MARWIS: A Management Architecture for Heterogeneous Wireless Sensor Networks

Gerald Wagenknecht, Markus Anwander, Torsten Braun, Thomas Staub, James Matheka, and Simon Morgenthaler

Institute of Computer Science and Applied Mathematics
University of Bern, Switzerland
Email: {wagen, anwander, braun, staub, matheka, morgenthaler}@iam.unibe.ch

Abstract. In this paper we present a new management architecture for heterogeneous wireless sensor networks (WSNs) called MARWIS. It supports common management tasks such as monitoring, (re)configuration, and updating program code in a WSN and considers specific characteristics of WSNs and restricted physical resources of the nodes such as battery, computing power, memory or network bandwidth and link quality. To handle large heterogeneous WSN we propose to subdivide it into smaller sensor subnetworks (SSNs), which contains sensor node of one type. A wireless mesh network (WMN) operates as backbone and builds the communication gateway between these SSNs. We show that the packet loss and the round trip time are decreased significantly in such an architecture. The mesh nodes operate also as a communication gateway between the different SSNs and perform the management tasks. All management tasks are controlled by a management station located in the Internet.

1 Introduction

A heterogeneous wireless sensor network (WSN) consists of several different types of sensor nodes (SNs). Various applications supporting different tasks, e.g., event detection, localization, and monitoring may run on these specialized SNs. In addition, new applications have to be deployed as well as new configurations and bug fixes have to be applied during the lifetime. In a network with thousands of nodes, this is a very complex task and a general management architecture is required. The questions are, how we can achieve that the monitoring, the configuration, and the code updating can be performed on heterogeneous sensor nodes during their life-time? How has such a heterogeneous WSN to be structured to handle these management tasks efficiently and automatically over the network?

In this paper, the usage of a wireless mesh network (WMN) as a backbone to build a heterogeneous WSN is motivated. The proposed new management architecture called MARWIS supports common management tasks such as monitoring the WSN, configuration of the WSN, and code updates.

This paper is structured as follows: Section 2 introduces related work on management of WSNs, middleware, code distribution, and reprogramming. Afterwards, the management architecture MARWIS is presented, including the description of a heterogeneous WSN using a WMN backbone (Section 3), the specification of the infrastructural elements (Section 4), and the management protocols (Section 5). The implementation of MARWIS is described in Section 6, the evaluation of the advantages of using WMNs as backbone for heterogenous WSNs in Section 7. A conclusion is presented in Section 8.

2 Related Work

Most management and code distribution approaches does not support heterogeneous WSN environments and distribute specific code for such SN platforms. However, advanced WSNs are typically composed of rather heterogeneous SNs, since the functionality required is highly versatile. To improve current research, our concept adds mechanisms to support heterogeneity for management in WSNs. In [1], we presented a short overview of the management architecture to be described in much more detail in this paper.

MANNA [2] is a management architecture for WSNs. It provides functions to establish configurations for WSN entities. The deployment of several manager nodes in a hierarchical way based on clustering has been proposed. TinyCubus [3] is a management and configuration framework for WSNs. It is based on a clustered architecture and assigns certain roles to the SNs. Another focus of TinyCubus is code distribution, minimizing the code fragments to be distributed in a WSN. The so-called Guerrilla management architecture [4] facilitates adaptive and autonomous management of heterogeneous ad hoc networks.

Promising concepts to hide hardware heterogeneity in WSNs are middleware approaches as presented in [5] based on either scripting language interpreters or virtual machines. MiLAN [6] provides a set of middleware mechanisms for adapting the WSN to affect the application supplied performance policy. No support for dynamic code update is included as its operation may not be changed at run-time. Impala [7] is a middleware architecture that enables modular application updates and offers repair capabilities for WSNs Maté [8] is a byte-code interpreter (virtual machine) running on TinyOS and allows run-time reprogramming. The Global Sensor Network (GSN) [9] provides a middleware for fast and flexible integration and deployment of heterogeneous WSN.

Surveys of software update techniques in WSNs are presented in [10], [11] and [12]. They focus on the execution environments at the SNs, the software distribution protocols in the network and optimization of transmitted updates. The authors of [13] propose efficient code distribution in WSNs. The focus is the reduction of the total amount of data for a code update by only transmitting the differences between the old and new code. Different optimizations like address shifts, padding and address patching are made. In [14] an incremental network programming protocol, which uses the Rsync algorithm to find variable-sized blocks that exist in both code images and then only transmits the differences is

presented. In [15] a scheme that uses incremental linking to reduce the number of changes in the code and transmits the code update with a diff-like algorithm is described. FlexCup [16] is more flexible, as the linking process is not performed at the base station, but on the SNs. Multi-hop Over-the-Air Programming (MOAP) [17] is a code distribution mechanism specifically targeted for Mica-2 motes. It focuses on energy-efficient and reliable code distribution.

3 WSN Management Scenario and Tasks

Many different applications may run in a WSN, e.g., event detection, localization, tracking, monitoring. Therefore, different types of SNs, which might measure different sensor values and perform different tasks, are required. Existing SN platforms in general have different radio modules, which are not interoperable. SNs of the same type build a sensor subnetwork (SSN), which is not able to communicate directly to another SSN. A heterogeneous WSN is built from several SSNs. To interconnect such a heterogeneous WSN mesh nodes (MNs) are proposed as gateways between these SSNs. A SN plugged into a serial interface (e.g. USB) to a MN works as gateway. The wireless MNs communicate among each other via IEEE 802.11. A possible scenario is shown in Fig. 1.

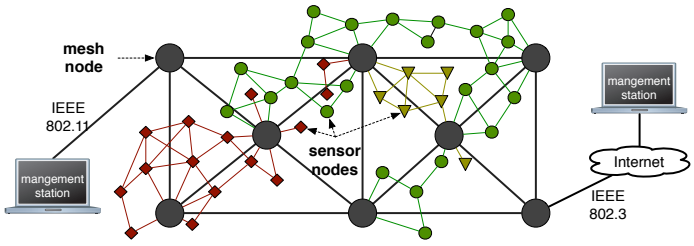


Fig. 1. A possible scenario for heterogeneous WSNs with management devices.

Currently available sensor nodes are mainly prototypes for research purposes. We have evaluated a number of sensor nodes and selected four of them to build a heterogeneous WSN: ESB nodes [18], tmote SKY [19], BTnodes [20], and MICAz [21]. For the management backbone a WMN consisting of mesh nodes with two IEEE 802.11g interfaces, an AMD Geode 233 CPU and 128 MB RAM have been selected. A 8 GB CompactFlash card can be attached.

The use of such an architecture with a WMN as backbone has various advantages. In addition to the communication gateway functions MNs further perform management tasks for heterogeneous WSNs. The main benefit is the ability to communicate with different types of SNs in several SSNs. Moreover, the use of a WMN has advantages by subdividing a huge WSN into smaller SSNs. A WSN consisting of thousands of nodes and one base station creates many communication problems. Most of them are caused by the high number of SN hops in

larger WSNs. Subdivision into smaller SSNs limits the sensitive SN links to 3 or 4 hops to the next sensor node gateway. These results in a better communication performance with a clearly lower packet delay, jitter and packet loss. SNs in the vicinity of the sink preserve energy and processing power, because they do not have to forward the whole WSN traffic. Another advantage of using a WMN is that a new SN platform can be easily inserted into the heterogeneous WSN by plugging a SN gateway into a MN. The IP address allocation depends on the corresponding MN, which makes it possible to allocate similar IP address in a physical neighborhood.

The MNs also provide management functionalities for heterogeneous WSNs. Hence, the limited SNs have less management functions to perform, which decreases memory and computation requirements. In a heterogeneous WSN with a large number of different SNs, a comprehensive management architecture is required. In addition to the MNs, providing the management functionality, there are one or more management stations (see Fig. 1). A user performs the management tasks with their support. From the management point of view there are several tasks required to manage a heterogeneous WSN. In general, the tasks can be divided into four areas: (1) monitoring the WSN and the SNs,(2) (re)configuring the WSN and the SNs, and (3) updating and reprogramming the SNs.

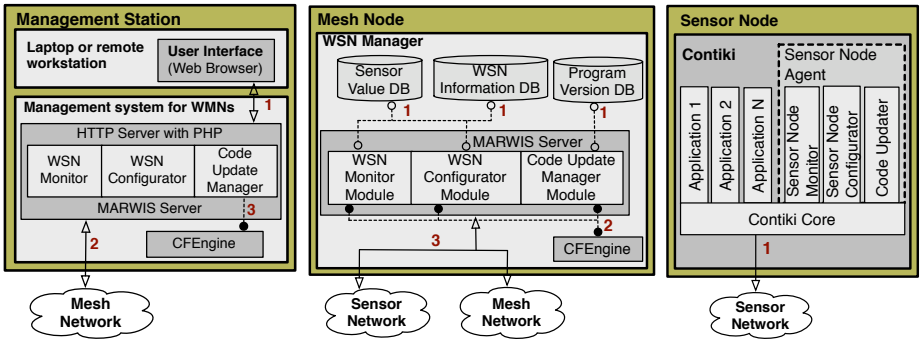
The management tasks include visualization of all SNs in the several subnetworks at the management station. Furthermore, status information about the SNs has to be monitored and displayed. This includes SN hardware features (micro-controller, memory, transceiver), SN software details (operating system versions, protocols, applications), dynamic properties (battery, free memory), and, if available, position information. SN configuration includes configuring the SNs, the running applications or the network. Updating and reprogramming the SNs is a very important issue. In a large WSN manual execution of this task is not feasible. A mechanism to handle this automatically and dynamically over the network is required. Both the operating system and applications must be updated, fully or partially. Mechanisms to handle incomplete, inconsistent, and failed updates have to be provided. Aggregating and managing the sensor values includes mechanisms to store and download the collected data from the SNs. There are many existing mechanisms and protocols, which have been designed for aggregating data from SNs (e.g. Directed Diffusion [25]). Therefore, in our architecture this task is treated as optional and not as major issue.

4 Management Architecture

The architecture to manage heterogeneous WSNs efficiently contains the following structural elements: one or more management stations, several MNs as management nodes, SN gateways plugged into a MN, and the different SNs. These elements are shown in (Fig. 2).

4.1 Management Station with Management System for WMNs

The **management station** is divided into two parts. It consists of a laptop or remote workstation to access a graphical user interface to control the WSN and a management system for WMNs [22], which is connected to the Internet and can be accessed by the remote workstation from anywhere. It includes a web server and is shown in Fig. 2(a).



(a) Management station (b) Mesh node (c) Sensor node

Fig. 2. Architecture of the MARWIS elements.

The **user interface** displays the WSN topology with the MNs including the subordinated SNs and information about the SNs (**1** in Fig. 2(a)). The **management system for WMNs** contains a small Linux distribution [22] including all required applications, especially a HTTPS server, which maintains different modules to handle the requests and transmits them to MNs, SNs or CFEngine [23], such as **WSN monitor**, **WSN configurator**, and **code update manager**. The communication with a MN is done via TCP/IP (**2**). The CFEngine distributes management data within the WMN (**3**).

4.2 Mesh Node with WSN Manager

The **WSN manager** is located on every MN provides the management functionality for the different SSNs. It consists of three databases, the **MARWIS server** with three program modules and the CFEngine (as shown in Fig. 2(b)).

The **WSN information database** stores all information about the SNs and the WSN, such as topology (neighbours, address), and states of the SNs (battery, memory). The **program version database** stores all versions of all programs for all platforms, which can be installed on the SNs, and the **sensor value database** stores all data measured by the sensors. All databases are accessible by an API to get and store data (**1** in Fig. 2(b)).

CFEngine is responsible for distributing management data within the WMN (**2**). Communication within the WMN als well with the SSN is done over TCP/IP

(**3**). The **WSN monitor module** connects to the WSN information database and to the sensor value database in order to handle the requests from the management station. It also stores data coming from the SNs into the databases. The **WSN configurator module** is responsible for the configuration tasks. It queries properties from the SNs and stores them in the WSN information database. The **code update manager module** stores newly received program images (and related information) in the program version database and notifies the management station about available programs. Compression mechanisms or differential patches are used to reduce the amount of transmitted data. To execute the updating process, it transmits the image to the SN.

4.3 Sensor Node with SN Agent

As shown in Fig. 2(c), the management tasks are handled by a **SN agent**. It consists of a SN monitor, a SN configurator, and a code updater. The **SN monitor** handles the monitor requests by sending the values to the MN. The **SN configurator** executes the configuration requests and notifies the MN. The **code updater** is responsible for the code replacement on the SN. It receives the program image of the application or operating system and performs the update by loading the new module and replacing the old one. Finally, it informs the MN about the success of the update. Communication within the SNs is done over TCP/IP (**1**).

5 WSN Management Protocols

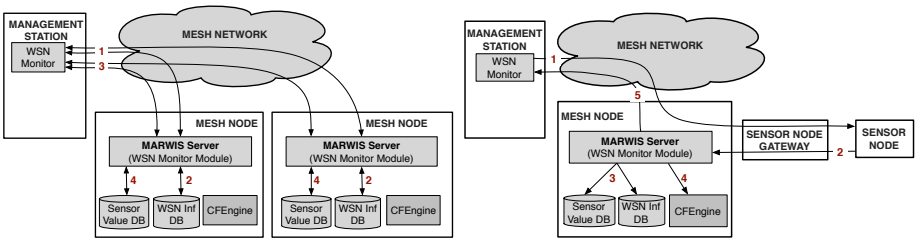
This section describes the management functionality in more detail. We consider the monitoring, configuring, and the code updating as important issues of our management architecture.

5.1 WSN Monitoring Protocol

Monitoring of the WSN can be performed in two ways. First, the management station explores the WMN and the subordinate SSNs. Alternatively, the user can query a selected sensor directly.

Fig. 3(a) shows how the management station queries the MNs about their SSNs **(1)**. The WSN monitor module queries the WSN information database **(2)**. Afterwards, the management station requests the current sensor values from every subordinate SN **(3)**, by querying the sensor value database **(4)**. All information from every SN is stored in the WSN information database and distributed in the whole WMN. Thus, we have a general view over the whole heterogeneous WSN. For displaying network topologies of SNs no additional transmissions to the SNs are required and the querying a MN is much faster than querying a SN.

Moreover, the user can request information from a SN directly and not query the WSN information database on the MN. This is shown in Fig. 3(b) and works as follows: the user requests information from a SN. The request is transmitted



(a) WSN monitor queries the MNs. (b) SN is queried directly.

Fig. 3. Monitoring the WSN.

to the queried SN **(1)**, which sends the requested value back to the MN **(2)**. The WSN Monitor Module writes the new value into the database **(3)**, and distributes it within the WMN **(4)**. Finally, it sends the requested value to the WSN monitor **(5)**.

By using a WMN as backbone the number of hops is decreased (by dividing the WSN into SSNs). This means that the communication load of a direct request to a SN occurs mainly in the WMN. Thus, the request can be processed much faster and more energy-efficient. Overload and congestion in WSNs are prevented.

5.2 WSN Configuration Protocol

With the WSN configuration protocol the properties of the SNs as well as the network can be configured. Examples are switching sensors on/off, or changing routing tables. The procedure is similar to the WSN monitoring, but a configuration command is included in the request. As the SN configurator has a universal interface and hides the SN type specific characteristics, the packets with the configuration commands are independent of the node type. The heterogeneity of the WSN is hidden from the user, and therefore the configuration can be processed without knowledge of the specific node type. One or more sensor nodes can be targeted. When a new SN joins, first an initial network configuration is negotiated. Afterwards all available data is requested from the SN by the configuration module on the MN and propagated within the WMN.

5.3 Code Update Protocol

The code update protocol consists of three main subtasks. The new image of an application or the operating system is uploaded and stored in the program version database and distributed within the WMN. The management station is notified about the programs available. Finally, the image is transmitted to the SN performing the update. One or more sensor nodes can be targeted.

The main part of the protocol is the updating process of the SNs as shown in Fig. 4. The program version and the SNs are selected, sent to the involved

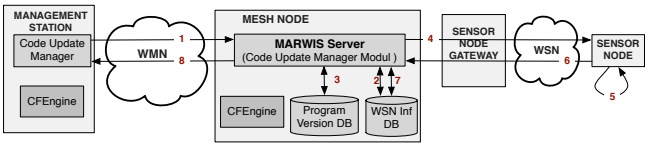


Fig. 4. The code update for the senor node is initiated.

MNs **(1)**, and checked by querying the WSN information database **(2)**. The image is taken from the program version database **(3)**, and sent to the selected SN **(4)**. On the SN the update is performed **(5)** and acknowledged **(6)**. The WSN information database is updated **(7)** and finally the management station is notified **(8)**.

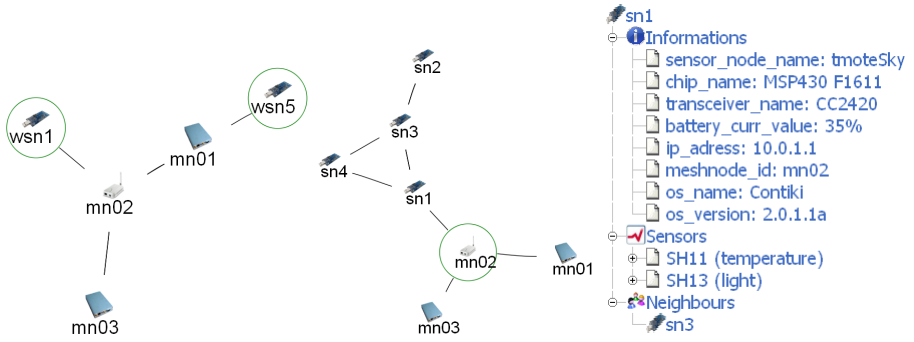
6 Implementation

This section describes the implementation of the first prototype of MARWIS. Four different components have to be realized: first, the management station with the **user interface** and the **management system for WMNs**; second, the MNs building a multi-hop WMN and providing the above described management functionality; third, the SN gateways to enable communication between the WMN and the WSN; and fourth, the different SNs running Contiki and building the heterogeneous WSN.

The management station consists of two computers. On one a Live-CD system is running, which contains all necessary programs and configurations to start the management software. The other simply provides a web browser for the user interface and connects over HTTPS to the computer running the Live-CD. This architecture is very flexible, because on one hand the Live-CD system requires only minimal hardware performance and can be booted on almost every standard computer. On the other hand, the system with the user interface can be located somewhere in the internet and has to support just a web browser.

The user interface visualizes the topology of the networks (Fig. 5) and the information about the SNs (Fig. 5(c)). By clicking on the hosting MN (e.g. mn01) the corresponding sensor subnetwork is shown (Fig. 5(b). By clicking on a SN (e.g sn01) the information about the SN, the installed sensors and the neighbors are shown (Fig. 5(c)). By selecting an element (e.g. the operating system version) it can be configured or an update can be initiated. The graphics are created by Graphviz 2.12 [26] using the neighborhood data from the WSN information database. Position data by GPS or distance estimation by radio signal analysis can be included in the neighborhood data for Graphviz to achieve an accurate topology illustration.

The Live-CD with the **management system for WMNs** contains a small Linux distribution (kernel 2.6.14.6) including all required applications, especially a HTTPS server for the connection with the user interface. The modules handling the management tasks and communication to the MNs are implemented as a



(a) WMN. (b) WSN. (c) SN information.

Fig. 5. User interface.

server written in C using sockets (MARWIS server). The databases are managed with sqlite3 [27]. The API for accessing the databases is implemented in C.

The WSN information database contains the following tables: meshnodes, meshnet, sensornodes, sensornet, sensors, swcomponents, and properties. The network tables list the neighbors of a MN responsible of a SN. The properties table is the central table, which contains the IDs of all possible properties, such as chip name, battery, memory, sensor values, software components. The tables sensornodes, sensors, and swcomponents contain the current values, and time stamps of the according properties. The sensor value database contain also the sensors table, and stores the whole history of the values. The program version database contains one table, which has meta information of a program's version (such as name, version, platform) and the link to the file. These databases are updated over our management data distribution system using CFEngine.

The CFEngine is designed to keep installations and configuration files in large computer networks up to date. In our approach, it distributes the topology and collected sensor data in the WMN. Therefore, a directory with files to share is specified in each MN. New files to distribute are copied into this directory. A neighbor node checks for new files and downloads them if necessary. Thus, the information is propagated through the whole network. Scripts can be executed handling the propagated data, e.g., write them into the according databases. MNs provide enough memory to store all this data (ca. 100MB for a 1000-node WSN running during one year with a measurement cycle of one hour).

On the MNs the same software as on the Live-CD system is running, except for the HTTPS server with PHP. The modules handling the management tasks and communication are also implemented in C. To communicate with the SSN a Serial Line Interface Protocol (SLIP) over the Linux TUN/TAP kernel module is used. SLIP connects the IP layer of the MN directly to the IP layer of the SN gateway. The SN gateway can communicate directly with the SSN.

Contiki [24] is running on the SNs as a operating system. The code updater on the SN is responsible for the code replacement. Contiki works with loadable

modules to allow replacing only parts of the operating system or applications. Except for the Contiki kernel, all modules can be replaced at run-time. The system has to be rebooted for kernel updates. The newly written applications have to fulfill just small constraints. To start and finish the application the functions _init() and _fini() are required and the application has to be complied as ELF (Executable and Linkable Format) loadable. The standard ELF or CELF (Compact ELF) is used. In contrast to ELF files CELF files are represented with 8 and 16-bit data types, which is adequate for 8-bit micro-controllers. Therefore, CELF files are usually half the size of the corresponding ELF file, but cannot be loaded by a standard ELF file handler. The code updater listens to the TCP port 6510 for new images of applications. On the MN side a small program is running, which sends a given image to port 6510 at a selected IP address.

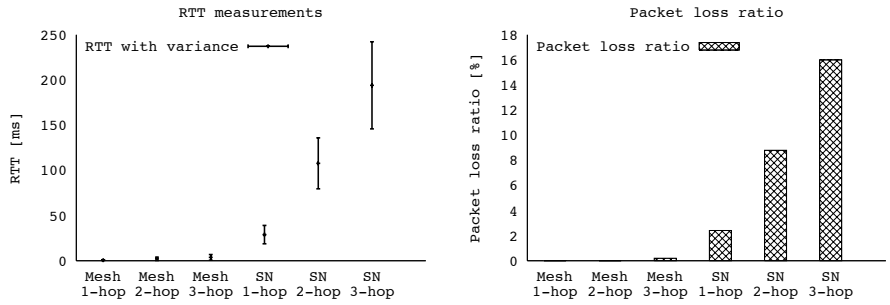
After reception the referenced variables are checked and functions of the new application are linked and relocated by the Contiki dynamic Link Editor (CLE) and the application is copied to the ROM. Then the code updater starts the application using the _init() function.

7 Evaluation

The further away a SN is located from the base station the higher the packet loss and the round trip time (RTT) are. Retransmissions for lost packets usually increase RTT and jitter. In our network architecture the WMN builds a fast backbone. Every SN in our network can connect with an average hop count of 2 to 3 hops to a mesh node. For investigation of the additional RTT and packet loss for increased hop count in WSN and WMN, we made experiments with ICMP echo request (ping). RTT time was evaluated with one, two and three hop sensor node and mesh node links. As MNs, an AMD Geode 233 CPU and 128 MB RAM with two 60 mW IEEE 802.11g interfaces, were used. The SN network tests were made with tmote sky nodes running a standard Contiki installation, as these SN platform features the fastest radio module (CC2420 with 1mW transmission power) and the highest amount of RAM. The SNs and MNs are located in different rooms. They are placed in such distance from each other distance between them is far from each other distanced that a node can only communicate with its one hop neighbors. Two hop neighbors are just recognized as interferences.

Fig. 6(a) shows the different RTT times over 500 measurements. An additional SN hop causes over 50 times longer RTT than a MN hop. Fig. 6(b) shows the packet loss over 500 packets, which is also much higher than on a mesh link. Although the measured values strongly depend on the hardware used as well as on the operating system and the communication protocols, the difference between a mesh and a sensor node hop is obvious. A large WSN with more than a dozen hops will be unserviceable. Moreover, it is possible to connect different SSNs with a WMN. With a MN backbone and directional antennas it is also possible to connect distant SN networks. Measurement where the sensor nodes

are located within one room on one table are not realistic, less than the 1% of the packets get lost.



(a) RTT. (b) Packet loss ratio.

Fig. 6. Evaluation.

8 Conclusion

Most of the existing management approaches for WSNs are not dealing with heterogeneity. MARWIS, a management architecture for heterogeneous WSNs, solves this drawback. We showed that a complex heterogeneous WSN can be managed in an efficient way by using WMNs forming a backbone. We evaluated that the packet loss and the RTT is decreased significantly using a WMN as backbone, which is a precondition for an efficient management of heterogeneous WSNs. The MARWIS architecture with the modules and the used protocols provides the monitoring, the reconfiguration and the code updating of SNs in large heterogeneous WSNs.

References

1. M. Anwander, G. Wagenknecht, T. Staub, T. Braun: Management of Heterogeneous Wireless Sensor Networks, *6. FG Sensornetzwerke, Aachen, Germany, July 2007.*
2. L. B. Ruiz, J. Nogueira, A. Loureiro: Manna: A Management Architecture for Wireless Sensor Networks. *IEEE Communications, 41(2):116-125, February 2003.*
3. P. J. Marrón, A. Lachenmann, D. Minder, J. Hähner, M. Gauger, O. Saukh, K. Rothermel: TinyCubus: A Flexible and Adaptive Framework for Sensor Networks. *EWSN'05, Istanbul, Turkey, January 2005.*
4. C. C. Shen, C. Srisathapornphat, C. Jaikaeo: An Adaptive Management Architecture for Ad Hoc Networks. *IEEE Communication, 41(2):108-115, February 2003.*
5. K. Römer, O. Kasten, F. Mattern: Middleware Challenges for Wireless Sensor Networks. *SIGMOBILE Mob. Comput. Commun. Rev., 6(4):59-61, October 2002.*

6. W. B. Heinzelman, A. L. Murphy, H. S. Carvalho, M. A. Perillo: Middleware to Support Sensor Network Applications. *IEEE Network, 18(1):6-14, Jan/Feb 2004.*
7. T. Liu, M. Martonosi: Impala: A Middleware System for Managing Autonomic, Parallel Sensor Systems. *PPoPP'03, San Diego, CA, USA, June 2003.*
8. P. Levis, D. Culler: Maté: A Tiny Virtual Machine for Sensor Networks. *ASPLOS-X'02, San Jose, CA, USA, October 2002.*
9. K. Aberer, G. Alonso, D. Kossmann: Data Management for a Smart Earth. *SIGMOD Rec., 35(4):40-45, December 2006.*
10. C. Han, R. Kumar, R. Shea, M. Srivastava: Sensor Network Software Update Management: A Survey. *Int. Journal of Network Man., 15(4):283-294, July 2005.*
11. S. Brown, C. J. Sreenan: Updating Software in Wireless Sensor Networks: A Survey. *Technical Report UCC-CS-2006-13-07, University College Cork, Ireland, July 2006.*
12. Q. Wang, Y. Zhu, L. Cheng: Reprogramming Wireless Sensor Networks: Challenges and Approaches. *IEEE Network, 20(3), 48-55, May/June 2006.*
13. N. Reijers, K. Langendoen: Efficient Code Distribution in Wireless Sensor Networks. *WSNA 03, San Diego, CA, USA, September 2003.*
14. J. Jeong, D. Culler: Incremental Network Programming for Wireless Sensors. *SECON'04, Santa Clara, CA, October 2004.*
15. J. Koshy, R. Pandy: Remote Incremental Linking for Energy-Efficient Reprogramming of Sensor Networks. *EWSN'05, Istanbul, Turkey, January 2005.*
16. P. J. Marrón, M. Gauger, A. Lachemann, D. Minder, O. Saukh, K. Rothermel: FlexCup: A Flexible and Efficient Code Update Mechanism for Sensor Networks. *EWSN'06, Zürich, Switzerland, February 2006).*
17. T. Stathopoulos, J. Heidemann, D. Estrin: A Remote Code Update Mechanism for Wireless Sensor Networks. *Technical Report CENS-TR-30, University of California, Los Angeles, USA, November 2003.*
18. Scatterweb: Platform for Self-configuring Wireless Sensor Networks. http://www.scatterweb.net. Last visit March 2008.
19. Tmote SKY: Reliable Low-power Wireless Sensor Networking Platform for Development. http://www.moteiv.com. Last visit March 2008.
20. BTnode: Flexible Platform for Fast-prototyping of Sensor and Ad-hoc Networks. http://www.btnode.ethz.ch. Last visit March 2008.
21. MICAz: The MICAz is a 2.4 GHz, IEEE/ZigBee 802.15.4, board used for low-power, wireless sensor networks. http://www.xbow.com. Last visit March 2008.
22. T. Staub, D. Balsiger, M. Lustenberger, T. Braun: Secure Remote Management and Software Distribution for WMNs *ASWN'07, Santander, Spain, May 2007.*
23. M. Burgess: A Tiny Overview of CFEngine: Convergent Maintenance Agent. *MARS/ICINCO'05, Barcelona, Spain, September 2005.*
24. A. Dunkels, B. Grönvall, T. Voigt: Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors. *EmNetS'04, Tampa, FL, USA, November 2004.*
25. C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, F. Silva: Directed Diffusion for Wireless Sensor Networking. *ACM/IEEE Transactions on Networking, 11(1):2-16, February 2002.*
26. Graphviz: A Graph Visualization Software http://www.graphviz.org. Last visit March 2008.
27. SQLite: A Self-contained, Serverless, Zero-configuration, Transactional SQL Database Engine. http://www.sqlite.org. Last visit March 2008.
28. T. Braun, T. Voigt, A. Dunkels: TCP Support for Sensor Networks. *WONS'07, Obergurgl, Austria, January 2007.*