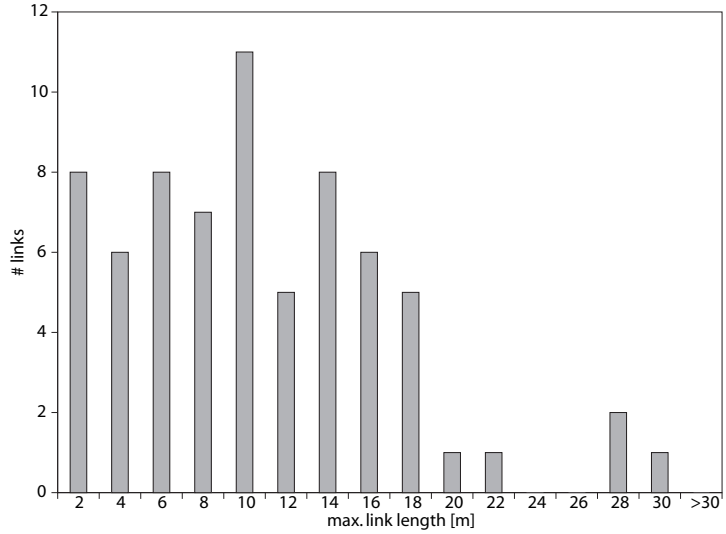

Figure 4-19
Random topology selection – link distance evaluation: The network topology derived in the experiment shown in Figure 4-15 and 4-18 yields the link-distance analysis results depicted on the left. Minimum 0.81 m, maximum 28.41 m, mean 9.76 m, standard deviation 6.31 m. Apart from a few long distance links, this shows an equal distribution of link lengths.

4.4.4 Random and RSSI-limited Selection Compared

The experiments have shown that the impact of the range of the devices was underestimated. In order to improve the topology control we investigated and incorporated the received signal stength indicator (RSSI) as a link metric. From the very first Bluetooth modules (see Figure 2-8) there has been a significant improvement as to the resolution and and error of the RSSI (see Figure 4-20). Moreover, an improvement in the Bluetooth standard allows to query the RSSI of a remote device with a HCI command already during the inquiry phase. The experimental evaluation has shown, that the RSSI values acquired during inquiry have a larger variance, but that they are still usable for topology control.

A simple change to the inquiry thread discussed in section 4.4.2.2 to select a node for connection not randomly, but according to the maximum RSSI value (see Algorithm 8) improved the

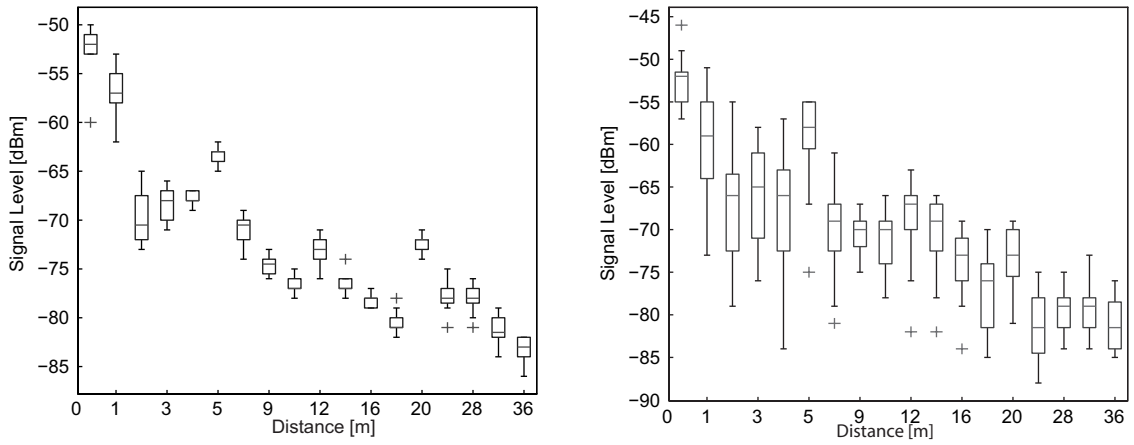

Figure 4-20
Bluetooth 1.2 received signal strength indicator – Two series of measurements are shown both for the received signal strength indicator while connected in a single point-to-point link (left), and during the inquiry phase (right).

resulting topology significantly in respect to the average link length.

Algorithm 8 DSNtrees Connection Manager – RSSI-limited Inquiry Thread

loop
$found\_nodes := inquiry();$
$node := select\_max\_rssi(found\_nodes);$
$connect(node);$
end loop

In similar experiments as described earlier, 30 nodes were distributed as shown for one example in Figure 4-21. Since, for larger deployments there were no links spanning similar maximum distances as shown in Figure 4-18, it was acceptable to only use a subset of the deployment described earlier. The resulting link-lengths for five independent experiments are shown in Figure 4-22. Compared to the histogram shown in Figure 4-19 the links are much shorter. But in order to achieve full connectivity using the topology control algorithm described, a certain amount of compromise has to be made. In the case were a node is unsuccessful in establishing a connection to it's estimated nearest neighbor (either due to interference or because the remote node cannot accommodate the new connection) the requesting node will back-off and randomly retry, resulting in the few, long range links that can be seen in Figure 4-21. The only way to counteract this, would be to introduce a negotiation of the network topology based on the local neighborhood estimation between nodes as proposed by Wattenhofer [WZ04] for mesh topologies.

4.4.5 Scalable Topology Control – Lessons Learned

Our algorithm works and performs well in principle, as the experiments have shown. The remaining problems can by large be attributed to system-software issues that can stall certain

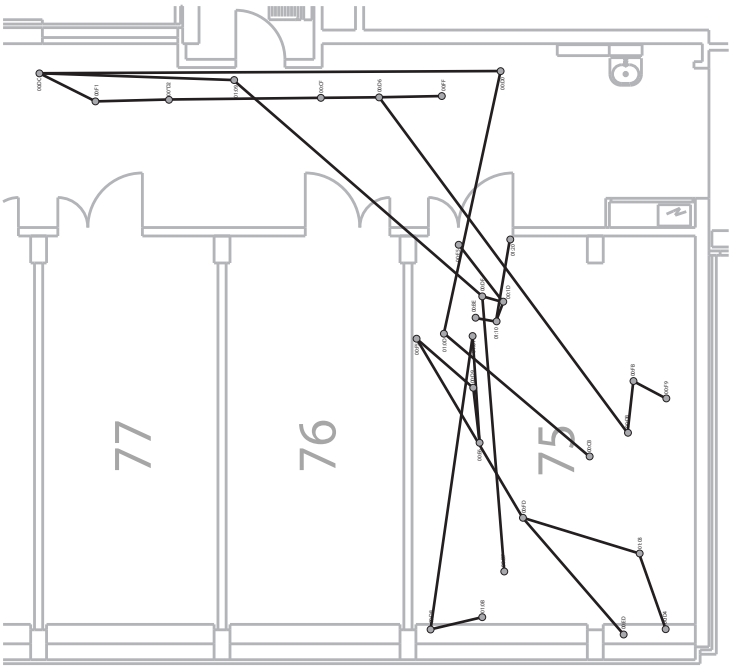

Figure 4-21
RSSI-limited topology selection – real world geometry: A similar experiment was repeated with the revised inquiry thread using RSSI-limited node selection. 30 nodes were deployed in a similar office environment. Since there is no inquiry-history or negotiation used between nodes, the longest links are created in phases where the random search only yields long-distance neighbors and one has to be selected and connected due to the connectivity requirement imposed.

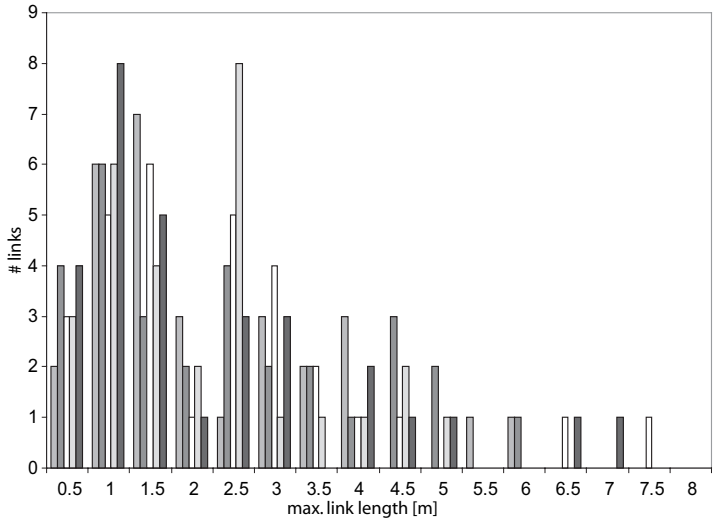

Figure 4-22
RSSI-limited topology selection – link distance evaluation: The experiment shown in Figure 4-21 yields on average significantly shorter links than the experiments based on random selection. Here, five independent runs of the same experiment with 30 nodes is shown. The key averaged values for all five experiments are: minimum 0.15 m, maximum 7.9 m, mean 2.19 m, standard deviation 1.78 m.

nodes without a possibility for recovery.

Using simple local algorithms has been a very encouraging experience due to their traceable and comprehensible nature. The increase in complexity when actually implementing an algorithm is not to be underestimated. In conjunction with the complex behavior of the devices, the exact interpretation of an effect can be very hard.

In current research, there is an apparent gap between the results of theoretical and practical work [KMW04]. Seemingly simple algorithms often rely on the availability of complex functions which are not readily supported by the actual hardware. For instance, a frequently used

function such as "send to all neighbors" typically has to be divided into smaller parts such as "search for all neighbors", "open a connection", "send the data", and "close the connection". If the function additionally has to be reliable, error-handling for all the subfunctions has to be provided to account for imperfections and failures. As described in sections 4.2, 4.3.1.1 and 4.4.2.1, peculiarities of the hardware may even make some functions impossible to implement.

A setup of many nodes in a small lab facilitates frequent and repeated experiments, but is not very realistic. In experiments involving radios, the high node density of a lab setup can actually lead to interference problems that would not occur in a sparse field setup. As a countermeasure, we reduced the inquiries of nodes connected to the host. This resulted in less interference and therefore faster connection buildup. Since we had observed that inquiring nodes occasionally lose links, the reduction of inquiries also increased the stability of the deployment-support network.

For the thorough analysis of a running system, the reliable extraction of data from the system is vital. This requires storing the data locally and synchronizing it upon extraction. When aggregating data from many nodes, coordination and throughput issues have to be managed. The data of unreachable (e.g. dead) nodes typically is of primary interest. For its extraction, it has to be stored in non-volatile memory and collected after a restart of the node.

4.4.5.1 Bluetooth Peculiarities

The effort to maintain a link can be separated into the control overhead necessary in the host, i.e. the Atmega128 CPU and the controller, i.e. the Zeevo ZV4002 Bluetooth system and the radio transmit and receive power necessary. Without transmit power-control (only required as a limiter in power class 1 and optional for power class 2 and 3 in Bluetooth) that is only available using an external power amplifier with the ZV4002 on the BTnode rev3, the power consumption is independent of the link-distance. The power budget of a single link is thus dominated by the control-overhead at both endpoints; to maintain a fully connected topology between any n nodes, $n-1$ links with similar energy requirements are mandatory by principle. Similar to the power budget, the radio interference at constant transmit power is more or less the same so that algorithms that try to achieve high spatial capacity by optimizing transmit power for minimum interference [BCSR03] are not of advantage here. In order to further optimize, the right way to go is link-state optimization using reduced duty-cycle (low-power) sniff and hold modes supported by modern Bluetooth devices such as explored by Negri [NBD05b].

With our algorithms presented, we cannot achieve nearest neighbor topology due to randomness introduced by the inquiry phase and a lack of negotiation. Only algorithms such as (XTC) proposed by Wattenhofer [WZ04] with first implementation results by Martin [Mar05] will be able to help here since they negotiate locally optimal topologies. Another benefit of such mesh topologies is the redundancy of links and as such resilience to link failures at the cost of more complicated data transport and routing.

In the sense of the application, it is not clear that a received signal stength indicator optimized topology is "better". It depends by large on the traffic patterns which topology is more a suitable than another. Is the majority of traffic between neighborhoods of one or two hops?

Between ten hops or even more? The question how many hops are actually preferable is a factor remaining often unclear today, although there is a trend to be observed that commercial systems are considering far less hops than theoretical studies.

4.4.5.2 Conclusions and Outlook

We have presented our implementation of a deployment-support network on the BTnode rev3 platform. Our measurements indicate that our implementation scales well to a large number of nodes. With a deployment-support network spanning 71 BTnodes, we have presented the largest Bluetooth scatternet reported to date.

After the first promising assessment of the DSN idea on the BTnode rev2, the transition to the new hardware was accomplished smoothly. The increased performance and reliability of the new devices has eased the implementation work considerably and has even surpassed our expectations.

The experiments reported here concentrate on the startup phase of the network; the maintenance phase necessitates further investigation. Thorough testing and measurements are needed to find optimal values for parameters such as the inquiry period and duration or the number of connection retries. These values can then be compared to those obtained by simulation [BBMP04]. The final result would be a network that is highly agile in the startup phase and that reduces the frequency of operations in the operating phase for maximum stability and minimum overhead.

Our first experiments were conducted with a simple tree topology. For increased resilience against link or node failures, redundant links are very desirable. The modular structure of our software allows us to easily replace the current connection manager. A comparison of various topologies with the reference tree topology can then be performed to trade off resilience against complexity.

5

Deployment – From Proof-of-Concept to Real-World Sensor Networks

With the increasing development of large-scale wireless sensor network applications, the coordinated development and deployment of sensor network devices are becoming an issue of increasing importance. Independent researchers have reported that when moving away from the engineer's desktop and beyond numbers of 10–20 nodes, deployment and testing become increasingly hard, and simulation will not solve all problems encountered [BKM+04, SPMC04]. While algorithms, system models, device architectures, and programming abstractions have been investigated for quite some time now, not much has been achieved in the area of deployment-support or even a concerted design, development and deployment methodology that allows for stepwise refinement and reliable monitoring of systems. Coordinated methods and tools for wireless sensor network deployment are missing today.

With our approach of a deployment-support network presented in this chapter, we push the limit for large-scale prototyping from simulation [LLWC03] and virtualization [GEC+04] to coordinated real-world deployment. The novel concept of the deployment-support network is based on the BTnode rev3 platform that has been presented in chapter 3. We motivate the need for concerted tools for the development and deployment based on related work presented in section 5.1 and continue to explain the details and operation of a deployment-support network in section 5.3. The design, implementation and testing work for a functional deployment-support network prototype would not have been possible alone. The work presented in this chapter was performed in collaboration with Philipp Blum, Matthias Dyer, Lennart Meier, Matthias Ringwald and others.

5.1 Design Tools and Development Methodology – Related Work

While the single embedded system is rather well understood both in its underlying technology and support by adequate design and development methods not much progress has been achieved to establish a methodology or a concerted design flow for the joint design and development of distributed networked embedded systems such as wireless sensor networks. The most important contributions in the area are summarized here.

5.1.1 System Design Aspects

From a system design perspective different approaches have been taken to develop energy-scalable algorithms [HSWC00] or application-specific protocol architectures for wireless sensor networks by Heinzelman [Hei00], Sinha [SWC02] and others.

The PicoRadio design flow aims at a combination of multiple disciplines, e.g. RF, low-power and low-voltage circuit design, antennas, networking, protocols and applications [RAdSJ+00], to create a bottom up methodology for the design of wireless sensor network nodes on a chip [dSJSA+01]. Different prototypes developed in the course of the project aim at prototyping certain aspects encountered during the progression of the project. Early systems based on an FPGA, a StrongARM processor are geared towards the prototype development of protocol and communication processing functions [TLR01, RAdSJ+00] whereas later systems specialize in low-power [RAK+02] and custom hardware integration [ROC+03].

For the network-centric component paradigm underlying the TinyOS system architecture and the underlying nesC programming abstraction see section 3.1.6.

The task of programming is tightly integrated with testing, debugging, and profiling (e.g. identifying performance bottlenecks and resource dissipation). In particular, all these tasks should – as far as possible – be performed at a similar level of abstraction. With the "generic role assignment" abstraction proposed by Frank [RFMB04, FR05], for example, role specifications to examine what is going wrong are mapped to C code.

5.1.2 Large Scale Simulation

Simulation as a typical means for evaluating algorithms and system properties has many followers in sensor networks. Apart from adapting well known networking simulators such as NS-2, Opnet or GloMoSim specialized simulation packages have been created to account for the embedded nature of the wireless sensor network devices. TOSSIM allows accurate and scalable simulation for TinyOS [LLWC03] with selectable simulation accuracy models. Here real TinyOS application are cross-compiled for the TOSSIM environment where multiple instantiations can be run simultaneously and communicate via an underlying radio model. Different such models exist as well as recent extensions for power-profiling within TOSSIM [SHC+04].

A comparable approach, based on a low-level abstraction of the AVR CPU architecture is Avrora [TLP05]. In contrast to TOSSIM, focus is more on the analysis and checking of embedded software prior to deployment on target hardware. The core simlator as well as different

profiling utilities and monitoring tools provide a basis infrastructure for experimentation, profiling and analysis [LWG05].

5.1.3 Virtualization and Emulation

A significant approach to bridge the gap between simulated and real world (see section 1.2) is the concept of virtualization pursued by the EmStar architecture [GEC+04]. The EmStar environment uses a ceiling-mounted grid of nodes connected to an emulation server to bridge between simulation and emulation in the testbed, but operation is complex and extensive infrastructure is necessary. Unique to this approach is the rather tight integration of real and virtual components in one framework, but the capability to capture all properties of the environment and of the devices themselves in the respective virtual counterparts is limited and complexity is high.

The BEE architecture [CKR+03, KCB+03, KCA+03] aims at prototyping the systems functionality of wireless transceiver systems prior to manufacturing of the integrated circuits. This is similar to widely adopted approaches of mixed hard- and software virtualization common in industrial SoC development, however the BEE makes use of extensive hardware resources to allow the virtualization of multiple system entities and elaborate channel models all on a single emulation platform.

5.1.4 Test Grids and Monitoring Tools

Laptops with wireless local area network (WLAN) support are commonly used in large-scale testbeds for reproducible ad hoc protocol evaluations such as performed by Lundgren using 37 laptops and network configuration tools for Linux running experiments based on the AODV, optimized link state routing (OLSR) and temporally-ordered routing algorithm (TORA) protocols [LLN+02]. This kind of approach is certainly not directly applicable to the much smaller and resource constrained WSN devices, but nevertheless, important system level insights can be achieved here through direct experimentation.

Similarly, researchers have used test grids fixed onto tables or ceiling mounts (see Figure 4-9) in conjunction with serial port multiplexers or serial–to–ethernet converters to access individual sensor network devices under test for monitoring in larger setups. Driven by the need for larger and automated experimentation, Welsh [WASW05] have developed moteLab, a testbed infrastructure based on Berkeley Motes attached to MIB600 programming boards each with direct ethernet connection, distributed across multiple office floors [HC02]. The ethernet connection is used for programming and for a monitoring back-channel using serial port tunneling. A web-enabled front end allows multiple users to use the Mote test grid in a time-shared manner by individual upload of a computing task to be performed on the devices. All responses and feedback is logged to a database for offline analysis.

The Mirage effort pursued at Intel Research is similar to moteLab but uses an auctioning mechanism for fair resource allocation to multiple users [CBA+05].

Connected to the work on EmStar, wireless network measurement tool based on packet delivery rates were developed for connectivity studies based on Mote hardware [CBE03] that can be

accesses using a fixed ceiling array or a portable laptop array [CE04] in EmStar.

The Kansei testbed consists of 210 extreme scaling motes (XSM) each connected to 210 extreme scale stargates (XSS). The stargates are connected using both wired and wireless ethernet providing a testbed infrastructure to conduct experiments with 802.11b networking and XSM [DGA+05]. This testbed has enabled the development of a 1000+ node wireless sensor network and a 200+ node peer-to-peer ad hoc network of 802.11b devices deployed within project ExScal [ARE+05].

Lifton [LSBP02] devised the Pushpin computing devices that form a flexible testbed for modeling, testing and deploying distributed peer-to-peer sensor networks with identical nodes in a controlled lab setting. Here the round Pushpin devices can be placed at random on a power-plane and attach to the power-grid through a set of pushpins. Flexible local communication is achieved either through infrared, capacitive coupling or radio modules that can be fitted onto the Pushpin devices. The Betha OS used, allows to process and exchange small programs and thus adds to the flexibility in experimentation. This work draws inspiration from the MIT Paintable Computing [But02] and the Amorphous Computing projects [AAC+00].

Wireless cable replacement is proposed for monitoring and diagnosing computer systems by Eberle [Ebe03]. The main goal of this work is to reduce cost and system complexity of the testing infrastructure only used temporarily. Here, wired, daisy-chained boundary-scan circuits are replaced by a more robust and flexible wireless network with direct point-to-multipoint connections to system components.

SensorScope [SDFV05] initially aims at solely operating wireless sensor network nodes in a realistic environment to gather sensor status and environmental data. However, increased usage with different objectives has led to the augmentation of an ethernet back-channel similar to moteLab.

The Bluetooth standard also offers the RFCOMM profile for serial-port communication, and various vendors are offering products based either on RFCOMM or on proprietary extensions. The drawback of these solutions is that they offer point-to-point connections between two endpoints only.

5.1.5 In-network Programming

Testbeds such as described above require frequent updates to the software running on the devices under test. Testbed infrastructure such as moteLab, Mirage and Kansei use a wired back-channel and auxiliary hardware for this task. Applications where a wired back-channel infrastructure is infeasible either require to collect and re-distribute all nodes manually or to use in-network programming services that operate on the wireless sensor network itself.

Tools for remote in-network programming of Mote sensor networks can be operated as a self-regulating code propagation service inside a sensor network [LPCS04], and it has been shown that this technique scales to large populations [HC04]. However, it requires alterations of the WSN devices, and offers only remote programming. The lack of dedicated feedback becomes even more important when monitoring data-centric systems without dedicated node identification [MFHH02].

5.1.6 Sensor Calibration and Verification

In his "Analysis of a Large Scale Habitat Monitoring Application" Szewczyk [SMP+04] reports that in order to understand the correlation between sensor reading and the actual situation in the environment, an auxiliary system consisting of cameras and a secondary network had to be used. With the application of sensor networks still in its infancy today, the problems of faulty, un-calibrated data derived from sensor networks remains unclear.

5.2 Full Life-Cycle Support for Sensor Networks

A typical design flow for embedded systems starts out from a concept and design phase where basic system properties and functions are defined and an evaluation of the different design alternatives takes place. Following this, prototypes are developed that are then programmed, tested, and debugged according to the specification created in the design process. The deployment phase is mainly characterized by extensive system testing in a realistic environment. When satisfying benchmark results have been achieved using a pilot deployment, a validation of the properties defined in the initial design specification has to be performed before a technical (type) approval required for an actual product launch can take place. For example, a fire alarm is required to report status of every node at least once every 30 sec (otherwise it is assumed that fire destroyed the detector), environmental sensors must deliver true and reliable readings over years.

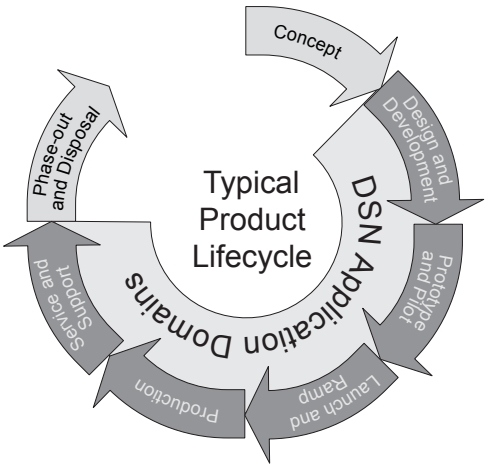

Figure 5-1
With future real-world sensor network applications emerging into successful industrial grade products, specific attention is required on all phases of the product life-cycle, namely from concept and design over development, prototype, pilot deployment, production, service and support finally leading to phase-out and decommission. In order to ensure customer and thus also commercial success, properties such as quality, reliability, and guaranteed performance are of paramount importance.

Such a process is neither new nor revolutionary, but a rather standard procedure in industrial product development. Variations of these processes exist, but in order to guarantee functional, timely, sustainable and correct sensor network applications with a reasonable use of resources (manpower), a minimum amount of iterations (cycles) in the design and development process is generally anticipated. With ever decreasing cost of the devices themselves, careless application design can thus lead to extensive development, deployment, and operating cost and so a prohibitive total cost of ownership.

In the context of sensor networks, the importance of coordinated methods and tools has been underestimated and so far concise approaches in this direction are missing. In the following we discuss the different phases of an example design flow in the light of the peculiarities of sensor networks. Particular emphasis is given to the fact that in a sensor network many devices have to cooperate in applications spanning collections of devices under real-world influence.

Concept, System Design and Development

In the earliest phase, system concepts and specifications are created based upon application requirements, early analysis and know-how from comparable projects. Subsequently system design tasks such as the selection of suitable components, partitioning into hard- and software building blocks and more detailed analysis of the required functions take place. As far as possible, standard components with well-known characteristics are used here to assure timeliness and cost-effectiveness through the reuse of modular components.

Stepwise design and development also means over-provisioning of resources to allow for changes at a later point in time and design-for-testability: Support for testing, debugging and monitoring have to be designed into systems already at an early stage. The use of automated tools accelerated the design and development process favorably. Early feedback from the systems under development back to designers is often required. The "classical" approach used for this feedback in the design and development of wireless sensor networks is simulation. After a satisfying design review, this phase concludes and leads to prototype development.

Prototype and Pilot Deployment

During prototype development the first system functionality is created, according to the specification created in the first conceptual phase. After initial checks of the functionality a pilot deployment is usually performed.

Feedback from such early prototype and subsystem tests into the development process often cause reiterations that are cumbersome as well as resource and time-consuming and may require to change initial concepts, design decisions and as a last resort, the specification.

Launch and Ramp

After a positive prototype review, deployment with use of the real target environment, actual numbers of devices and correct scale allows extensive system tests. Both the initialization phase and continuous operation are of interest here. Performance benchmarks, functional validation, verification under production conditions are required.

Production and Operation

Final handover to a customer or end-user requires a demonstration of functionality with sustainable operation under varying operating conditions. Often also a duplicate application to related domains is of concern here. In this case, means for system testing and validation are also required in this phase.

Service and Support

While monitoring of system status and operation is actually part of the operation in a final production environment, the reaction on the detection of failures and as far possible the correction of errors have to take place in the service phase. Depending on the requirements, upgrades or changes to a running system can be necessary too, requiring both flexible, transparent and re-targetable mechanisms.

Phase-out and Disposal

In the decommissioning phase different problems need to be accounted for: (i) A pollution and trash problem due to the devices put in place, (ii) post-mortem and system behavior analysis and (iii) to learn for subsequent development. While this sounds seemingly simple, researchers and practitioners report that non existent local logging facilities on sensor nodes or imprecise location tagging of devices deployed in the outdoors make this task next to impossible [SMP+04].

To create a simple yet effective means for full life-cycle service and support for the design and deployment of wireless sensor networks throughout all the phases described above is the goal of the methodology behind the next-generation deployment support presented in the next section.

5.3 Next-Generation Deployment-Support for Sensor Networks

Classic approaches to develop and deploy wireless sensor networks use serial cables for program download, control and monitoring. Although successful in lab setups, this approach is limited due to scalability issues and completely infeasible for deployment in the field. When moving away from the engineer's desktop and beyond numbers of 10–20 nodes, deployment and testing become increasingly hard. This is because wired connections to every node become infeasible. The loss of these connections reduces the possibilities for control and monitoring considerably, often resulting in trial-and-error procedures. Success and exact results then rely on sufficient manpower [HBAB04], individual skill [CEH+01], many iterations [MCP+02], and also a certain amount of luck [SOP+04].

In a coordinated design flow with stepwise refinement and validation (see Figure 5-2), it is vital to be able to monitor and control the target systems at all times. It is therefore desirable to be able to connect to any target device as if it were located on an engineer's desktop, ideally with minimal influence on the device's operation.

A step in the right direction are techniques that use the wireless sensor network itself to provide a connection to the WSN devices [LPCS04, HC04]. However, this implies altering the system and thus its behavior. Furthermore, it requires a relatively stable wireless sensor network. On failures, which occur frequently in early prototyping phases, a manual recovery is required.

A deployment-support network is useful throughout the entire development and deployment cycle allowing transparent access to all system components under development with minimum impact on the devices and systems being developed: In an early phase, where the focus is on getting the first functions operational and distributing new code to all devices; during system

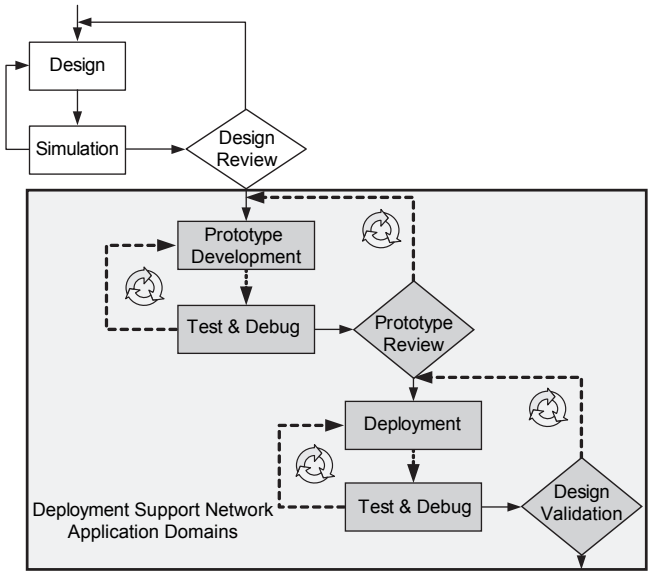

Figure 5-2
Deployment-support network enable stepwise refinement – Any wireless sensor network system can be programmed, tested, debugged, deployed and verified both on- and offline while being embedded in the physical environment. The often necessary iterations in the design, development and deployment process can be considerably reduced while enabling full life-cycle service and support.

testing in the final stages of development; in a production-like deployment phase where monitoring, validation and measurement with minimal interference are the primary focus. With the DSN approach, operation and testing in the field is made possible, significantly decreasing the amount of necessary cycles in the development process and achieving correct implementations in a timely manner. A deployment-support network can be used by both skilled systems-engineers, installation personnel, operators and end-users and especially non system experts.

5.3.1 Deployment-Support Networks

Deployment-support networks (DSNs) [BDH+04] have been proposed as a non-permanent, wireless cable replacement for the development, test, deployment and validation of sensor networks in different environments (e.g. first in a lab setup and subsequently in the field). This approach allows to deploy and test large numbers of devices in a realistic physical scenario. The DSN is transparent, highly scalable, and can be quickly deployed. Due to its nature of an overlay network and autonomous operation, it does not disturb the target sensor network any more than the traditional, cable-based approach. For the engineer, everything actually looks as if the usual cables were in place; he can thus use the same tools. The DSN nodes are attached to sensor network target devices via a programming and debugging cable and form an autonomous network. The sensor nodes can then be accessed through serial-port tunnels for online monitoring, debugging, and uploading software updates, operated over the DSN (see Figure 5-3). With this tool, the limit for large-scale prototyping is pushed from simulation [LLWC03] and virtualization [GEC+04] to coordinated real-world deployment. The DSN acts as an infrastructure-less external observer with minimum impact on the observed system, offering a seamless transition from the lab to the field, and – when removed after a validation and verification phase – autonomous operation under production conditions [BDMT05].

The ability to reliably connect to a WSN target device without altering it can be achieved by simply attaching a DSN device to every WSN target device, and letting the DSN devices

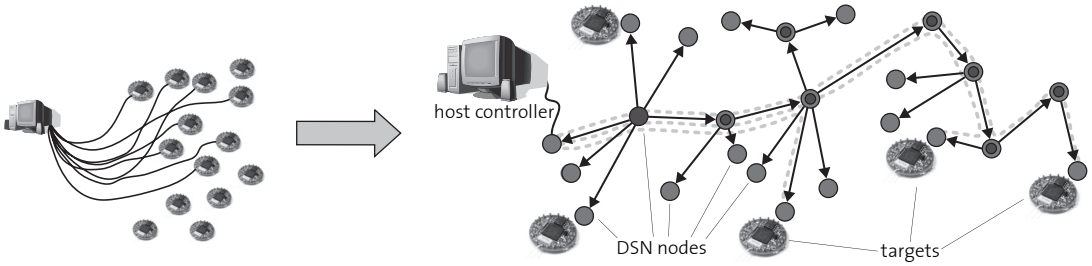

Figure 5-3
The approach of a deployment-support network is a wireless serial-cable replacement offering reliable and transparent connections by attaching a BTnode to every target node, e.g. a Berkeley Mote, in a certain deployment scenario. Using the target wireless sensor network itself for control and monitoring is not an option since any additional traffic could disturb the actual WSN application, and in the early phases of development the target wireless sensor network is too unreliable.

construct and maintain an autonomous multihop network. A host, e.g. a PC, can tap into this network by attaching to one of the DSN devices, and open a virtual connection to an arbitrary DSN node. The host is then able to communicate both with this node and with its attached target. Multiple virtual connections are possible, either originating at the same host, or at hosts attached to different DSN nodes.

There are different possibilities to interact with the target, such as serial-port connections for target monitoring and control, and remote target programming and resetting. In the case of serial-port tunneling, the target's serial port is replicated on the host system. The tunneled data can be used as if the target device was directly attached to the host. Standard tools like an in-system programmer or an in-circuit debugger can be used without modification, although the target may be multiple hops away. For operations requiring access to specific IO pins of the target device, e.g. remote programming or resetting, additional general-purpose IO pins of the DSN node have to be used. A general overview of the DSN components and some exemplary host–target virtual connections are shown in Figure 5-4.

5.3.2 Deployment-Support Network Prototype

First experiences with a deployment-support network prototype has been made using BTnode platform discussed in chapter 3 as well as the topology control and multihop data transport mechanisms presented in chapter 4. The basic operation can be described as follows: The BTnode devices construct and maintain a multihop backbone network. A host, e.g. a PC, can then attach to any of them and open a virtual connection to an arbitrary BTnode in the network. The host can then communicate both with this node and with its attached target through the virtual connection. So far, we have implemented remote programming, target control and monitoring (see Figures 5-4 and 5-3); other BTnode–target operations are also possible.

In this application, a multihop network formed by Bluetooth Scatternets is most suitable due to the reliable link layer, multiplexing, QoS capabilities and rather high bandwidth of Bluetooth. This bandwidth is necessary when tunneling the aggregate traffic of multiple virtual connec-

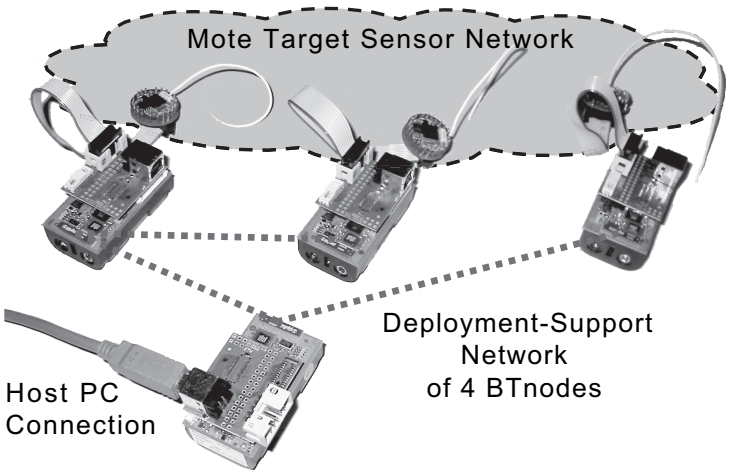

Figure 5-4
Each WSN target device (Mica2Dot Mote) is attached to a DSN node (BTnode rev3) using an adapter cable. In this example, three Motes can be debugged via a four-node deployment-support network spanning three hops and a host PC attached to the deployment-support network.

tions. A simple, robust and distributed algorithm constructs a tree topology. The tree topology is self-healing, i.e. it automatically takes care of joining and leaving nodes.

The virtual connection replaces the direct host–target serial cable. The packet switching at every BTnode is based on asynchronous transfer mode (ATM) virtual circuits and automatically forwards traffic to the appropriate connection. Thus, no routing is necessary on our tree topology.

Benefits of this deployment-support network are scalable and transparent connections to arbitrary target devices. It is plug-and-play, self-configuring and requires no alteration of the target system. In demonstration setups, we have successfully built tree topologies spanning 30–40 BTnodes and having multiple virtual connections with data-rates up to 57.6 kbps.

5.3.3 Beyond the Function of a Deployment-Support Network

But even now that deployment-support networks have been proposed we are still lacking proper analysis tools that can make use of distributed monitoring data from many nodes and feed them back into the design cycle for refinement. How do we assess the performance of a whole system? How do we analyze the interaction and synergies between nodes? Is it possible to validate? Is it possible to quantify correctness of the application? It remains a challenge to create appropriate metrics and benchmarks for profiling, validation and functional verification of sensor network applications that are vital when approaching type approval, required as the final development step before a real-world product launch.

The advantages of a deployment-support network are:

++ Full life-cycle service and support for the development and deployment of wireless sensor networks

++ De-coupling from an yet unreliable wireless sensor network while under development through use of a secondary overlay network

++ Minimal invasive, i.e. not reliant on resources on the devices under test

++ Simple application by both skilled system engineers and non system experts

++ On- and offline usage

++ Plug and play operation

++ Testing and deployment in any target environment using autonomous, battery-powered, wireless devices

6

Conclusions

In this chapter we summarize the contributions of our work and discuss possible directions of future work.

6.1 Summary of Contributions

With the work presented in this thesis we contribute towards a systematic approach for the development and deployment of wireless networked embedded systems. By providing a view on different aspects of the design, development and deployment process we warrant a broad applicability of the concepts and methods discussed.

In chapter 2 we develop a location management service for wireless sensor networks based on the specific characteristics of the network topology and resources encountered on such systems. This is one of the first algorithms to be described in the literature. We analyze the functional and qualitative requirements necessary for implementation based on measurements and simulation.

An analysis of current wireless sensor network platforms in chapter 3 motivated the design of the BTnode platform based on a need for event-driven interaction, standardized interfaces, flexible and sufficient resources, development and deployment support to allow convenient fast-prototyping of wireless sensor network applications. Especially the ability to connect to other Bluetooth enables devices and the system software support using plain C programming have made the BTnode a successful platform with numerous applications both in education and research.

Based on the BTnode platform, we have developed Bluetooth multihop topology control algorithms suitable to the prerequisites of the platform and conducted several practical experiments. These experiments constitute the largest connected Bluetooth scatternets with 70+ nodes reported of to date.

The experience gained through numerous implementations and the lack for appropriate development and deployment support has led to the development of the concept of a deployment-

support network as a powerful tool for the development, deployment, test and validation of wireless sensor networks. We have shown that scalable deployment-support networks can be implemented on currently available hardware.

6.2 Future Work and Concluding Remarks

Our experiences and developments have spawned a number of interesting questions to be pursued in continuing work.

The next step, which has already been started, is to test the deployment-support network with a real-world sensor network application in a larger test setup. This will allow to measure performance and adapt operating parameters and performance based real data. Future activities will include the measurements of throughput, power consumption, and the interference with other radios. The tree topologies used so far are very susceptible to single link failures, as every link failure cuts off a whole subtree. Therefore investigation based on mesh topologies, e.g. XTC are already under way and an optimal tradeoff between redundant topologies and routing overhead will have to be found.

Distributed debugging, based on the deployment-support network and it's simultaneous connections to many target devices will allow to derive knowledge about the distributed operation and interaction of nodes that can then be fed back into the design and development process. However it is unclear how this is to be done, i.e. (i) the probing interface at the targets should be minimal invasive as not to influence the target device and (ii) how to analyze and reason from the collected data.

The multiple radio front ends on the BTnode rev3 also deserve attention. Proposals have been made for the operation of dual radios with differing characteristics on wireless sensor networks and many mobile computing and communication consumer devices today feature a host of different interfaces. In the domain of wireless sensor networks this has so far only been exploited to bridge between infrastructure networks and wireless sensor networks but not for tradeoffs or the enhancement of performance.

Over the past years many platforms for wireless sensor networks have been developed too numerous to be counted and standardized software and interfaces are emerging. This obviously suggests that one should consider not to build another custom platform from scratch as we have done for the BTnode but to reuse existing components. Practice has shown, that it takes more time, experience, commitment and careful attention to details and constraints to fully understand and make profitable use of a platform as one might think at first. In some cases you might as well have developed such a system yourself in the meantime. However we strongly advocate to make use of existing, standardized, modular components and technology where ever possible. With the emergence of tools, methodologies, support of general applicability and standardized abstractions the limitations imposed and symptoms of infancy so dominating in practice today will hopefully cease over time.

The biggest problem we have encountered, was actually developing the deployment-support network application without having a DSN already in place to be used for the development and deployment work.

Bibliography

[AAC+00] H. Abelson, D. Allen, D. Coore, C. Hanson, G. Homsy, and T.F. Knight. Amorphous computing. *Communications of the ACM*, 43(5):74–82, 2000.

[ABD+04] U. Anliker, J. Beutel, M. Dyer, R. Enzler, P. Lukowicz, L. Thiele, and G. Tröster. A systematic approach to the design of distributed wearable systems. *IEEE Transactions on Computers*, 53(8):1017–1033, August 2004.

[ACH+01] M. Addlesee, R. Curwen, S. Hodges, J. Newman, P. Steggles, A. Ward, and A. Hopper. Implementing a sentient computing system. *IEEE Computer*, 34(8):50–56, August 2001.

[AMS02] S. Antifakos, F. Michahelles, and B. Schiele. Proactive instructions for furniture assembly. In *Proc. 4th Int'l Conf. Ubiquitous Computing (UbiComp 2002)*, volume 2498 of *Lecture Notes in Computer Science*, pages 351–360. Springer, Berlin, September 2002.

[Ant04] S. Antifakos. *Improving Interaction with Context-Aware Systems*. PhD thesis, Dept. Computer Science, ETH Zürich, Switzerland, 2004.

[ARE+05] A. Arora, R. Ramnath, E. Ertin, P. Sinha, S. Bapat, V. Naik, V. Kulathumani, H. Zhang, H. Cao, M. Sridharan, S. Kumar, N. Seddon, C. Anderson, T. Herman, N. Trivedi, C. Zhang, M. Nesterenko, R. Shah, S. Kulkarni, M. Aramugam, L. Wang, M. Gouda, Y. Choi, D. Culler, P. Dutta, C. Sharp, G. Tolle, M. Grimmer, B. Ferriera, and K. Parker. ExScal: Elements of an extreme scale wireless sensor network. In *Proc. 11th IEEE Int'l Conf. Embedded and Real-Time Computing Systems and Applications (RTCSA 2005)*, page to appear, 2005.

[Bas02] S. Basagni. Remarks on ad hoc networking. In E. Gregori, G. Anastasi, and S. Basagni, editors, *Advanced Lectures on Networking: NETWORKING 2002 Tutorials*, volume 2497 of *Lecture Notes in Computer Science*, pages 101–123, Pisa, Italy, May 2002. Springer, Berlin.

[BB03] A.L. Barabási and E. Bonabeau. Scale-free networks. *Scientific American*, pages 60–69, May 2003.

[BBDL03] P. Bonnet, A. Beaufour, M.B. Dydensborg, and M. Leopold. Bluetooth-based sensor networks. *ACM SIGMOD Record*, 32(4):35–40, December 2003. SPECIAL ISSUE: Special section on sensor network technology and sensor data managment.

[BBDM05] J. Beutel, P. Blum, M. Dyer, and C. Moser. *BTnode Programming - An Introduction to BTnut Applications*. Computer Engineering and Networks Lab, ETH Zürich, Switzerland, 1.0 edition, May 2005.

[BBEH02] N. Bulusu, V. Bychkovskiy, D. Estrin, and J. Heidemann. Scalable, ad hoc deployable, RF-based localization. In *Proc. Grace Hopper Conf. Celebration of Women in Computing*, October 2002.

[BBMP04] S. Basagni, R. Bruno, G. Mambrini, and C. Petrioli. Comparative performance evaluation of scatternet formation protocols for networks of Bluetooth devices. *Wireless Networks*, 10(2):197–213, March 2004.

[BBP03] S. Basagni, R. Bruno, and C. Petrioli. A performance comparison of scatternet formation protocols for networks of Bluetooth devices. In *Proc. 1st IEEE Int'l Conf. Pervasive Computing and Communications (PerCom 2003)*, pages 341–350. IEEE CS Press, Los Alamitos, CA, March 2003.

[BBS+01] L. Blazevic, L. Buttyan, Capkun S., S. Giordano, J.P. Hubaux, and J.Y. Le Boudec. Self organization in mobile ad hoc networks: The approach of Terminodes. *IEEE Communications Magazine*, 39(6):166–174, June 2001.

[BCSR03] R. Balaji, J.K. Chen, S. Shakkottai, and T.S. Rappaport. Connectivity of sensor networks with power control. In *Proc. 37th Asilomar Conf. Signals, Systems and Computers, 2003*, volume 2, pages 1691–1693. IEEE, Piscataway, NJ, November 2003.

[BD05] J. Beutel and A. Dogan. Using TinyOS on BTnodes. In *Proc. 4th GI/ITG KuVS Fachgespräch Drahtlose Sensornetze*, Technical Report 418, pages 6–10. Dept. Computer Science, ETH Zürich, Switzerland, March 2005.

[BDH+04] J. Beutel, M. Dyer, M. Hinz, L. Meier, and M. Ringwald. Next-generation prototyping of sensor networks. In *Proc. 2nd ACM Conf. Embedded Networked Sensor Systems (SenSys 2004)*, pages 291–292. ACM Press, New York, November 2004.

[BDMT05] J. Beutel, M. Dyer, L. Meier, and L. Thiele. Scalable topology control for deployment-sensor networks. In *Proc. 4th Int'l Conf. Information Processing in Sensor Networks (IPSN '05)*, pages 359–363. IEEE, Piscataway, NJ, April 2005.

[BDO+05] S.J. Bellis, K. Delaney, B. O'Flynn, J. Barton, K.M. Razeeb, and C. O'Mathuna. Development of field programmable modular wireless sensor network nodes for ambient systems. *Computer Communications*, page to appear, 2005.

[Beu04] J. Beutel. *Handbook of Sensor Networks: Compact Wireless and Wired Sensing Systems*, chapter Location Management in Wireless Sensor Networks. CRC-Press, Boca Raton, FL, 2004.

[Beu05] J. Beutel. Robust topology formation using BTnodes. *Computer Communications*, page to appear, 2005.

[BG03] M. Beigl and H. Gellersen. Smart-Its: An embedded platform for smart objects. In *Proc. Smart Objects Conference (SOC 2003)*, Grenoble, France, May 2003.

[BGS00] P. Bonnet, J. Gehrke, and P. Seshadri. Querying the physical world. *IEEE Personal Communications*, 7(5):10–15, October 2000.

[BHE00] N. Bulusu, J. Heidemann, and D. Estrin. GPS-less low-cost outdoor localization for very small devices. *IEEE Personal Communications*, 7(5):28–34, October 2000.

[BHE01] N. Bulusu, J. Heidemann, and D. Estrin. Adaptive beacon placement. In *Proc. 21st Int'l Conf. Distributed Computing Systems (ICDCS 2001)*, pages 489–498. IEEE, Piscataway, NJ, April 2001.

[BHS03] A. Boulis, C.C. Han, and M.B. Srivastava. Design and implementation of a framework for programmable and efficient sensor networks. In *Proc. 1st ACM/USENIX Conf. Mobile Systems, Applications, and Services (MobiSys 2003)*, pages 187–200. ACM Press, New York, May 2003.

[Big89] N.L. Biggs. *Discrete Mathematics*. Oxford University Press, New York, revised edition, 1989.

[BKM+04] J. Beutel, O. Kasten, F. Mattern, K. Römer, F. Siegemund, and L. Thiele. Prototyping wireless sensor network applications with BTnodes. In *Proc. 1st European Workshop on Sensor Networks (EWSN 2004)*, volume 2920 of *Lecture Notes in Computer Science*, pages 323–338. Springer, Berlin, January 2004.

[BKR03a] J. Beutel, O. Kasten, and M. Ringwald. BTnodes – a distributed platform for sensor nodes. In *Proc. 1st ACM Conf. Embedded Networked Sensor Systems (SenSys 2003)*, pages 292–293. ACM Press, New York, November 2003.

[BKR03b] J. Beutel, O. Kasten, and M. Ringwald. BTnodes – applications and architecture compared. In *Proc. 1st GI/ITG KuVS Fachgespräch Drahtlose Sensornetze*, TKN Technical Report TKN-03-012, pages 34–37. Telecommunication Networks Group, Technical University Berlin, July 2003.

[BP00] P. Bahl and V.N. Padmanabhan. RADAR: An in-building RF-based user location and tracking system. In *Proc. 19th Ann. Joint IEEE Conf. Computer Communiucation Soc. (Infocom 2000)*, volume 2, pages 775–784. IEEE, Piscataway, NJ, 2000.

[BP02] S. Basagni and C. Petrioli. Multihop scatternet formation for Bluetooth networks. In *Proc. 55th IEEE Semiannual Vehicular Technology Conference (VTC Spring 2002)*, pages 424–428. IEEE, Piscataway, NJ, May 2002.

[Bul02] N. Bulusu. *Self-configuring Localization Systems*. PhD thesis, Dept. Computer Sciences, Univ. of California, Los Angeles, CA, 2002.

[But02] W.J. Butera. *Programming a Paintable Computer*. PhD thesis, Massachusetts Institute of Technology, February 2002.

[BWG99] V. Bose, D. Wetherall, and J. Guttag. Next century challenges: RadioActive networks. In *Proc. 5th ACM/IEEE Ann. Int'l Conf. Mobile Computing and Networking (MobiCom '99)*, pages 242–248. ACM Press, New York, August 1999.

[Caf00] J.J. Caffery. A new approach to the geometry of TOA location. In J.H. Weber, J.C. Arnbak, and R. Prasad, editors, *Proc. 52nd IEEE Vehicular Technology Conf. Fall 2000 (VTC 2000)*, volume 4, pages 1943–1949. IEEE, Piscataway, NJ, 2000.

[CBA+05] B.N. Chun, P. Buonadonna, A. AuYoung, C. Ng, D.C. Parkes, J. Shneidman, A.C. Snoeren, and A. Vahdat. Mirage: A microeconomic resource allocation system for sensornet testbeds. In *Proc. 2nd IEEE Workshop on Embedded Networked Sensors (EmNetS-II)*, page to appear. IEEE, Piscataway, NJ, May 2005.

[CBE03] A. Cerpa, N. Busek, and D. Estrin. SCALE: A tool for simple connectivity assessment in lossy environments. Technical Report 21, Center for Embedded Networked Sensing (CENS), Univ. of California, Los Angeles, CA, September 2003.

[CCM+01] M.V.S. Chandrashekar, P. Choi, K. Maver, R. Sieber, and K. Pahlavan. Evaluation of interference between IEEE 802.11b and Bluetooth in a typical office environment. In *Proc. IEEE Symp. Personal, Indoor and Mobile Radio Communications (PIMRC 2001)*, volume 1, pages 71–75, September 2001.

[CE04] A. Cerpa and D. Estrin. ASCENT: Adaptive self-configuring sensor networks topologies. *IEEE Transactions on Mobile Computing*, 3(3):272–285, July 2004.

[CEH+01] A. Cerpa, J.E. Elson, M. Hamilton, J. Zhao, D. Estrin, and L. Girod. Habitat monitoring: application driver for wireless communications technology. *ACM SIGCOMM Computer Communication Review*, 31(2):20–41, April 2001.

[CHB+01] D. Culler, J. Hill, P. Buonadonna, R. Szewczyk, and A. Woo. A network-centric approach to embedded software for tiny devices. In *First Int'l Workshop on Embedded Software (EMSOFT 2001)*, volume 2211 of *Lecture Notes in Computer Science*, pages 114–130. Springer, Berlin, October 2001.

[CHH01] S. Capkun, M. Hamdi, and J.P. Hubaux. GPS-free positioning in mobile ad hoc networks. In R.H. Sprague, editor, *Proc. 34th Ann. Hawaii Int'l Conf. System Sciences (HICSS 2001)*, pages 10–19. IEEE CS Press, Los Alamitos, CA, January 2001.

[CHH02] S. Capkun, M. Hamdi, and J.P. Hubaux. GPS-free positioning in mobile ad hoc networks. *Cluster Computing*, 5(2):157–167, 2002.

[CKR+03] C. Chang, K. Kuusilinna, B. Richards, A. Chen, N. Chan, and R.W. Brodersen. Rapid design and analysis of communication systems using the BEE hardware emulation environment. In *Proc. 14th IEEE Int'l Workshop Rapid Systems Prototyping (RSP 2003)*, pages 148–154. IEEE, Piscataway, NJ, June 2003.

[CM04] D. Culler and H. Mulder. Smart sensors to network the world. *Scientific American*, pages 84–91, June 2004.

[CMY+02] A. Chen, R.R. Muntz, S. Yuen, I. Locher, S.I. Sung, and M.B. Srivastava. A support infrastructure for the smart kindergarten. *IEEE Pervasive Computing*, 1(2):49–57, April 2002.

[Com01] Committee on Networked Systems of Embedded Computers, Computer Science and Telecommunications Board, Division on Engineering and Physical Sciences, National Research Council, editor. *Embedded, Everywhere: A research agenda for networked systems of embedded computers*. National Academy Press, Washington, 2001.

[CS98] J.J. Caffery and G.L. Stüber. Overview of radiolocation in CDMA cellular systems. *IEEE Communications Magazine*, 36(4):38–45, April 1998.

[CS02] D. Cavin and Y. Sasson. On the accuracy of MANET simulators. In *ACM Workshop Principles Of Mobile Computing (POMC 02)*, pages 38–43. ACM Press, New York, October 2002.

[DBM05] M. Dyer, J. Beutel, and L. Meier. Deployment support for wireless sensor networks. In *Proc. 4th GI/ITG KuVS Fachgespräch Drahtlose Sensornetze*, Technical Report 418, pages 25–28. Dept. Computer Science, ETH Zürich, Switzerland, March 2005.

[dBvKMOS00] M. de Berg, van Kreveld M., M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer, Berlin, 2 edition, 2000.

[Del02] K.A. Delin. The sensor web: A macro-instrument for coordinated sensing. *Sensors*, 2(7):270–285, 2002. Special Issue: Networked Sensors and Wireless Sensor Platforms.

[DGA+05] P. Dutta, M. Grimmer, A. Arora, S. Bibyk, and D. Culler. Design of a wireless sensor network platform for detecting rare, random, and ephemeral events. In *Proc. 4th Int'l Conf. Information Processing in Sensor Networks (IPSN '05)*, pages 497–502. IEEE, Piscataway, NJ, April 2005.

[DGV04] A. Dunkels, B. Grönvall, and T Voigt. Contiki – a lightweight and flexible operating system for tiny networked sensors. In *Proc. 1nd IEEE Workshop on Embedded Networked Sensors (EmNetS-I)*, pages 455 – 462. IEEE, Piscataway, NJ, November 2004.

[DJJ+05] K.A. Delin, S.P. Jackson, D.W. Johnson, S.C. Burleigh, R.R. Woodrow, J.M. McAuley, J.M. Dohm, F. Ip, T.P.A. Ferré, D.F. Rucker, and V.R. Baker. Environmental studies with the sensor web: Principles and practice. *Sensors*, 5(2):103–117, 2005. Special Issue, Sensors for Environmental Monitoring.

[DKBZ05] C. Decker, A. Krohn, M. Beigl, and T. Zimmer. The particle computer system. In *Proc. 4th Int'l Conf. Information Processing in Sensor Networks (IPSN '05)*, pages 443–448. IEEE, Piscataway, NJ, April 2005.

[DMS98] C. Drane, M. Macnaughtan, and C. Scott. Positioning GSM telephones. *IEEE Communications Magazine*, 36(4):46–54, 1998.

[DNHKK04] C. Decker, S. Nguissi, J. Haller, and R. Kilian-Kehr. Proximity as a security property in a mobile enterprise application context. In *Proc. 37th Ann. Hawaii Int'l Conf. System Sciences (HICSS 2005)*, volume 07, page 70189b. IEEE CS Press, Los Alamitos, CA, 2004.

[DPEG01] L. Doherty, K.S.J. Pister, and L. El-Ghaoui. Convex position estimation in wireless sensor networks. In *Proc. 20th Ann. Joint IEEE Conf. Computer Communiucation Soc. (Infocom 2001)*, volume 3, pages 1655–1663. IEEE, Piscataway, NJ, 2001.

[dSJSA+01] J.L. da Silva Jr., J. Shamberger, M.J. Ammer, C. Guo, S. Li, R. Shah, T. Tuan, M. Sheets, J.M. Rabaey, B. Nikolic, A. Sangiovanni-Vincentelli, and P. Wright. Design methodology for PicoRadio networks. In *Proc. Conf. Design, Automation and Test in Europe (DATE 2001)*, pages 314–323. IEEE, Piscataway, NJ, March 2001.

[DWBP01] L. Doherty, B.A. Warneke, B. Boser, and K.S.J. Pister. Energy and performance considerations for smart dust. *Int'l J. Parallel and Distributed Systems and Networks*, 4(3):121–133, 2001.

[Dyd04] M.B. Dydensborg. *Connection Oriented Sensor Networks*. PhD thesis, Department of Computer Science, University of Copenhagen, Denmark, December 2004.

[EBB+03] J. Elson, S. Bien, N. Busek, V. Bychkovskiy, A. Cerpa, D. Ganesan, L. Girod, B. Greenstein, T. Schoellhammer, T. Stathopoulos, and D. Estrin. EmStar: An environment for developing wireless embedded systems software. Technical Report 0009, Center for Embedded Networked Sensing (CENS), Univ. of California, Los Angeles, CA, March 2003.

[Ebe03] H. Eberle. A radio network for monitoring and diagnosing computer systems. *IEEE Micro*, 23(1):60–65, January 2003.

[ECPS02] D. Estrin, D. Culler, K. Pister, and G. Sukhatme. Connecting the physical world with pervasive networks. *IEEE Pervasive Computing*, 1(1):59–69, 2002.

[EEHDP04] C.C. Enz, A. El-Hoiydi, J.D. Decotignie, and V. Peiris. WiseNET: An ultralow-power wireless sensor network solution. *IEEE Computer*, 37(8):62–70, August 2004.

[EGHK99] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar. Next century challenges: Scalable coordination in sensor networks. In *Proc. 5th ACM/IEEE Ann. Int'l Conf. Mobile Computing and Networking (MobiCom '99)*, pages 263–270. ACM Press, New York, August 1999.

[EHD04] A. El-Hoiydi and J.D. Decotignie. WiseMAC: An ultra low power MAC protocol for multi-hop wireless sensor networks. In S. Nikoletseas and J.D.P. Rolim, editors, *Proc. 1st Int'l Workshop Algorithmic Aspects of Wireless Sensor Networks (ALGOSENSORS 2004)*, volume 3121 of *Lecture Notes in Computer Science*, pages 18–31. Springer, Berlin, June 2004.

[Els03] J.E. Elson. *Time Synchronization in Wireless Sensor Networks*. PhD thesis, Dept. Computer Sciences, Univ. of California, Los Angeles, CA, 2003.

[ER02] A. Erni and S. Reichmuth. Bluetooth Anbindung für Lego Mindstorms. Term thesis, Computer Engineering and Networks Lab, ETH Zürich, Switzerland, July 2002.

[FC02] C.C. Foo and K.C. Chua. BlueRings - Bluetooth scatternets with ring structures. In *Proc. IASTED Int'l Conf. on Wireless and Optical Communication (WOC 2002)*. ACTA Press, Calgary, Canada, July 2002.

[FKO03] T. Fuhrmann, M. Klein, and M. Odendahl. The BlueWand as interface for ubiquitous and wearable computing environments. In *Proc. 5th European Personal Mobile Communications Conf. (EPMCC '03)*, pages 91–95. IEE, London, April 2003.

[FP05a] E. Ferro and F. Potorti. Bluetooth and Wi-Fi wireless protocols: A survey and a comparison. *IEEE Wireless Communications*, 12(1):12–26, 2005.

[FP05b] H.J. Freeland and P.F.Cummins. Argo: A new tool for environmental monitoring and assessment of the world's ocean, an example from the NE pacific. *Progress in Oceanography*, 64(1):31–44, 2005.

[FR05] C. Frank and K. Römer. Algorithms for generic role assignment in wireless sensor networks. In *Proc. 3rd ACM Conf. Embedded Networked Sensor Systems (SenSys 2005)*, page to appear, November 2005.

[Fre03] U. Frey. Topology and position estimation in Bluetooth ad hoc networks. Master's thesis, Computer Engineering and Networks Lab, ETH Zürich, Switzerland, March 2003.

[FRL05a] C.L. Fok, G.C. Roman, and C. Lu. Mobile agent middleware for sensor networks: An application case study. In *Proc. 4th Int'l Conf. Information Processing in Sensor Networks (IPSN '05)*. ACM Press, New York, April 2005.

[FRL05b] C.L. Fok, G.C. Roman, and C. Lu. Rapid development and flexible deployment of adaptive wireless sensor network applications. In *Proc. 25rd Int'l Conf. Distributed Computing Systems (ICDCS 2005)*. IEEE, Piscataway, NJ, 2005.

[GEC+04] L. Girod, J. Elson, A. Cerpa, T. Stathapopoulos, N. Ramananthan, and D. Estrin. EmStar: A software environment for developing and deploying wireless sensor networks. In *Proc. USENIX 2004 Annual Tech. Conf.*, pages 283–296, June 2004.

[GK00] P. Gupta and P.R. Kumar. The capacity of wireless networks. *IEEE Transactions on Information Theory*, 46(2):388–404, March 2000.

[GKSB04] H. Gellersen, G. Kortuem, A. Schmidt, and M. Beigl. Physical prototyping with Smart-Its. *IEEE Pervasive Computing*, 3(3):74–82, July 2004.

[GKW+02] D. Ganesan, B. Krishnamachari, A. Woo, D. Culler, D. Estrin, and S. Wicker-et. Complex behavior at scale: An experimental study of low-power wireless sensor networks. Technical Report CSD-TR 02-0013, Dept. Computer Sciences, Univ. of California, Los Angeles, CA, February 2002.

[GLvB+03] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler. The nesC language: A holistic approach to networked embedded systems. In *Proc.*

ACM SIGPLAN 2003 Conf. Programming Language Design and Implementation (PLDI 2003), pages 1–11. ACM Press, New York, June 2003.

[Gou05] J. Gould. From swallow floats to Argo – the development of neutrally buoyant floats. *Deep-Sea Research II*, 52(3-4):529–543, 2005.

[GRSV03] R. Guérin, J. Rank, S. Sarkar, and E. Vergetis. Forming connected topologies in Bluetooth ad-hoc networks – an algorithmic perspective. In *Providing Quality of Service in Heterogeneous Environments, Proc. 18th Int'l Teletraffic Congress (ITC-18)*, Teletraffic Science and Engineering. Elsevier B.V., Amsterdam, The Netherlands, September 2003.

[Haa00] J.C. Haartsen. The Bluetooth radio system. *IEEE Personal Communications*, 7(1):28–36, February 2000.

[Haa01] J.C. Haartsen. Bluetooth – ad hoc networking in an uncoordianted environment. In *Proc. 2001 Int'l Conf. Acoustics, Speech, and Signal Processing (ICASSP 2001)*, volume 4, pages 2029–2032. IEEE, Piscataway, NJ, May 2001.

[HBAB04] B. Hemingway, W. Brunette, T. Anderl, and G. Borriello. The Flock: Mote sensors sing in undergraduate curriculum. *IEEE Computer*, 37(8):72–78, August 2004.

[HBE+01] J. Heidemann, N. Bulusu, J. Elson, C. Intanagonwiwat, K.C. Lan, Y. Xu, W. Ye, D. Estrin, and R. Govindan. Effects of detail in wireless network simulation. In *Proc. SCS Multiconference Distributed Simulation 2001*, pages 3–11. USC/Information Sciences Institute, Society for Computer Simulation, Los Angeles, CA, January 2001.

[HBFZ04] H.J. Hof, E.O. Blass, T. Fuhrmann, and M. Zitterbart. Design of a secure distributed service directory for wireless sensornetworks. In *Proc. 1st European Workshop on Sensor Networks (EWSN 2004)*, volume 2920 of *Lecture Notes in Computer Science*, pages 276–290. Springer, Berlin, January 2004.

[HC02] J.L. Hill and D. Culler. Mica: A wireless platform for deeply embedded networks. *IEEE Micro*, 22(6):12–24, November 2002.

[HC04] J.W. Hui and D. Culler. The dynamic behavior of a data dissemination protocol for network programming at scale. In *Proc. 2nd ACM Conf. Embedded Networked Sensor Systems (SenSys 2004)*, pages 81–94. ACM Press, New York, November 2004.

[HCB02] W. Heinzelman, A. P. Chandrakasan, and H. Balakrishnan. An application-specific protocol architecture for wireless microsensor networks. *IEEE Transactions on Wireless Communications*, 1(4):660–670, October 2002.

[Hei00] W. Heinzelman. *Application-Specific Protocol Architectures for Wireless Networks*. PhD thesis, Massachusetts Institute of Technology, Boston, MA, 2000.

[HGK+04] L.E. Holmquist, H.W. Gellersen, G. Kortuem, A. Schmidt, M. Strohbach, S. Antifakos, F. Michahelles, B. Schiele, and M. Beigl anb R. Mazé. Building intelligent environments with Smart-Its. *IEEE Computer Graphics and Applications*, pages 56–64, January 2004.

[HGLBV01] J.P. Hubaux, T. Gross, J.Y. Le Boudec, and M. Vetterli. Toward self-organized mobile ad hoc networks: The Terminodes project. *IEEE Communications Magazine*, 39(1):118–124, January 2001.

[HH89] P. Horowitz and W. Hill. *The Art of Electronics*. Cambridge University Press, Cambridge, UK, 2nd edition, 1989.

[HHKK04] J.L. Hill, M. Horton, R. Kling, and L. Krishnamurthy. Wireless sensor networks: The platforms enabling wireless sensor networks. *Communications of the ACM*, 47(6):41–46, June 2004.

[HHS+99] A. Harter, A. Hopper, P. Steggles, A. Ward, and P. Webster. The anatomy of a context-aware application. In *Proc. 5th ACM/IEEE Ann. Int'l Conf. Mobile Computing and Networking (MobiCom '99)*, pages 59–68, August 1999.

[Hil04] J.L. Hill. *System Architecture for Wireless Sensor Networks*. PhD thesis, UC Berkeley, 2004.

[HKS+04] T. He, S. Krishnamurthy, J.A. Stankovic, T. Abdelzaher, L. Luo, R. Stoleru, T. Yan, L. Gu, J. Hui, and B. Krogh. Energy-efficient surveillance system using wireless sensor networks. In *Proc. 2nd ACM/USENIX Conf. Mobile Systems, Applications, and Services (MobiSys 2004)*, pages 270–283. ACM Press, New York, June 2004.

[HLBG+00] J.P. Hubaux, J.Y. Le Boudec, S. Giordano, M. Hamdi, L. Blazevic, L. Buttyan, and M. Vojnovic. Towards mobile ad hoc wans: Terminodes. In *Proc. 2000 IEEE Wireless Communications and Networking Conf. (WCNC 2000)*, volume 3, pages 1052–1059. IEEE, Piscataway, NJ, September 2000.

[HLBGH99] J.P. Hubaux, J.Y. Le Boudec, S. Giordano, and M. Hamdi. The Terminode project: Towards mobile ad hoc WANs. In *Proc. 1999 IEEE Int'l Workshop Mobile Multimedia Communications (MoMuC 99)*, pages 124–128. IEEE, Piscataway, NJ, November 1999.

[HM00] J.C. Haartsen and S. Mattisson. Bluetooth – a new low-power radio interface providing short-range connectivity. *Proceedings of the IEEE*, 88(10):1651–1661, October 2000.

[HPH+05] V. Handziski, J. Polastre, J.H. Hauer, C. Sharp, A. Wolisz, and D. Culler. Flexible hardware abstraction for wireless sensor networks. In *Proc. 2nd European Workshop on Sensor Networks (EWSN 2005)*, pages 145–157. IEEE CS Press, Los Alamitos, CA, January 2005.

[HSW+00] J.L. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. In *Proc. 9th Int'l Conf. Architectural Support Programming Languages and Operating Systems (ASPLOS-IX)*, pages 93–104. ACM Press, New York, November 2000.

[HSWC00] W.R. Heinzelman, A. Sinha, A. Wang, and A.P. Chandrakasan. Energy-scalable algorithms and protocols for wireless microsensor networks. In *Proc. 2000 Int'l Conf. Acoustics, Speech, and Signal Processing (ICASSP 2000)*, volume 6, pages 3722–3725. IEEE, Piscataway, NJ, 2000.

[HWH03] R.K. Harle, A. Ward, and A. Hopper. Single reflection spatial voting. In *Proc. 1st ACM/USENIX Conf. Mobile Systems, Applications, and Services (MobiSys 2003)*, pages 1–15. ACM Press, New York, May 2003.

[ICP+99] A. Iwata, C.C. Chiang, G. Pei, M. Gerla, and T.W. Chen. Scalable routing strategies for ad hoc wireless networks. *IEEE Journal on Selected Areas in Communication*, 17(8):1369–1379, August 1999. Special Issue on Ad Hoc Networks.

[IGE00] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *Proc. 6th ACM/IEEE Ann. Int'l Conf. Mobile Computing and Networking (MobiCom 2000)*, pages 56–67. ACM Press, New York, August 2000.

[IK96] T. Imielinski and H. Korth, editors. *Mobile Computing*. Kluwer Academic Publishers, Norwell, MA, 1996.

[JKKG01] P. Johansson, M. Kazantzidis, R. Kapoor, and M. Gerla. Bluetooth: An enabler for personal area networking. *IEEE Network*, 1(5):28–37, September 2001.

[JLT03] H. Junker, P. Lukowicz, and G. Tröster. PadNET: Wearable physical activity detection network. In *Proc. Int. Symp. Wearable Computers (ISWC '03)*, pages 244–245. IEEE, Piscataway, NJ, October 2003.

[JM96] D.B. Johnson and D.A. Maltz. Dynamic source routing in ad hoc wireless networks. In T. Imielinski and H. Korth, editors, *Mobile Computing*. Kluwer Academic Publishers, Norwell, MA, 1996.

[JOW+02] P. Juang, H. Oki, Y. Wang, M. Martonosi, L.S. Peh, and D. Rubenstein. Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiences with ZebraNet. In *Proc. 10th Int'l Conf. Architectural Support*

Programming Languages and Operating Systems (ASPLOS-X), pages 96–107. ACM Press, New York, October 2002.

[JPC05] X. Jiang, J. Polastre, and D. Culler. Perpetual environmentally powered sensor networks. In *Proc. 4th Int'l Conf. Information Processing in Sensor Networks (IPSN '05)*, pages 463–468. IEEE, Piscataway, NJ, April 2005.

[KAH+04] R. Kling, R. Adler, J. Huang, V. Hummel, and L. Nachman. Intel mote: Using Bluetooth in sensor networks. In *Proc. 2nd ACM Conf. Embedded Networked Sensor Systems (SenSys 2004)*, page 318. ACM Press, New York, November 2004.

[Kas05] O. Kasten. *A State-Based Programming Framework for Wireless Sensor Networks*. PhD thesis, Dept. Computer Science, ETH Zürich, Switzerland, 2005.

[KB95] R.H. Katz and E.A. Brewer. The case for wireless overlay networks. In T. Imielinski and H. Korth, editors, *Mobile Computing*, pages 621–650. Kluwer Academic Publishers, Norwell, MA, 1995.

[KBA+96] R.H. Katz, E.A. Brewer, E. Amir, H. Balakrishnan, A. Fox, S. Gribble, T. Hodes, D. Jiang, N. Giao Thanh, V. Padmanabhan, and M. Stemm. The bay area research wireless access network (BARWAN). In *Proc. 41st IEEE Comp. Soc. Int'l Conf.: Technologies for the Information Superhighway (COMPCON 1996)*, pages 15–20. IEEE CS Press, Los Alamitos, CA, February 1996.

[KCA+03] K. Kuusilinna, C. Chang, M.J. Ammer, B. Richards, and R.W. Brodersen. Designing BEE: A hardware emulation engine for signal processing in low-power wireless applications. *EURASIP Journal on Applied Signal Processing*, 2003(6):502–523, 2003. Special issue on Rapid Prototyping of DSP Systems.

[KCB+03] K. Kuusilinna, C. Chang, H.M. Bluethgen, W.R. Davis, B. Richards, B. Nikolic, and R.W. Brodersen. *Winning the SoC Revolution, Experiences in Real Design*, chapter Real-time System-on-a-Chip Emulation: Emulation Driven System Design with Direct Mapped Virtual Components, pages 229–253. Kluwer Academic Publishers, Norwell, MA, 2003.

[KKP99] J.M. Kahn, R.H. Katz, and K.S.J. Pister. Next century challenges: Mobile networking for smart dust. In *Proc. 5th ACM/IEEE Ann. Int'l Conf. Mobile Computing and Networking (MobiCom '99)*, pages 271–278. ACM Press, New York, August 1999.

[KL01] O. Kasten and M. Langheinrich. First experiences with Bluetooth in the Smart-It's distributed sensor network. In *Workshop on Ubiquitous Computing and Communication, Int'l Conf. Parallel Architectures and Compilation Techniques (PACT 2001)*, September 2001.

[KMW04] F. Kuhn, T. Moscibroda, and R. Wattenhofer. Initializing newly deployed ad hoc and sensor networks. In *Proc. 10th ACM/IEEE Ann. Int'l Conf. Mobile Computing and Networking (MobiCom 2004)*, pages 260–274. ACM Press, New York, 2004.

[KNE03] D. Kotz, C. Newport, and C. Elliott. The mistaken axioms of wireless-network research. Technical Report TR2003-467, Dartmouth College Computer Science, July 2003.

[KNG+04] D. Kotz, C. Newport, R.S. Gray, J. Liu, Y. Yuan, and C. Elliott. Experimental evaluation of wireless simulation assumptions. In *Int'l Workshop Modeling Analysis and Simulation of Wireless and Mobile Systems (MSWiM 04)*, pages 78–82. ACM Press, New York, October 2004.

[KR05] O. Kasten and K. Römer. Beyond event handlers: Programming wireless sensors with attributed state machines. In *Proc. 4th Int'l Conf. Information Processing in Sensor Networks (IPSN '05)*, pages 45–52. IEEE, Piscataway, NJ, April 2005.

[KTZ05] S. Künzli, L. Thiele, and E. Zitzler. A modular design space exploration framework for embedded systems. *IEE Proc. Computers & Digital Techniques*, 152(2):183–192, March 2005.

[KV98] Y.B. Ko and N.H. Vaidya. Location-aided routing (LAR) in mobile ad hoc networks. In *Proc. 4th ACM/IEEE Ann. Int'l Conf. Mobile Computing and Networking (MobiCom '98)*, pages 66–75. ACM Press, New York, August 1998.

[KWZ03] F. Kuhn, R. Wattenhofer, and A. Zollinger. Worst-case optimal and average-case efficient geometric ad hoc routing. In *Proc. 4th ACM Int'l Symp. Mobile Ad Hoc Networking and Computing (MobiHoc 2003)*, pages 267–278. ACM Press, New York, June 2003.

[LC02] P. Levis and D. Culler. Maté: A tiny virtual machine for sensor networks. In *Proc. 10th Int'l Conf. Architectural Support Programming Languages and Operating Systems (ASPLOS-X)*, pages 85–95. ACM Press, New York, October 2002.

[LC04] J. Liu and P.H. Chou. Distributed embedded systems for low power: A case study. In *Proc. 18th Int'l Parallel and Distributed Processing Symposium (IPDPS 2004)*, pages 26–30. IEEE, Piscataway, NJ, April 2004.

[LCS+00] Y. Lu, E. Chung, T. Simunic, L. Benini, and G. De Micheli. Quantitative comparison of power management algorithms. In *Proc. Conf. Design, Automation and Test in Europe (DATE 2002)*, pages 20–26. IEEE, Piscataway, NJ, March 2000.

[LDB03] M. Leopold, M.B. Dydensborg, and P. Bonnet. Bluetooth and sensor networks: A reality check. In *Proc. 1st ACM Conf. Embedded Networked Sensor Systems (SenSys 2003)*, pages 103–113. ACM Press, New York, November 2003.

[Leo04] M. Leopold. Power estimation using the Hogthrob prototype platform. Master's thesis, Department of Computer Science, University of Copenhagen, Denmark, 2004.

[LLN+02] H. Lundgren, D. Lundberg, J. Nielsen, E. Nordstrom, and C. Tschudin. A large-scale testbed for reproducible ad hoc protocol evaluations. In *Proc. 2002 IEEE Wireless Communications and Networking Conf. (WCNC 2002)*, volume 1, pages 412–418. IEEE, Piscataway, NJ, March 2002.

[LLWC03] P. Levis, N. Lee, M. Welsh, and D. Culler. TOSSIM: Accurate and scalable simulation of entire TinyOS applications. In *Proc. 1st ACM Conf. Embedded Networked Sensor Systems (SenSys 2003)*, pages 126–137. ACM Press, New York, November 2003.

[LMG+04] P. Levis, S. Madden, D. Gay, J. Polastre, R. Szewczyk, A. Woo, Brewer E., and D. Culler. The emergence of networking abstractions and techniques in TinyOS. In *Proc. First Symp. Networked Systems Design and Implementation (NSDI '04)*, pages 1–14. ACM Press, New York, March 2004.

[LMP+05] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler. *Ambient Intelligence*, chapter TinyOS: An Operating System for Sensor Networks, pages 115–148. Springer, Berlin, 2005.

[LMS03] C. Law, A.K. Mehta, and K.Y. Siu. A new Bluetooth scatternet formation protocol. *ACM/Kluwer Mobile Networks and Applications*, 8(5):485–498, October 2003.

[Log92] T. Logsdon. *The Navstar Global Positioning System*. Van Nostrand Reinhold, New York, 1992.

[LPCS04] P. Levis, N. Patel, D. Culler, and S. Shenker. Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In *Proc. First Symp. Networked Systems Design and Implementation (NSDI '04)*, pages 15–28. ACM Press, New York, March 2004.

[LSBP02] J. Lifton, D. Seetharam, M. Broxton, and J. Paradiso. Pushpin computing system overview: A platform for distributed, embedded, ubiquitous sensor networks. In F. Mattern and M. Naghshineh, editors, *Proc. First Int'l Conf. Pervasive Computing (Pervasive 2002)*, volume 2414 of *Lecture Notes in Computer Science*, pages 139–151. Springer, Berlin, August 2002.

[LSS03] S. Li, S. H. Son, and J. A. Stankovic. Event detection services using data service middleware in distributed sensor networks. In F. Zhao and L. Guibas, editors, *Proc. 2nd Int'l Conf. Information Processing in Sensor Networks (IPSN '03)*, volume 2634 of *Lecture Notes in Computer Science*. Springer, Berlin, April 2003.

[LSZM04] T. Liu, C.M. Sadler, P. Zhang, and M. Martonosi. Energy-efficient surveillance system using wireless sensor networks. In *Proc. 2nd ACM/USENIX Conf. Mobile Systems, Applications, and Services (MobiSys 2004)*, pages 256–269. ACM Press, New York, June 2004.

[LTC03] T.Y. Lin, Y.C. Tseng, and K.M. Chang. Formation, routing, and maintenance protocols for the Bluering scatternet of Bluetooths. In *Proc. 36th Ann. Hawaii Int'l Conf. System Sciences (HICSS 2003)*, page 10pp. IEEE CS Press, Los Alamitos, CA, January 2003.

[LWG05] O. Landsiedel, K. Wehrle, and S. Götz. Accurate prediction of power consumption in sensor networks. In *Proc. 2nd IEEE Workshop on Embedded Networked Sensors (EmNetS-II)*, page to appear. IEEE, Piscataway, NJ, May 2005.

[Mar05] K. Martin. Adaptive XTC on BTnodes. Master's thesis, Computer Engineering and Networks Lab, ETH Zürich, Switzerland, May 2005.

[MC03] R. Min and A. Chandrakasan. Top five myths about the energy consumption of wireless communication. *Mobile Computing and Communications Review*, 7(1):65–67, January 2003.

[MCB+02] R. Min, S. Cho, M. Bhardwaj, E. Shih, A. Wang, and A.P. Chandrakasan. Power-aware wireless microsensor networks. In M. Pedram and J.M. Rabaey, editors, *Power Aware Design Methodologies*, pages 335–372. Kluwer Academic Publishers, Norwell, MA, 2002.

[MCP+02] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson. Wireless sensor networks for habitat monitoring. In *Proc. 1st ACM Int'l Workshop Wireless Sensor Networks and Applications (WSNA 2002)*, pages 88–97. ACM Press, New York, September 2002.

[MFHH02] S.R. Madden, M.J. Franklin, J.M. Hellerstein, and W. Hong. TAG: A tiny aggregation service for ad-hoc sensor networks. In *Proc. 5th Symp. Operating Systems Design and Implementation (OSDI 2002)*, pages 131–146. ACM Press, New York, December 2002.

[MFT05] L. Meier, P. Ferrari, and L. Thiele. Energy-efficient Bluetooth networks. Technical Report 204, Computer Engineering and Networks Lab, ETH Zürich, Switzerland, January 2005.

[Mic04] F. Michahelles. *Innovative Application Development for Ubiquitous and Wearable Computing*. PhD thesis, Dept. Computer Science, ETH Zürich, Switzerland, 2004.

[MS02] F. Michahelles and B. Schiele. Better rescue through sensors. In *Proc. 4th Int'l Conf. Ubiquitous Computing (UbiComp 2002)*, September 2002.

[MSLS04] M. Maróti, G. Simon, Á. Lédeczi, and J. Sztipanovits. Shooter localization in urban terrain. *IEEE Computer*, 37(8):60–61, August 2004.

[MT02] A. Merkle and A. Terzis. *Digitale Funkkommunikation mit Bluetooth*. Franzis' Verlag, Poing, Germany, 2002.

[MW05] F. Michel and P. Wüger. Angewandte Uhrensynchronisation auf BTnodes. Term thesis, Computer Engineering and Networks Lab, ETH Zürich, Switzerland, February 2005.

[MWF02] P. Murphy, E. Welsh, and J.P. Frantz. Using Bluetooth for short-term ad hoc connections between moving vehicles: a feasibility study. In *Proc. 55th IEEE Semiannual Vehicular Technology Conference (VTC Spring 2002)*, volume 1, pages 414–418. IEEE, Piscataway, NJ, May 2002.

[NBD05a] L. Negri, J. Beutel, and M. Dyer. The power consumption of Bluetooth scatternets. page submitted, 2005.

[NBD05b] L. Negri, J. Beutel, and M. Dyer. The power consumption of Bluetooth scatternets. Technical Report 220, Computer Engineering and Networks Lab, ETH Zürich, Switzerland, May 2005.

[NKA+05] L. Nachman, R. Kling, R. Adler, J. Huang, and V. Hummel. The Intel mote platform: A Bluetooth-based sensor network for industrial monitoring. In *Proc. 4th Int'l Conf. Information Processing in Sensor Networks (IPSN '05)*, pages 437–442. IEEE, Piscataway, NJ, April 2005.

[NLYP03] L.M. Ni, Y. Liu, Lau Y.C., and A.P. Patil. LANDMARC: Indoor location sensing using active RFID. In *Proc. 1st IEEE Int'l Conf. Pervasive Computing and Communications (PerCom 2003)*, pages 407–415. IEEE CS Press, Los Alamitos, CA, March 2003.

[NN01] D. Niculescu and B. Nath. Ad hoc positioning system (APS). In *Proc. IEEE Global Telecommunications Conf. (GLOBECOM 2001)*, pages 2926–2931. IEEE, Piscataway, NJ, November 2001.

[NWS03] K. Naik, D.S.L. Wei, and Y.T. Su. Packet interference in a heterogeneous cluster of Bluetooth piconets. In *Proc. 58th IEEE Semiannual Vehicular Technology Conference (VTC Fall 2003)*, volume 1, pages 582–586. IEEE, Piscataway, NJ, October 2003.

[PB94] C.E. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. *SIGCOMM Computer Communications Review*, 24(4):234–244, 1994.

[PBC02] C. Petrioli, S. Basagni, and I. Chlamtac. BlueMesh: Degree-constrained multihop scatternet formation for Bluetooth networks. *ACM/Kluwer Mobile Networks and Applications*, 9(1):33–47, February 2002.

[PBC03] C. Petrioli, S. Basagni, and I. Chlamtac. Configuring BlueStars: Multi-hop scatternet formation in Bluetooth networks. *IEEE Transactions on Computers*, 52(6):779–790, Jun. 2003.

[PC04] C. Park and P.H. Chou. Power utility maximization for multiple-supply systems by a load-matching switch. In *Proc. Int'l Symp. Low Power Electronics and Design (ISLPED 2004)*, pages 168–173. ACM Press, New York, August 2004.

[PCB00] N.B. Priyantha, A. Chakraborty, and H. Balakrishnan. The Cricket location-support system. In *Proc. 6th ACM/IEEE Ann. Int'l Conf. Mobile Computing and Networking (MobiCom 2000)*, pages 32–43. ACM Press, New York, 2000.

[Per01] C.E. Perkins, editor. *Ad Hoc Networking*. Addison-Wesley, Boston, MA, 2001.

[PEW+02] C. Plessl, R. Enzler, H. Walder, J. Beutel, M. Platzner, and L. Thiele. Reconfigurable hardware in wearable computing nodes. In *Proc. Int. Symp. Wearable Computers (ISWC '02)*, pages 215–222. IEEE, Piscataway, NJ, October 2002.

[PEW+03] C. Plessl, R. Enzler, H. Walder, J. Beutel, M. Platzner, L. Thiele, and G. Tröster. The case for reconfigurable hardware in wearable computing. *Personal and Ubiquitous Computing*, 7(5):299–308, October 2003.

[PH00] D.D. Perkins and H. Hughes. A performance comparison of routing protocols for mobile ad hoc networks. In M.S. Obaidat, F. Davoli, and M.A. Marsan, editors, *Proceedings of the 2000 Symposium on Performance Evaluation of Computer and Telecommunication Systems*, pages 480–487, 2000.

[PHC04] J. Polastre, J. Hill, and D. Culler. Versatile low power media access for wireless sensor networks. In *Proc. 2nd ACM Conf. Embedded Networked Sensor Systems (SenSys 2004)*, pages 95–107. ACM Press, New York, 2004.

[PK00] G.J. Pottie and W.J. Kaiser. Wireless integrated network sensors. *Communications of the ACM*, 43(5):51–58, May 2000.

[PLC05] C. Park, J. Liu, and P.H. Chou. Eco: An ultra-compact low-power wireless sensor node for real-time motion monitoring. In *Proc. 4th Int'l Conf. Information Processing in Sensor Networks (IPSN '05)*, pages 398–403. IEEE, Piscataway, NJ, April 2005.

[PSC05] J. Polastre, R. Szewczyk, and D. Culler. Telos: Enabling ultra-low power wireless research. In *Proc. 4th Int'l Conf. Information Processing in Sensor Networks (IPSN '05)*, pages 364–369. IEEE, Piscataway, NJ, April 2005.

[RAdSJ+00] J.M. Rabaey, M.J. Ammer, J.L. da Silva Jr., D. Patel, and S. Roundy. PicoRadio supports ad hoc ultra-low power wireless networking. *IEEE Computer*, 33(7):42–48, July 2000.

[RAK+02] J.M. Rabaey, J. Ammer, T. Karalar, Li. S., B. Otis, M. Sheets, and T. Tuan. Picoradios for wireless sensor networks: the next challenge in ultra-low-power design. In *Proc. Int'l Solid-State Circuits Conf. (ISSCC 2002)*, volume 1, pages 200–201. IEEE, Piscataway, NJ, February 2002.

[RBM05] K. Römer, P. Blum, and L. Meier. *Sensor Networks*, chapter Time Synchronization and Calibration in Wireless Sensor Networks. John Wiley & Sons, New York, July 2005.

[RFMB04] K. Römer, C. Frank, P.J. Marrón, and C. Becker. Generic role assigment for wireless sensor networks. In *Proc. 11th ACM SIGOPS European Workshop*, pages 7–12. ACM Press, New York, September 2004.

[RM04] K. Römer and F. Mattern. The design space of wireless sensor networks. *IEEE Wireless Communications*, 11(6):54–61, December 2004.

[ROC+03] S. Roundy, B. Otis, Y.H. Chee, J.M. Rabaey, and P. Wrigh. A 1.9GHz RF transmit beacon using environmentally scavenged energy. In *Proc. Int'l Symp. Low Power Electronics and Design (ISLPED 2003)*, August 2003. design contest.

[Röm03] K. Römer. The Lighthouse location system for smart dust. In *Proc. 1st ACM/USENIX Conf. Mobile Systems, Applications, and Services (MobiSys 2003)*, pages 15–30. ACM Press, New York, May 2003.

[Röm04a] K. Römer. Determination of time and location in large-scale dynamic networks of tiny sensors. In A. Ferscha, H. Hoertner, and G. Kotsis, editors, *Advances in Pervasive Computing*, pages 125–132. Austrian Computer Society (OCG), April 2004.

[Röm04b] K. Römer. Tracking real-world phenomena with smart dust. In *Proc. 1st European Workshop on Sensor Networks (EWSN 2004)*, volume 2920 of *Lecture Notes in Computer Science*, pages 307–322. Springer, Berlin, January 2004.

[Röm05] K. Römer. *Time Synchronization and Localization in Sensor Networks*. PhD thesis, Dept. Computer Science, ETH Zürich, Switzerland, 2005.

[Ros04] P.E. Ross. 10 tech companies for the next 10 years. *IEEE Spectrum*, pages 32–38, November 2004.

[RPW+04] V. Raghunathan, T. Pering, R. Want, A. Nguyen, and P. Jensen. Experience with a low power wireless mobile computing platform. In *Proc. Int'l Symp. Low Power Electronics and Design (ISLPED 2004)*, pages 363–368. ACM Press, New York, August 2004.

[RR05] M. Ringwald and K. Römer. BitMAC: A deterministic, collision-free, and robust MAC protocol for sensor networks. In *Proc. 2nd European Workshop on Sensor Networks (EWSN 2005)*, pages 57–69. IEEE CS Press, Los Alamitos, CA, January 2005.

[RTVS03] H. Ritter, M. Tian, T. Voigt, and J. Schiller. A highly flexible testbed for studies of ad-hoc network behaviour. In *Proc. 28th Ann. IEEE Conf. Local Comp. Networks (LCN 2003)*, pages 746–752. IEEE, Piscataway, NJ, October 2003.

[RWR03] S. Roundy, P.K. Wright, and J.M. Rabaey. *Energy Scavenging for Wireless Sensor Networks with Special Focus on Vibrations*. Springer, Berlin, 2003.

[SB97] G. Strang and K. Borre. *Linear Algebra, Geodesy and GPS*. Wellesley-Cambridge Press, Wellesley, MA, 1997.

[SBP+04] G. Simon, G. Balogh, G. Pap, M. Maróti, B. Kusy, J. Sallai, Á. Lédeczi, A. Nádas, and K. Frampton. Sensor network-based countersniper system. In *Proc. 2nd ACM Conf. Embedded Networked Sensor Systems (SenSys 2004)*, pages 1–12. ACM Press, New York, November 2004.

[SBS02] E. Shih, P. Bahl, and M. Sinclair. Wake on wireless: An event driven energy saving strategy for battery operated devices. In *Proc. 6th ACM/IEEE Ann. Int'l Conf. Mobile Computing and Networking (MobiCom 2001)*, pages 160–171. ACM Press, New York, September 2002.

[Sch02] A. Schmidt. *Ubiquitous Computing – Computing in Context*. PhD thesis, Computing Department, Lancaster University, U.K., 2002.

[SDFV05] T. Schmid, H. Dubois-Ferrière, and M. Vetterli. SensorScope: Experiences with a wireless building monitoring sensor network. In *Proc. Workshop on Real-World Wireless Sensor Networks (REALWSN '05)*, page to appear, June 2005.

[SF03] F. Siegemund and C. Flörkemeier. Interaction in pervasive computing settings using Bluetooth-enabled active tags and passive RFID technology together with mobile phones. In *Proc. 1st IEEE Int'l Conf. Pervasive Computing and Communications (PerCom 2003)*, pages 378–387. IEEE CS Press, Los Alamitos, CA, March 2003.

[SFV04] F. Siegemund, C. Flörkemeier, and H. Vogt. The value of handhelds in smart environments. In *Proc. 17th Int'l Conf. on Architecture of Computing Systems - Organic and Pervasive Computing (ARCS '04)*, volume 2981 of *Lecture Notes in Computer Science*, pages 291–308. Springer, Berlin, January 2004.

[SHC+04] V. Shnayder, M. Hempstead, B. Chen, G. Werner-Allen, and M. Welsh. Simulating the power consumption of large-scale sensor network applications. In *Proc. 2nd ACM Conf. Embedded Networked Sensor Systems (SenSys 2004)*, pages 188–200. ACM Press, New York, November 2004.

[SHS01] A. Savvides, C.C. Han, and M.B. Strivastava. Dynamic fine-grained localization in ad hoc networks of sensors. In *Proc. 7th ACM/IEEE Ann. Int'l Conf. Mobile Computing and Networking (MobiCom 2001)*, pages 166–179. ACM Press, New York, July 2001.

[Sie02] F. Siegemund. Spontaneous interaction in ubiquitous computing settings using mobile phones and short text messages. In *Proc. Workshop Supporting Spontaneous Interaction in Ubiquitous Computing Settings (at Ubicomp 2002)*, September 2002.

[Sie04a] F. Siegemund. A context-aware communication platform for smart objects. In A. Ferscha and F. Mattern, editors, *Proc. Second Int'l Conf. Pervasive Computing (Pervasive 2004)*, number 3001 in Lecture Notes in Computer Science, pages 69–86. Springer, Berlin, April 2004.

[Sie04b] F. Siegemund. *Cooperating Smart Everyday Objects – Exploiting Heterogeneity and Pervasiveness in Smart Environments*. PhD thesis, Dept. Computer Science, ETH Zürich, Switzerland, 2004.

[SK99] M. Stemm and R.H. Katz. Vertical handoffs in wireless overlay networks. *ACM/Baltzer Mobile Networks and Applications (MONET))*, 3(4):319–334, January 1999.

[SK04a] F. Siegemund and P. Keller. Tuplespace-based collaboration for Bluetooth-enabled devices in smart environments. In *Proc. Informatik 2004, 34. Jahrestagung der Gesellschaft für Informatik, Workshop on Mobile Ad-Hoc Networks*, pages 133–137. Gesellschaft für Informatik, Bonn, Germany, September 2004.

[SK04b] F. Siegemund and T. Krauer. Integrating handhelds into environments of cooperating smart everyday objects. In *Proc. 2nd Europ. Symp. on Ambient Intelligence (EUSAI 2004)*, volume 3295 of *Lecture Notes in Computer Science*, pages 160–171. Springer, Berlin, January 2004.

[SMP+04] R. Szewczyk, A. Mainwaring, J. Polastre, J. Anderson, and D. Culler. An analysis of a large scale habitat monitoring application. In *Proc. 2nd ACM Conf. Embedded Networked Sensor Systems (SenSys 2004)*, pages 214–226. ACM Press, New York, November 2004.

[SOP+04] R. Szewczyk, E. Osterweil, J. Polastre, M. Hamilton, A. Mainwaring, and D. Estrin. Habitat monitoring with sensor networks. *Communications of the ACM*, 47(6):34–40, June 2004.

[Spm02] Special issue on collaborative signal and information processing in microsensor networks. *IEEE Signal Processing Magazine*, 19(2), March 2002.

[SPMC04] R. Szewczyk, J. Polastre, A. Mainwaring, and D. Culler. Lessons from a sensor network expedition. In *Proc. 1st European Workshop on Sensor Networks (EWSN 2004)*, volume 2920 of *Lecture Notes in Computer Science*, pages 307–322. Springer, Berlin, January 2004.

[SPS02] A. Savvides, H. Park, and M.B. Srivastava. The bits and flops of the n-hop multilateration primitive for node localization problems. In *Proc. 1st ACM Int'l Workshop Wireless Sensor Networks and Applications (WSNA 2002)*, pages 112–121. ACM Press, New York, September 2002.

[SR02] F. Siegemund and M. Rohs. Rendezvous layer protocols for Bluetooth-enabled smart devices. In H. Schmeck, T. Ungerer, and L. Wolf, editors, *Proc. 16th Int'l Conf. on Architecture of Computing Systems - Trends in Network and Pervasive Computing (ARCS 2002)*, volume 2299 of *Lecture Notes in Computer Science*, pages 256–273. Springer, Berlin, 2002.

[SR03] F. Siegemund and M. Rohs. Rendezvous layer protocols for Bluetooth-enabled smart devices. *Personal and Ubiquitous Computing*, 7(2):91–101, July 2003.

[SRB01] C. Savarese, J.M. Rabaey, and J. Beutel. Locationing in distributed ad hoc wireless sensor networks. In *Proc. 2001 Int'l Conf. Acoustics, Speech, and Signal Processing (ICASSP 2001)*, volume 4, pages 2037–2040. IEEE, Piscataway, NJ, May 2001.

[SRL02] C. Savarese, J.M. Rabaey, and K. Langendoen. Robust positioning algorithms for distributed ad hoc wireless sensor networks. In *Proc. 2002 USENIX Annual Technical Conference*, pages 317–327. USENIX Association, Berkeley, CA, June 2002.

[SS02] A. Savvides and M.B. Srivastava. A distributed computation platform for wireless embedded sensing. In *Proc. Int'l Conf. Computer Design: VLSI in Computers and Processors (ICCD 2002)*, pages 220–225. IEEE, Piscataway, NJ, August 2002.

[STGS02] C. Schurgers, V. Tsiatsis, S. Ganeriwal, and M. Srivastava. Topology management for sensor networks: Exploiting latency and density. In *Proc. 3rd ACM Int'l Symp. Mobile Ad Hoc Networking and Computing (MobiHoc 2002)*, pages 135–145. ACM Press, New York, June 2002.

[SWC02] A. Sinha, A. Wang, and A. P. Chandrakasan. Energy scalable system design. *IEEE Transactions on VLSI Systems*, 10(2):135–145, April 2002.

[TCGK02] L. Thiele, S. Chakraborty, M. Gries, and S. Künzli. A framework for evaluating design tradeoffs in packet processing architectures. In *Proc. 39th Design Automation Conf. (DAC 2002)*, pages 880–885. ACM Press, New York, June 2002.

[Ten00] D. Tennenhouse. Proactive computing. *Communications of the ACM*, 43(5):43–50, 2000.

[TLP05] B. Titzer, D.K. Lee, and J. Palsberg. Avrora: Scalable sensor network simulation with precise timing. In *Proc. 4th Int'l Conf. Information Processing in Sensor Networks (IPSN '05)*, pages 477–482. IEEE, Piscataway, NJ, April 2005.

[TLR01] T. Tuan, S. Li, and J.M. Rabaey. Reconfigurable platform design for wireless protocol processors. In *Proc. 2001 Int'l Conf. Acoustics, Speech, and Signal Processing (ICASSP 2001)*, volume 2, pages 893–896. IEEE, Piscataway, NJ, May 2001.

[TPB98] T.E. Truman, R. Pering, T. amd Doering, and R.W. Brodersen. The InfoPad multimedia terminal: A portable device for wireless information access. *IEEE Transactions on Computers*, 47(10):1037–1087, 1998.

[TSBW04] B. Thorstensen, T. Syversen, T.A. Bjornvold, and T. Walseth. Electronic shepherd: A low-cost, low-bandwidth, wireless network system. In *Proc. 2nd ACM/USENIX Conf. Mobile Systems, Applications, and Services (MobiSys 2004)*, pages 245–255. ACM Press, New York, June 2004.

[TSS+97] D.L. Tennenhouse, J.M. Smith, W.D. Sincoskie, D.J. Wetherall, and G.J Minden. A survey of active network research. *IEEE Communications Magazine*, 35(1):80–86, January 1997.

[TW96] D. Tennenhouse and D. Wetherall. Towards an active network architecture. In *Proc, 1996 SPIE Conf. Multimedia and Networking (MMCN '96)*, volume 2667, pages 2–16. SPIE, Bellingham, MA, March 1996.

[vDL03] T. van Dam and K. Langendoen. An adaptive energy efficient MAC protocol for wireless sensor networks. In *Proc. 1st ACM Conf. Embedded Networked Sensor Systems (SenSys 2003)*, pages 171–180. ACM Press, New York, November 2003.

[vGR05] J. van Greunen and J.M Rabaey. Location and timing synchronization services. In W. Weber, J.M. Rabaey, and E. Aarts, editors, *Ambient Intelligence*, pages 173–197. Springer, Berlin, 2005.

[VGSR05] E. Vergetis, R. Guérin, S. Sarkar, and J. Rank. Can Bluetooth succeed as a large-scale ad hoc networking technology? *IEEE Journal on Selected Areas in Communication*, 23(3):644–656, March 2005. Special Issue on Wireless Ad Hoc Networks.

[WAJR+05] G. Werner-Allen, J. Johnson, M. Ruiz, J. Lees, and M. Welsh. Monitoring volcanic eruptions with a wireless sensor network. In *Proc. 2nd European Workshop on Sensor Networks (EWSN 2005)*, pages 108–120. IEEE CS Press, Los Alamitos, CA, January 2005.

[WASW05] G. Werner-Allen, P. Swieskowski, and M. Welsh. MoteLab: A wireless sensor network testbed. In *Proc. 4th Int'l Conf. Information Processing in Sensor Networks (IPSN '05)*, pages 483–488. IEEE, Piscataway, NJ, April 2005.

[WEG+03a] H. Wang, J. Elson, L. Girod, D. Estrin, and K. Yao. Target classification and localization in habit monitoring. In *Proc. 2003 Int'l Conf. Acoustics, Speech, and Signal Processing (ICASSP 2003)*, volume 4, pages 844–847. IEEE, Piscataway, NJ, April 2003.

[WEG03b] H. Wang, D. Estrin, and L. Girod. Preprocessing in a tiered sensor network for habitat monitoring. *EURASIP Journal on Applied Signal Processing*, 2003(4):392–401, March 2003. Special Issue on Sensor Networks.

[Wei91] M. Weiser. The computer for the 21st century. *Scientific American*, 265(3):66–75, September 1991.

[WFF03] E. Welsh, W. Fish, and J.P. Frantz. GNOMES: A testbed for low power heterogeneous wireless sensor networks. In *Proc. 2003 Int'l Symp. Circuits and Systems (ISCAS '03)*, volume 4, pages 836–839. IEEE, Piscataway, NJ, May 2003.

[WHSC01] A. Wang, W.R. Heinzelman, A. Sinha, and A.P. Chandrakasan. Energy-scalable protocols for battery-operated microsensor networks. *Journal VLSI Signal Processing*, 29(3):223–237, November 2001.

[WJH97] A. Ward, A. Jones, and A. Hopper. A new location technique for the active office. *IEEE Personal Communications*, 4(5):42–47, October 1997.

[WLLP01] B. Warneke, M. Last, B. Liebowitz, and K.S.J. Pister. Smart Dust: Communicating with a cubic-millimeter computer. *IEEE Computer*, 34(1):44–51, January 2001.

[WMF02] E. Welsh, P. Murphy, and J.P. Frantz. Improving connection times for Bluetooth devices in mobile environments. In *Proc. Int'l Conf. Fundamentals of Electronics Communications and Computer Sciences (ICFS 2002)*, March 2002.

[WMM+01] A. Wang, R. Min, M. Miyazaki, A. Sinha, and A.P. Chandrakasan. Low power sensor networks. In A. Gatherer and E. Auslander, editors, *The Application of Programmable DSPs in Mobile Communications*, pages 299–326. John Wiley & Sons, New York, 2001.

[WSA+96] R. Want, B.N. Schilit, N.I. Adams, R. Gold, K. Petersen, D. Goldberg, J.R. Ellis, and M. Weiser. The ParkTab ubiquitous computing experiment. In T. Imielinski and H.F. Korth, editors, *Mobile Computing*, pages 45–102. Kluwer Academic Publishers, Norwell, MA, 1996.

[Wu02] J. Wu. *Handbook of Wireless Networks And Mobile Computing*, chapter Dominating-Set-Based Routing in Ad Hoc Wireless Networks, pages 425–450. John Wiley & Sons, New York, 2002.

[WYM+02] H. Wang, L. Yip, D. Maniezzo, J.C. Chen, R.E. Hudson, J. Elson, and K. Yao. A wireless time-synchronized COTS sensor platform: Applications to beamforming. In *Proc. IEEE CAS Workshop Wireless Communications and Networking 2002*. IEEE, Piscataway, NJ, September 2002.

[WZ04] R. Wattenhofer and A. Zollinger. XTC: A practical topology control algorithm for ad-hoc networks. In *Proc. 18th Int'l Parallel and Distributed Processing Symposium (IPDPS 2004)*, page 216. IEEE CS Press, Los Alamitos, CA, April 2004.

[XRC+04] N. Xu, S. Rangwala, K.K. Chintalapudi, D. Ganesan, A. Broad, R. Govindan, and D. Estrin. A wireless sensor network for structural monitoring. In *Proc. 2nd ACM Conf. Embedded Networked Sensor Systems (SenSys 2004)*, pages 13–24. ACM Press, New York, November 2004.

[YBV00] K. Young-Bae and N.H. Vaidya. Location-aided routing (LAR) in mobile ad hoc networks. *Wireless Networks*, 6(4):307–321, 2000.

[YHE04] W. Ye, J. Heidemann, and D. Estrin. Medium access control with coordinated adaptive sleeping for wireless sensor networks. *IEEE Transactions on Networking*, 12(3):493–506, June 2004.

[ZBC01] G.V. Záruba, S. Basagni, and I. Chlamtac. BlueTrees – scatternet formation to enable Bluetooth-based personal area networks. In *Proc. IEEE Int'l Conf. on*

Communications (ICC 2001), volume 1, pages 273–277. IEEE, Piscataway, NJ, June 2001.

[Zha02] J. Zhang. *Handbook of Wireless Networks And Mobile Computing*, chapter Location Management in Cellular Networks, pages 27–50. John Wiley & Sons, New York, 2002.

[ZSLM04] P. Zhang, C.M. Sadler, S.A. Lyon, and M. Martonosi. Hardware design experiences in ZebraNet. In *Proc. 2nd ACM Conf. Embedded Networked Sensor Systems (SenSys 2004)*, pages 227–238. ACM Press, New York, November 2004.

Curriculum Vitae

Personal Data

Jan Beutel
born August 6, 1973 in Munich, Germany

Education

1980-1993	Various schools	Germany and Saudi Arabia
	• School education	
1993	Alexander von Humboldt Gymnasium	Constance, Germany
	• High school graduation	
1994-1999	ETH Zurich	Zurich, Switzerland
	• Studies in Electrical Engineering at ETH Zurich	
2000	ETH Zurich	Zurich, Switzerland
	• Master in Electrical Engineering	
2000-2005	ETH Zurich	Zurich, Switzerland
	• PhD student	
2004/2005	Austrian Association of Skiing Instructors	Innsbruck, Austria
	• International Certification as Ski Instructor (ISIA) and Alpine Ski Guide	

Professional Experience

1993-1994 Dieffenbacher Group Germany and Ireland

- Internship and technical staff, hydraulic press systems

1995-1998 Institute of Forming Technology, ETH Zurich Zurich, Switzerland

- Technical staff, development of measurement systems and automation applications for a metallurgy research project

1996-1998 Electronics Lab, ETH Zurich Zurich, Switzerland

- Teaching assistant for undergraduate courses and labs in electronic design

1998-1999 u-blox AG Zurich, Switzerland

- Technical staff, research and development of GPS receivers
- Evaluation and maintenance of CAD tools

1999 UC Berkeley Berkeley, CA

- Associate researcher at the Berkeley Wireless Research Center
- PicoRadio project

2000-2005 Computer Engineering and Networks Lab, ETH Zurich Zurich, Switzerland

- Research and teaching assistant for graduate courses and labs in computer architecture and embedded systems design
- Supervision of numerous term and master theses projects
- Research projects in the areas of wireless ad hoc and sensor networks, wearable and mobile computing systems
- Hard- and software design and development
- Project management and development; industrial technology transfer of the BTnode platform