ENERGY-EFFICIENT MANAGEMENT OF HETEROGENEOUS WIRELESS SENSOR NETWORKS

Inauguraldissertation
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Gerald Wagenknecht

von Görlitz, Deutschland

Leiter der Arbeit:
Professor Dr. Torsten Braun
Institut für Informatik und angewandte Mathematik

ENERGY-EFFICIENT MANAGEMENT OF HETEROGENEOUS WIRELESS SENSOR NETWORKS

Inauguraldissertation
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Gerald Wagenknecht

von Görlitz, Deutschland

Leiter der Arbeit:
Professor Dr. Torsten Braun
Institut für Informatik und angewandte Mathematik

Von der Philosophisch-naturwissenschaftlichen Fakultät angenommen.

Bern, 06.05.2013

Der Dekan:
Prof. Dr. Silvio Decurtins

Abstract

Various applications for the purposes of event detection, localization, and monitoring can benefit from the use of wireless sensor networks (WSNs). Wireless sensor networks are generally easy to deploy, with flexible topology and can support diversity of tasks thanks to the large variety of sensors that can be attached to the wireless sensor nodes. To guarantee the efficient operation of such a heterogeneous wireless sensor networks during its lifetime an appropriate management is necessary.

Typically, there are three management tasks, namely monitoring, (re) configuration, and code updating. On the one hand, status information, such as battery state and node connectivity, of both the wireless sensor network and the sensor nodes has to be monitored. And on the other hand, sensor nodes have to be (re)configured, e.g., setting the sensing interval. Most importantly, new applications have to be deployed as well as bug fixes have to be applied during the network lifetime. All management tasks have to be performed in a reliable, time- and energy-efficient manner.

The ability to disseminate data from one sender to multiple receivers in a reliable, time- and energy-efficient manner is critical for the execution of the management tasks, especially for code updating. Using multicast communication in wireless sensor networks is an efficient way to handle such traffic pattern. Due to the nature of code updates a multicast protocol has to support bulky traffic and end-to-end reliability. Further, the limited resources of wireless sensor nodes demand an energy-efficient operation of the multicast protocol. Current data dissemination schemes do not fulfil all of the above requirements.

In order to close the gap, we designed the Sensor Node Overlay Multicast (SNOMC) protocol such that to support a reliable, time-efficient and energy-efficient dissemination of data from one sender node to multiple receivers. In contrast to other multicast transport protocols, which do not support reliability mechanisms, SNOMC supports end-to-end reliability using a NACK-based reliability mechanism. The mechanism is simple and easy to implement and can significantly reduce the number of transmissions. It is complemented by a data acknowledgement after successful reception of all data fragments by the receiver nodes. In SNOMC three different caching strategies are integrated for an efficient handling of necessary retransmissions, namely, caching on each intermediate node, caching on branching nodes, or caching only on the sender node. Moreover, an option was included to pro-actively request missing fragments.

SNOMC was evaluated both in the OMNeT++ simulator and in our in-house real-world testbed and compared to a number of common data dissemination protocols, such as Flooding, MPR, TinyCubus, PSFQ, and both UDP and TCP. The results showed that SNOMC outperforms the selected protocols in terms of transmission time, number of transmitted packets, and energy-consumption. Moreover,

we showed that SNOMC performs well with different underlying MAC protocols, which support different levels of reliability and energy-efficiency. Thus, SNOMC can offer a robust, high-performing solution for the efficient distribution of code updates and management information in a wireless sensor network.

To address the three management tasks, in this thesis we developed the Management Architecture for Wireless Sensor Networks (MARWIS). MARWIS is specifically designed for the management of heterogeneous wireless sensor networks. A distinguished feature of its design is the use of wireless mesh nodes as backbone, which enables diverse communication platforms and offloading functionality from the sensor nodes to the mesh nodes. This hierarchical architecture allows for efficient operation of the management tasks, due to the organisation of the sensor nodes into small sub-networks each managed by a mesh node. Furthermore, we developed a intuitive -based graphical user interface, which allows non-expert users to easily perform management tasks in the network. In contrast to other management frameworks, such as Mate, MANNA, TinyCubus, or code dissemination protocols, such as Impala, Trickle, and Deluge, MARWIS offers an integrated solution monitoring, configuration and code updating of sensor nodes.

Integration of SNOMC into MARWIS further increases performance efficiency of the management tasks. To our knowledge, our approach is the first one, which offers a combination of a management architecture with an efficient overlay multicast transport protocol. This combination of SNOMC and MARWIS supports reliably, time- and energy-efficient operation of a heterogeneous wireless sensor network.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Wireless sensor networks are widely used by real-world applications to serve research, industry and individual costumers. Hence, wireless sensor networks often may consist of a large number of heterogeneous sensor nodes. The operation of such heterogeneous network needs to be cost-efficient, energy-efficient and ensure functional reliability. The main objective of this thesis is to design and develop a management architecture that can support reliable, time- and energy-efficient operation of heterogeneous wireless sensor networks.

This chapter gives a brief overview over the field of wireless sensor networks, describes the related challenges and made contributions and finally outlines the structure of the thesis.

1.1 Overview

In last years wireless sensor networks have emerged as the technical means to support a growing number of applications not only in research and in industry but also in our daily life. This is majorly due to the characteristics of wireless sensor networks and the benefits thereof.

Wireless sensor networks are composed of large numbers of inexpensive small electronic devices called sensor nodes. A sensor node is characterized by an easy installation and, due to adaptive self-configuration, low necessity of maintenance. Each sensor node operates autonomously and is equipped with various sensors, a micro-controller and a radio module for wireless data communication. The installed sensors can measure a wide range of environmental conditions, such as temperature, humidity, illumination, pressure, and many more. A sensor node is usually battery driven, which makes energy-efficiency a very important field of current research in the area of wireless sensor networks.

Figure 1.1 depicts several typical applications for wireless sensor networks. We briefly comment on them.

- **Environmental monitoring:** Wireless sensor networks are widely used for environmental monitoring as well as for habitat and wildlife monitoring.

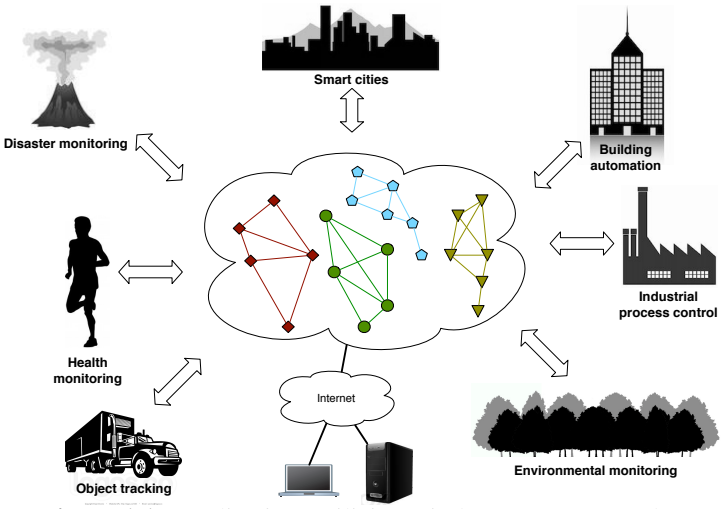


Figure 1.1: Applications utilizing wireless sensor networks.

They enable continuous, long-term unattended data collection in large areas, which are difficult to access otherwise. One example of environmental motoring is the A4-Mesh project [1]. Water flow and hydrological balance are monitored in the Swiss Alps and the results help to enhance the water supply during dry summer periods.

- **Industrial process monitoring:** Wireless sensor networks are used to monitor and control industrial processes. Sensors can measure data such as pressure, humidity, temperature, flow, viscosity, density and vibration intensity and transfer them to a control system. By this, status and condition of machines or systems can be monitored. In case of an unexpected behaviour an alarm can be triggered. Requirements for industrial applications are often stricter compared to other domains, since system failure may lead to loss of production or even worse, loss of lives.

- **Building automation:** A building automation system consists of heating, ventilation and air-conditioning, lighting, indoor transportation, security systems used to improve indoor climate, reduce energy costs and optimize building operation. The usage of wireless sensor networks in the area of building automation can reduce installation costs since wiring is avoided. Furthermore, an existing building automation system can be easily extended.

- **Smart cities:** In the area of smart cities wireless sensor networks can be used for different purposes, such as traffic and parking management as well as safety and surveillance. In order to avoid traffic jams sensor nodes are used to detect traffic density and in combination with actuators (smart traffic signs) the traffic can be redirected to less used roads. Furthermore, sensor nodes can be used to measure noise and air pollution. Detecting and tracking people can be applied for security surveillance, crowd detection, people counting

or for timekeeping systems. There is also a wide usage of wireless sensor networks in the area of smart grids.

- **Disaster monitoring:** Natural disasters such as landslides, tsunamis, earthquakes or volcanic eruptions are difficult to predict. Wireless sensor networks can be useful in two ways. First, the usage of wireless sensor networks enables a more convenient early warning system to reduce the impact of these events on lives and property. Second, wireless sensor networks provide a system able to research theses phenomena.

- **Health monitoring:** Wireless sensor networks have for years contributed in the area of health care. Continuous monitoring and analysing of vital functions of the human body is crucial for detecting when a patient's state of health changes. The sensors measure important parameters such as temperature, blood pressure, glucose level or heart and brain activity, which are analysed on a base station. This will allow continuous monitoring of the patient's state of health outside the hospital to improve life quality of especially elderly people by prolonging the time living at their home.

- **Object detection and tracking:** In the area of logistics and transportation wireless sensor networks are more and more used to detect and track object such as containers, packets, and other goods. Thus, an object can be followed at any stage of the supply-chain.

1.2 Problem Statement

The research presented in this thesis addresses major problems concerning the management of heterogeneous wireless sensor networks as well as the efficient communication supporting the management tasks.

In terms of network management the following problems are addressed:

- **Monitoring:** Overseeing the operation of a wireless sensor network is critical to ensure its purpose. Therefore it is important to follow the status of the network and the sensor nodes. Are there dead sensor nodes? What is the current connectivity of the nodes in the wireless sensor network? What is the battery state of the sensor nodes? Monitoring should include continuous collecting of status information about the sensor nodes and proper visualisation of all sensor nodes in the network and their states. The status information includes sensor node hardware features (micro-controller, memory, transceiver), sensor node software details (operating system versions, protocols, applications), dynamic properties (battery, free memory), and, if available, position information.

- **(Re)Configuration:** Configuration and re-configuration of sensor nodes is another critical task necessary to keep the wireless sensor network in opera-

tion. It is responsible for configuring the sensor nodes, their running applications and the network itself. Examples of configuration tasks are setting the sensing intervals or setting up the communications protocols (duty cycles, timers, etc).

- **Code updating:** Updating and reprogramming the sensor nodes is another important issue. An application on the sensor node might be buggy or not working. Updating solves the problem and thus keeps the network running. In a large wireless sensor network manual execution is not feasible. Hence, a mechanism to handle this automatically and dynamically over the network is required. Mechanisms to handle incomplete, inconsistent, and failed updates have to be provided as well.

To perform the described management tasks data has to be transmitted from one sender node to many heterogeneous receiver nodes in a reliable, time- and energy-efficient matter. A powerful communication scheme that provides all these properties is required:

- **Reliability:** Reliability is a key issue for critical management tasks such as code updating or (re)configuration. It should be ensured that a code image or a configuration message is successfully transmitted to all addressed receivers, posing the need for an end-to-end reliability mechanism.

- **Time-efficiency:** Although monitoring, (re)configuration and code updates are no high priority tasks, they should be executed in reasonable time. The shorter time is needed, e.g., to handle a code update, the less the operation of the wireless sensor network is interrupted.

- **Energy-efficiency:** Energy-efficient operation of the management tasks plays a very important role, given the limited power capacity of battery-operated sensor nodes. Reducing energy consumption increases the lifetime of individual sensor nodes but also of the wireless sensor network as a whole.

- **Heterogeneity:** Since today's wireless sensor networks are composed of various types of sensor node platforms performing different tasks, the management architecture should be able to service heterogeneous sensor node platforms.

1.3 Contributions

The contributions of this thesis can be grouped into two main categories: **reliable, time-, and energy-efficient overlay multicast protocol** (SNOMC) and a **management architecture** (MARWIS), which supports efficient monitoring, (re)configuration and code updating in heterogeneous wireless sensor networks. The combination of both protocols with MARWIS as management architecture and

SNOMC as transport protocol allows an efficient operation and management of heterogeneous wireless sensor networks. First, the individual contributions pertaining to the SNOMC (Sensor Node Overlay Multicast) protocol can be summarized as follows:

- Design of an overlay multicast protocol specifically targeting a reliable, time-efficient and energy-efficient dissemination of data in multicast manner. Using SNOMC supports an efficient operation of application such as MARWIS. Similar to other overlay protocols a distribution tree consisting of one *sender node*, several *forwarding nodes* and *branching nodes*, and one or more *receiver nodes* is built.
- Support of end-to-end reliability - in contrast to other multicast protocols, which do not integrate a reliability mechanisms, SNOMC supports a NACK-based reliability mechanism. It is combined with a data acknowledgement after successful reception of all data fragments by the receiver nodes. Using a NACK-based reliability mechanism is straightforward, easy to implement and reduces the number of transmissions significantly.
- Efficient handling of necessary retransmissions through several different caching strategies - more specifically three caching strategies were used, namely, caching on each intermediate node, caching on branching nodes, or caching only on the sender node. Moreover, an option was included to pro-actively request missing fragments. In this case the intermediate nodes, which cache the fragments actively request a detected missed fragment.
- Evaluation of SNOMC in both the OMNeT++ simulator and the Contiki OS and the comparison of its performance to other popular data dissemination protocols such as Flooding, MPR (Multipoint Relay), PSFQ (Pump Slowly, Fetch Quickly), TinyCubus, and Directed Diffusion as well as the unicast-based protocols UDP and TCP. SNOMC outperforms the mentioned protocols in terms of transmission time, number of transmitted packets and energy consumption. The evaluations show that SNOMC delivers very good performance in various environments and with different underlying MAC protocols, such as ContikiMAC and NullMAC, independent of the supported levels of reliability and energy-efficiency.
- Demonstrating that avoidance of expensive end-to-end retransmissions by caching on intermediate nodes improves performance significantly. Contrary to this, pro-actively requesting of missed fragments does not improve the performance at all. It only causes additional packets to be sent, increasing the probability of collisions and cancelling the advantage of SNOMC's optimization.

In summary SNOMC offers a robust, high-performing solution for the efficient distribution of code updates and management information in a wireless sensor network. This, it can be used in combination with MARWIS for an efficient operation

and management of heterogeneous wireless sensor networks. Second, we summarize the contributions pertaining to MARWIS (Management Architecture for Wireless Sensor Networks).

- Design and implementation of an architecture for monitoring, (re) configuration, and code updating in heterogeneous wireless sensor networks. In contrast to other management frameworks, such as Mate, MANNA, TinyCubus, or code dissemination protocols, such as Impala, Trickle, and Deluge, MARWIS offers an integrated solution of the three main management task (monitoring, configuration and code updating of the sensor nodes).
- Execution of management tasks in reliable, time- and energy-efficient manner. This to a big extend is thanks to the distinguished feature of MARWIS, namely, the use of wireless mesh nodes as backbone. The approach enables diverse communication platforms and offloading functionality from the sensor nodes to the mesh nodes. Moreover, the hierarchical architecture allows an efficient operation of the management tasks, because it divides a large sensor network into smaller sub-networks. The main components of MARWIS are a management station and a management (mesh) node, which enable the interaction between end users (via a user interface) and sensor node(s). Furthermore, management information can be stored on the mesh nodes, requested monitoring information can be directly transferred from the mesh node to the user, reducing energy consumption of the sensor node.
- An intuitive web-based graphical user interface that allows easy administration of the network. Users can perform management tasks on the sensor nodes in remote and user friendly fashion.
- Integration of a management architecture (MARWIS) and a communication protocol (SNOMC) to increase performance efficiency of the management tasks. The MARWIS Communication Server integrates SNOMC into MARWIS and handles the communication in the wireless mesh network and with the sub-ordinated sensor nodes. To our knowledge, our approach is the first one that offers such combination of a management architecture with an efficient overlay multicast transport protocol. This combination is also the key to the support of reliable, time- and energy-efficient operation of a heterogeneous wireless sensor network.

1.4 Thesis Outline

The main body of the thesis consists of two parts, one for each scientific contribution. Beforehand, Chapter 2 discusses the most important related work related in the areas of management and multicast in heterogeneous wireless sensor networks.

Part I focuses on the Sensor Node Overlay Multicast (SNOMC) protocol. In Chapter 3 we present the design and architecture of the protocol. Subsequently,

in Chapter 4 we discuss the implementation of SNOMC, first, in the OMNeT++ simulator and, second, in the Contiki OS. In Chapter 5 we evaluate the SNOMC protocol and compare its performance against a number of other data dissemination protocols for wireless sensor networks such as Flooding, MPR (Multipoint Relay), PSFQ (Pump Slowly, Fetch Quickly), TinyCubus, and Directed Diffusion as well as against unicast protocols such as UDP and TCP. The evaluation is done using the OMNeT++ simulator and the Wisebed testbed.

Part II introduces the second contribution of the thesis, MARWIS (Management Architecture for Wireless Sensor Networks). In this part Chapter 6 describes the design and the architecture specification of MARWIS. The implementation of MARWIS is given in Chapter 7. In Chapter 8 we present the integration of the SNOMC protocol into the MARWIS architecture and evaluate their combined performance.

Finally, the thesis concludes with Chapter 9 summarizing our main contributions on therms of designed management solutions and evaluation results. It further gives an outlook on subsequent research opportunities.

Chapter 2

Related Work

This chapter introduces and discusses the most important studies found in the literature related to our contributions in the thesis, namely, management and multicast in heterogeneous wireless sensor networks.

First we start with an introduction to the hardware and software platforms we used to built heterogeneous wireless sensor networks, presented in Section 2.1 and Section 2.2, respectively. In order to evaluate our contributed protocols we are using different evaluation platforms, which are introduced in Section 2.3. Section 2.4 presents approaches for multicasting in wireless sensor networks addressing several challenges in the area. Section 2.5 presents data dissemination protocols commonly used in wireless sensor networks. Section 2.6 discusses the Contiki protocol stack including link layer (Section 2.6.1) and network layer (Section 2.6.2). Literature related to management in wireless sensor networks is discussed in Section 2.7. Finally, Section 2.8 summarises the chapter and indicates its relation to the work presented in the rest of the thesis.

2.1 Hardware Platforms

A wireless sensor network consists of a large number of nodes, often randomly distributed in a large area. Generally, a sensor node consists of a micro-controller, some sensors, and a low-power radio for communication. Currently, available wireless sensor nodes are mainly prototypes for research purposes. We have selected several types of sensor nodes and selected four types to build a heterogeneous wireless sensor network:

- **Tmote Sky[30] / TelosB**[29],
- **MSB** [101],
- **MICAz** [28],
- and **BTnodes** [20].

They are widely used in the research community, well documented and have the adequate properties in terms memory, energy-efficiency, etc. for an efficient operation in wireless sensor networks. For MARWIS we used all four types of sensor nodes in order to build a heterogeneous wireless sensor network. For the evaluation of SNOMC we used the Tmote Sky sensor nodes. For the management and communication backbone **ALIX 3d2 mesh nodes** [81] have been selected. The individual hardware is described in more detail in the subsequent sections.

2.1.1 PC Engines ALIX

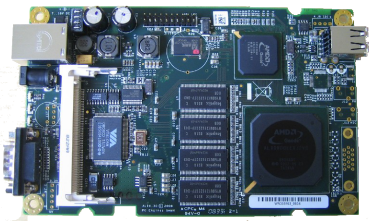


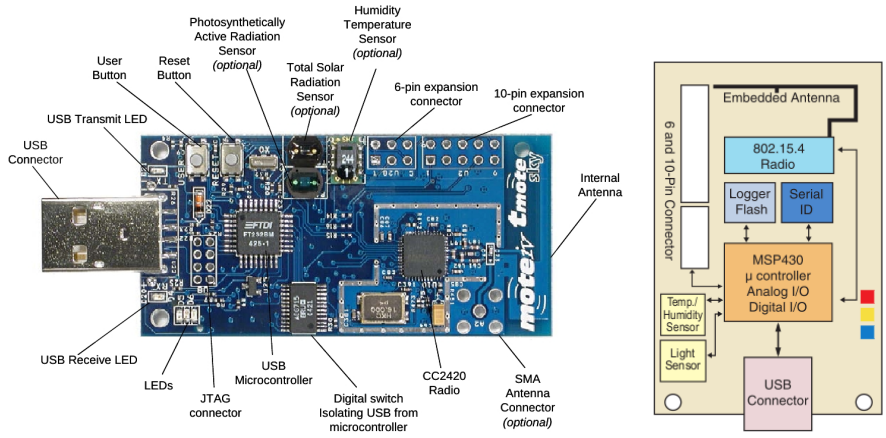
Figure 2.1: PC Engines ALIX 3d2 Board[81]

Figure 2.1 shows an ALIX 3d2 Board [81] from PC Engines GmbH. It can be used to build a fully meshed network using IEEE 802.11 radio transmitters. In general, wireless mesh nodes can be deployed in various environments, indoors as well as outdoors. They are usually realised as embedded systems to perform few dedicated functions. In case of our contribution in the thesis MARWIS (Management Architecture for Wireless Sensor Networks), these mesh nodes are used to build a backbone network and divide the wireless sensor network into smaller subnetworks. Furthermore, the mesh nodes handle the management functionality provided by MARWIS (cf. Section 6.2). To connect the sensor node gateway with the mesh node the Serial Line Interface Protocol (SLIP) is used to communicate over a Linux TUN/TAP interface. TUN/TAP are virtual network kernel devices, which create a virtual interface for the serial line. SLIP connects the IP layer of the mesh node directly to the IP layer of the sensor node gateway. The ALIX.3d2 boards feature the following components:

- An x86 compatible 500MHz AMD Geode LX800 CPU offering 256MB RAM.
- A CompactFlash socket to be equipped with a exchangeable storage card.
- one Ethernet port, two miniPCI sockets for 802.11
- Several interfaces: one serial port, two USB ports, LPC, and I2C.
- An RTC battery can be added.

- The AMD Geode processor contains a hardware watchdog, i.e., a timer that reboots the node if not periodically reset. This helps in recovering a node from a non-responsive state (self-healing).

2.1.2 Crossbow Tmote Sky / TelosB



(a) Front part[30]. (b) Block diagram[29].

Figure 2.2: Crossbow Tmote Sky.

The Tmote Sky[30] sensor node depicted in Figure 2.2 is the successor of Crossbows TelosB[29] platform. Figure 2.2(a) shows the front part of the sensor node with the components, such as micro-controller, radio transceiver and many more. Figure 2.2(b) shows the simplified block diagram of the Tmote Sky. The Tmote Sky platform was developed by the Crossbow Corporation and the University of Berkley and is an open source, low-power and high data-rate wireless sensor module designed to enable innovative experimentation for the research community. The on-board bootloader allows to flash the sensor node without any additional device. This advantage makes it easy-to-use and very user friendly. It features the following components:

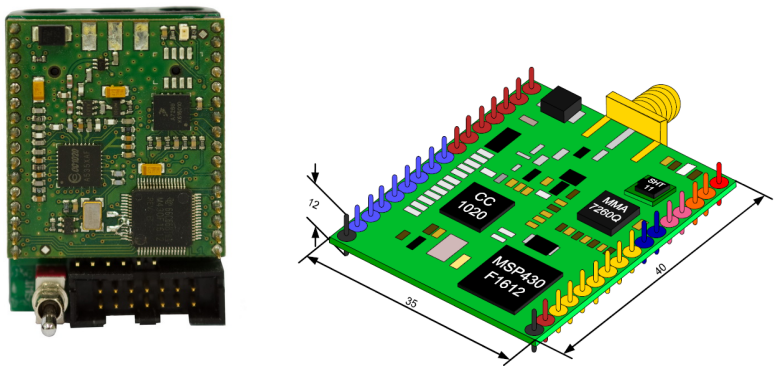
- A Texas Instruments MSP430 series RISC CPU (MSP430F1611)[117] offering some 48kB of ROM and 10kB of RAM.

- A Chipcon CC2420 [120] radio transceiver, which is an IEEE 802.15.4 compliant radio for wireless communications operating in the 2.4GHz ISM band. The radio provides a faster data rate (250 kbps) compared to the MSB430 boards, however, at a price of a lower range.

- An FT232BL chip from FTDI [48] connects the Universal Asynchronous Receiver Transmitter (UART) bus of the MSP430F1611 micro controller with the USB interface.

- The onboard temperature and humidity sensor Sensirion SHT11 [104] is capable of measuring temperature and relative humidity.
- An additional Micron M25P80 Flash memory [76] provides 1024kB of external space, e.g., for code or for logging data.

2.1.3 Scatterweb Modular Sensor Board



(a) Front part[101]. (b) Block diagram[16].

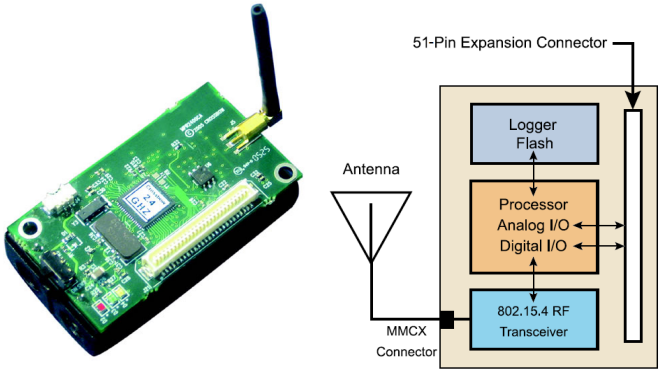
Figure 2.3: Scatterweb MSB430.

Figure 2.3 shows an MSB430 Sensor Node Platform [101]. It is developed by Freie Universität Berlin and ScatterWeb GmbH [102]. This modular node platform features the following main components:

- A Texas Instruments MSP430 series RISC CPU (MSP430F1612)[117]: this CPU can be clocked from 100 kHz up to 11 MHz. The clock speed can be adapted by a software configurable digital controlled oscillator (DCO). The MSP430F1612 has 55 KB flash memory and 5 KB RAM, and further 18-digital I/O pins connected to analog-to-digital (ADC) and digital-to-analog (DCA) converters.
- A CC1020 Chipcon [118] configurable wireless radio transceiver using a low-noise amplifier that operates in the ISM-band around 868 MHz. Its output power reaches an amplitude up to 8.6 dBm (7.2 mW). The CC1020 uses 8 channels with a data rate of 19.2 kbit/s when using Manchester encoding.
- A Secure Digital Memory Card (SD) Reader can store large amounts of data on Secure Digital High-Capacity (SDHC) cards with a capacity up to 32 GB.
- A Temperature and Humidity Sensor Sensirion SHT11 [104] is capable of measuring temperature and relative humidity.

- A Freescale MMA7260Q Accelerometer [47] is capable of measuring the acceleration in 3 dimensions (x,y,z).

2.1.4 Crossbow MICAz



(a) Front part. (b) Block diagram.

(c) Connected to the USB board.

Figure 2.4: Crossbow MICAz[28]

The MICAz Mote [28] depicted in Figure 2.4 is a third generation device used for enabling low-power, wireless sensor networks available in 2.4GHz. It was developed by the Crossbow Corporation and offers the following features:

- The MPR2400CA low-power micro-controller is based on the Atmel ATmega128L. It offers 128KB program flash memory, 4KB RAM and 4KB EEPROM.

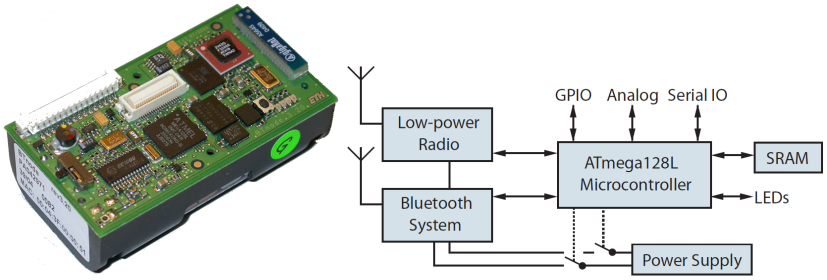
- Like the TelosB it also offers the Chipcon CC2420 [120] radio transceiver.

- The USB board uses a FT2232C chip from FTDI [48] to connect the USB interface with the ATMega128L micro-controller shown in Figure 2.4(c).

- The 51-pin expansion connector supports Analog Inputs, Digital I/O, I2C, SPI and UART interfaces. These interfaces make it easy to connect to a wide variety of external peripherals.

- The node does not provide any sensors or a serial interface. Sensors must be attached separately. For example, a MTS400CA sensor board can be attached a MicaZ. It provides the following sensors: Temperature & Humidity, Humidity Accuracy $< 3.5\%$, Temperature Accuracy < 0.5 Deg. C, Barometric Pressure 300mbar to 1100mbar, 3% Accuracy, Ambient Light Sensor (400-1000nm) response, 2-Axis Accelerometer (ADXL202).
- An additional Micron M25P80 Flash memory [76] provides 512kB of external space, e.g., for code or for logging data.

2.1.5 BTnode



(a) Front part[20]. (b) Block diagram[19].

Figure 2.5: BTnode platform.

The BTnode [20] depicted in Figure 2.5 is an autonomous wireless communication and computing platform and serves as a demonstration platform for research in distributed wireless sensor networks. It has been jointly developed at ETH Zürich by the Computer Engineering and Networks Laboratory (TIK) and the Research Group for Distributed Systems. They basically come with the same hardware features as the widely used MICA2 Mote from Crossbow [27]. However, they have more SRAM and an additional Bluetooth radio interface. It offers the following features:

- It contains an Atmel ATmega128L micro-controller[15] (AVR RISC 8 MHz @ 8 MIPS) with, 128 KByte flash memory, 4 KByte RAM, AND 4 KByte EEPROM.
- As Bluetooth subsystem is has the Zeevo ZV4002 chip supporting AFH/SFH Scatternets with max. 4 Piconets/7 Slaves.
- The Chipcon CC1000 [119] is the low-power radio and operates in the ISM Band 433-915 MHz.
- The CC1000 radio transceiver and the Atmel ATmega128L micro-controller are connected using the SPI bus.
- Using the UART bus the BTnode can be connected to an external device.

- Further external interfaces are I2C, GPIO, ADC, Clock, Timer, LEDs

2.2 Operating Systems

We are using two different software platforms. The ADAM (Administration and Deployment of Adhoc Mesh networks) [113] platform is running on the mesh nodes. For the sensor nodes we are using the Contiki OS [38].

2.2.1 ADAM

ADAM [113] [112] has been developed to provide a user-friendly, intuitive and extendable build system for a customized embedded Linux operating system for WMNs. ADAM can cope with unavailable nodes and automatically repairs configuration and software update errors and does not require a co-located backbone network for management. It improves connectivity between the network nodes and avoids costly on-site repairs. The ADAM framework is released under GPLv2 license [2]. Beside MARWIS, it has been also used in other projects, e.g., CTI-Mesh [4], WISEBED [105], A4-Mesh [1], and LBA [8].

Concept and Architecture

ADAM uses a **(1)** decentralized distribution mechanism, **(2)** self-healing mechanisms for safe configuration and software updates and **(3)** a separation of node specific configuration and binary software images that are specific for a node type.

- The decentralized mechanism for distributing software and configuration updates is the first main concept of ADAM. Each node periodically pulls new software or configuration updates from its one-hop neighbors. In this way, an update is transmitted through the network from one to the next node, independently from the underlying routing protocol. If an update reaches a mesh node it is applied automatically.

- The second main concept of ADAM are the self-healing mechanisms, which include monitoring of the network topology during updates, detection of isolated nodes, and automatic rollback to the latest running software if a software update fails to boot properly. For example, if the self-healing mechanism monitors a reduced number of neighbours after lowering the transmission power, it step-wisely increases the transmission in order to reach at least predefined network connectivity. If a node detects that it does not have any neighbours and is thus isolated, it follows an automatic lost node procedure for re-joining the network. If a software update failed and the node was prevented from properly booting after the update it starts an automatic rollback process and loads the latest known working software.

- The third main concept of ADAM is the separation of node specific configuration and binary software images. Each node in an ADAM network contains two image files: one for the operating system kernel and the binaries, and one for the configuration, which holds the node specific parts. ADAM even splits up this configuration image into the normal configuration files and a special network configuration file, which contains all dynamic network parameters. As a result ADAM must usually only distribute this small (10KB) network configuration file and not the whole software image (6MB). This drastically reduces the total amount of transferred data for an update.

ADAM Build System

The goal of the ADAM build system is to simplify all necessary steps for image creation. It consists of two tools, namely the *build-tool* to compile the software and the *image-tool* to pack the software correctly into the images.

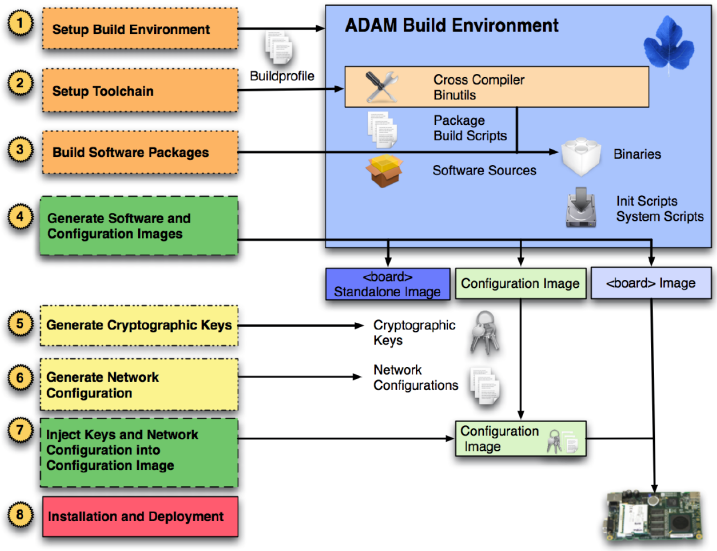


Figure 2.6: Steps of the build and setup process for a node [113]

.

Figure 2.6 illustrates the necessary steps to build a Linux distribution for an ADAM mesh node. Step 1 contains the creation of the build environment for the target platform and the set up with all necessary parameters for the cross compilation process, such as of library and compiler paths. In step 2 the tool-chain for the cross compiler is set up. The cross compiler is used to compile all software packages for the target platform in step 3. In step 4, the image-tool is used to generate the software image for the target platform and individual configuration images for each node. In the steps 5 and 6, cryptographic key pairs for the distribution engine and the network configuration for each node are generated. The node-specific keys and the network configurations are then injected into the configuration image of

the corresponding node in step 7. In the final step, the generated Linux system images are loaded on the secondary storage of the new nodes or distributed using the ADAM distribution engine.

2.2.2 Contiki Operating System

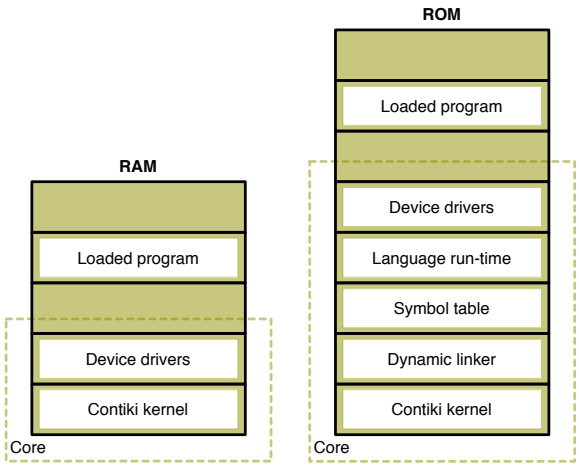


Figure 2.7: Partitioning in Contiki: the core and loadable programs in RAM and ROM. [38]

The Contiki OS [38] is a lightweight and flexible operating system designed for resource constraint embedded systems. It was developed at the Swedish Institute of Computer Science (SICS). A Contiki system is divided into two parts: the core and the loadable programs as shown in Figure 2.7. The core consists of the Contiki kernel, device drivers, a set of standard applications, parts of the C language library, and a symbol table. All parts of the operating system are written in C. Loadable programs are loaded on top of the core and do not modify the core. The Contiki kernel is event-driven and supports a simple first in / first out (FIFO) scheduler. It does not provide a hardware abstraction layer, but device drivers and applications can communicate directly with the hardware. Communication between processes always goes through the kernel. Preemptive multitasking can be added to specific processes by using an application library. Contiki is highly portable and supports amongst others the **Tmote Sky / TelosB**, the **MSB / ESB**, and the **MicaZ** sensor node platforms. In order to communicate with other sensor nodes and to integrate wireless sensor networks with IP networks, Contiki provides the small TCP/IP stack called µIP [34] and another lightweight layered communication stack for sensor networks called RIME [35, 40].

The main characteristics of Contiki can be summarized as:

- Multi-thread processing, which is implemented using a hybrid model to handle the processes.

- An event-driven kernel where pre-emptive multi-threading uses an application library, which is optionally linked with programs that explicitly require it.
- A set of libraries, which can be loaded to the devices' memory according to the application requisites.
- Only one stack to buffer the data in the memory management
- Reprogramming and dynamic linking.
- Small but fully compliant TCP/IP stack called µIP in combination with the RIME stack (described in Section 2.6.2).

In the next sections we describe the main parts of Contiki, namely the handling of processes and the scheduling, the reprogramming and the protocol stacks of µIP and RIME, and the integrated MAC protocols.

Contiki Processes and Scheduling

Most programming environments for sensor nodes are based on an event-driven model and not on traditional multi-threading. In event-driven systems, programs are implemented as event handlers. The event-handler is responsible for the external or internal events, and has to run to completion. Due to the run-to-completion semantics, an event-handler cannot execute a blocking wait. On the other hand the system can use a single, shared stack, which leads to a reduced memory overhead compared to a multi-threaded system, where memory must be allocated for a stack for each running program.

Using processes gives the developer the possibility to execute simultaneously different tasks on only one CPU or micro-controller. The operating system takes care of switching between the different tasks. The context switch consists of saving and restoring the state of a process. This is very time consuming, since a lot of data has to be copied. In this programming style an application is developed as a finite state machine. Only the different states have to be saved and not the whole stack. Depending on the state, a specific action is executed.

In Contiki so called protothreads [39] are introduced, which combine normal processes with event-driven programming. Using protothreads simplifies the development of event-driven code, since the application can be described as a linear sequence of program statements. The blocking wait statement is triggered by the `PT_WAIT_UNTIL()` function. It blocks the protothread, the code execution is stopped and the next protothread in the process queue is executed. Without this, and since a protothread consists of an infinite main loop, only one protothread would be executed and the CPU would be monopolized by busy waiting. Using the blocking wait statement `PT_WAIT_UNTIL()` the developer decides whether and when a protothread should give away the control of the CPU. This statement should be used as often as possible to avoid busy waiting. An important side-effect

of using a protothread is that all used variables have to be declared global and not local in the scope of the protothread.

Reprogramming and Run-Time Dynamic Linking

Contiki supports reprogramming of sensor nodes during run-time by using loadable modules (cf. [37]). With loadable modules, only parts of the system need to be modified when a single program is changed. A dynamic linker, which is part of the Contiki OS can link, relocate, and load standard ELF object code files [6].

A loadable module contains the native machine code of the program that is to be loaded into the system. The machine code in the module usually contains references to functions or variables in the system. These references must be resolved to the physical address of the functions or variables before the machine code can be executed. This is called linking. Linking can be done either when the module is compiled (pre-linking) or when the module is loaded (dynamic linking). A pre-linked module contains the absolute physical addresses of the referenced functions or variables. In contrary to the pre-linked module, a dynamically linked module contains the symbolic names of all system core functions or variables that are referenced in the module. This information increases the size of the dynamically linked module compared to the pre-linked module.

Additionally, the machine code in the module also contains references to functions or variables within the module itself. The physical address of these functions will change depending on the memory address at which the module is loaded in the system. The addresses of the references must therefore be updated to the physical address that the function or variable will have when the module is loaded. This process is called relocation. Like linking, relocation can be done either at compile-time or at run-time.

When a module has been linked and relocated the program loader loads the module into the system. This happens by copying the linked and relocated native code into a place in the memory from where the program can be executed.

2.3 Evaluation Platforms

This section we introduce different evaluation platforms we use to evaluate our protocol against existing protocols. Particularly, we distinguish between two types of evaluation platform: **network simulators** and **real-world testbeds**. Furthermore, we describe the energy measurement in Contiki OS.

Network simulators are mainly used at the development phase of a protocol. They provide flexible set ups for wireless sensor networks with different topologies and different sizes (up to hundreds or thousands of sensor nodes). Experiments can be repeated many times with the same network conditions. Moreover, the simulation can be stopped and reseted at any time to identify bugs and problems in the simulated protocol. Since, the behaviour of a single wireless sensor node can

be monitored and visualized the debugging of the protocol implementation is an easy process.

Due to wide gaps between simulation results and real-world results, the suitability of simulations has been questioned. Contrary to simulations protocols, the wireless environment does not need to be modelled using a real-world testbed for wireless sensor networks. Thus, an evaluation of protocols gives much more realistic results.

There are a number of existing real-world testbeds for wireless sensor networks. Beside the different Wisebed testbeds [105], the Harward University (MoteLab[130]), the Technical University of Berlin (TWIST [51]), the Ohio State University (Kansei [42]), University of Southern California (TutorNet [121]), and the University of Uppsala (Sensei-UU [92]) are running different wireless sensor network testbeds.

2.3.1 OMNeT++ Network Simulation Framework

For our evaluations we have used the Objective Modular Network Testbed (OMNeT++) Network Simulation Framework [9] OMNeT++ is a modular open-source network simulator and contains core modules, the Graphical User Interface (GUI), analysis tools and free available extensions such as the INET Framework [11] or the Castalia project [3]. An advantage of OMNeT++ is the modularity and that the code is open-source. This allows to add own and external functionality. The GUI support and the clean design of OMNeT++ helps to ease a straightforward simulation development process. All modules are written in C++ programming language and interconnected using the high-level language Network Description (NED). NED is used to define the network topology as well as to attach different network protocol modules to a network stack.

We are using the INET Framework, which includes UDP, TCP, and IP support. To model the radio wave propagation and support IEEE 802.15.4 protocols we use the Castalia project.

A major challenge of using simulations is to model the radio wave propagation in way that this reflects the real-world environment. To model the radio wave propagation we use a free space model for direct waves, which is parametrized for the physical characteristics of the CC2420 radio transceiver using an OQPSK modulation of a 2.4 GHz carrier wave. The receiving power (P_{Rdb}) in decibel is calculated by using the transmitting power (P_{Tdb}) in decibel and the distance (d_{Tr}) between the sender and the receiver as following formula shows:

$$P_{Rdb} = P_{Tdb} + 20log_{10}\left(\frac{\lambda}{4 \cdot \pi \cdot d_{Tr}}\right) \tag{2.1}$$

The signal strength corresponds to the calculated receiving power (P_{Rdb}). The total noise is calculated by summing up the receiving power of all other concurrently ongoing transmissions, the thermal noise and the receiver noise. The Signal-to-Noise Ratio (SNR) is the result of this calculation and determines the probability

of a bit error. The error probability corresponding to the calculated SNR value is taken from Castalia [3].

In a real-world environment external interface plays a major role. To consider these effects we simulate a probability of 15% for concurrently occurring external interferences by placing device randomly, which transmits with 1 mW (0 dBm).

2.3.2 Wisebed WSN Testbed Controlled by TARWIS

To evaluate our proposed protocols we are using a real-world testbed for wireless sensor networks called Wisebed. In a testbed for wireless sensor networks the sensor nodes are usually wired connected to a controlling unit. This can be a computer or a mesh node. With this controlling unit the image is uploaded to the sensor node and debugging data and statistically data is retrieved from the node. Further, the user can monitor the behaviour of the sensor node. Our Wisebed site is located in the building of the Institute of Computer Science and Applied Mathematics at the University of Bern (IAM). It contains 40 TelosB/TMote Sky sensor nodes. The Wisebed testbed and including the available sensor nodes placed in the IAM building are shown in Figure 2.8.

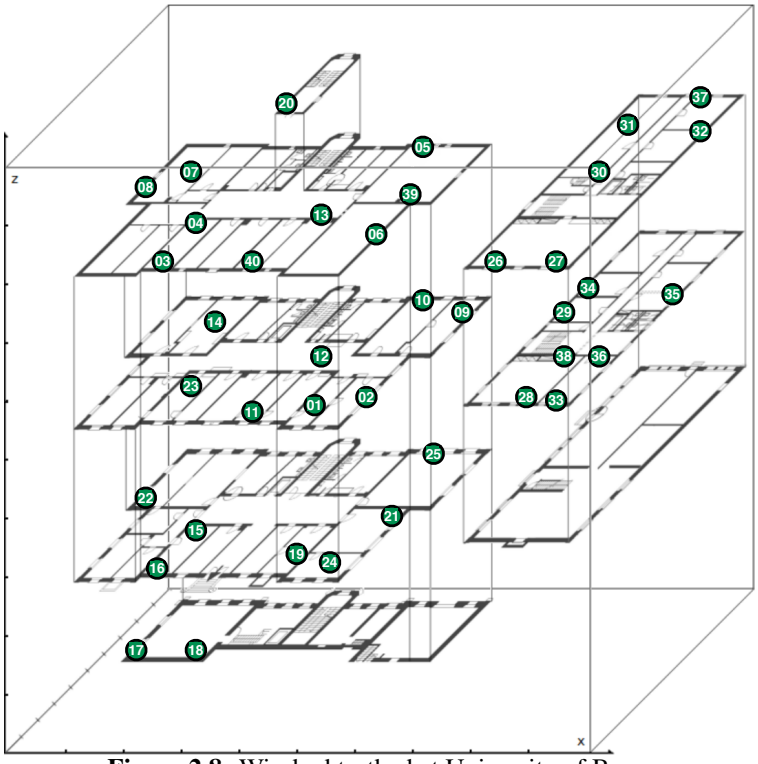


Figure 2.8: Wisebed testbed at University of Bern.

The Wisebed testbed is controlled by the Testbed Management Architecture for Wireless Sensor Networks (TARWIS) software [57, 58, 56, 55, 26]. TARWIS is a Web Services-based management system for the administration and management

of research testbeds of wireless sensor networks. It allows a researcher to manage and administrate experiments using a real-world testbed. A configuration of an experiment contains the duration, the required nodes and the software images for the individual nodes. Additionally, the experiment configuration supports so-called commands to control the behaviour of the sensor nodes during the experiment. To provide exclusive usage of the sensor nodes during an experiment a reservation system has been implemented. Furthermore, researchers can monitor the running experiments and interact with them (using commands).

To store the recorded experiment output and results an XML-based language called Wireless Sensor Network Markup Language (WiseML) [32] is used. Each printed output, which a sensor node writes to the serial interface, including a timestamp, is written to a WiseML file.

TARWIS offers an intuitive and easy-to-use web-based user interface for the reservation of sensor nodes, configuration and monitoring of the experiments and gathering the experiment output.

2.3.3 Energy Measurement

To measure the consumption of energy during a certain time period there are two possibilities. First, a hardware-based energy measurement method. This method is based on attaching an oscilloscope to the power source of a wireless sensor node to measure the current voltage and to calculate the current draw. This method can be very costly and inadequate with increasing size of the wireless sensor network. For the whole wireless sensor network this would be very expensive and not practicable. A second method would be a software-based energy measurement. Such methods are inexpensive and yield accurate results as pointed out in [36]. Contiki has a built-in energy measurement tool called Powertrace [36]. It is a software-based energy measurement, which is implemented as a common protothread and can be started as soon as the energy measurement begins. To measure the energy consumption, Powertrace uses so-called *energest* values [41]. These *energest* values are clock ticks of the used micro-controller. They can be converted to a time value (in seconds) by dividing them by the clock rate of the used micro-controller. In case of a TmoteSky sensor node, which uses an `msp430` micro-controller [117], one second corresponds to 32768 clock ticks. Powertrace provides *energest* values for interesting energy consumers like the CPU or the radio transmitter. To get the *energest* value for the radio communication, Powertrace simply sums up the *energest* values for listening and transmitting of packets during the desired time period. These *energest* values describe how much time in clock ticks was spent for radio communication or common tasks using the CPU.

Based on those *energest* values, the energy consumption can be calculated by converting these values to seconds and multiplying them with the used power in Watt. Multiplying seconds with Watt, which is $\frac{Joule}{Second}$, gives the energy consumption expressed in Joule. The current consumption for certain actions like listening for data or transmitting data can be found in the datasheet of the used component

[30]. For the CC2420 radio transceiver [120] the current consumption for listening is 21.8 mA, and 19.5 mA for transmitting. To convert these values to Watt, they need to be multiplied by the current voltage, which is 3.6 V in case of a TmoteSky sensor node. That amounts to a power consumption of 78.48 mW for listening, and 70.2 mW for receiving.

2.4 Multicast in Wireless Sensor Networks

In this section we are presenting an overview of protocols developed for the support of multicast. While some of these protocols have been designed at the link layer, others implemented on the transport layer, multicast routing protocols are located on the network layer. We are more interested in a classification of the protocols based on the network function that they support. There is a large group of protocols especially designed for multicast routing. We can of course not cover all existing works but we show a representative overview of several important routing protocols in Section 2.4.1. Focus is kept on wireless networks. Routing is not directly related to data completeness and thus reliable data transport. Therefore, we have excluded them as possible candidates to compare SNOMC performance too.

Another big group of protocols we considered are protocols for data dissemination mainly at the transport layer. These would be more appropriate for comparison since their main responsibility is the delivery of data and it can be expected that reliability will be addressed. Not many solutions can be found in the literature that focus on wireless sensor networks. Most multicast protocols with the support of reliable data transfer are designed for wireless mesh or ad hoc networks. Such networks, although relying on wireless transmissions, deal with different terminal characteristics contrary to WSNs, where energy efficiency and limited processing resources form the main constraints on communication. Therefore, we decided to focus on evaluation comparison of traditional protocols for data dissemination in wireless sensor networks, as described in Section 2.5, which (1) reflect better on wireless sensor network specifics even if not supporting data reliability and (2) are more commonly used in test-beds and real-world deployments.

2.4.1 Multicast Routing

In this section we present a number of multicast routing protocols. Although SNOMC leaves out multicast routing aspects and focusses on reliable and energy-efficient multicast transport, multicast routing protocols are presented due to reasons of completeness. Table 2.1 summarises the characteristics of the protocols in terms of the affected protocol layer, the addressed type of network, and support of reliability and energy-efficiency.

The first five presented approaches are especially designed for multicast routing in wireless sensor networks. They neither support reliability mechanisms nor energy-efficiency.

Table 2.1: Overview over multicast routing protocols.

protocol	affected layer	addressed network	reliability	energy
VLM^2 [106]	network/transport	WSN	no	no
[46]	network/transport	WSN	no	no
RBMulticast [43]	network	WSN	no	no
TinyADMR [95]	network	WSN	no	no
AOM [24]	network	WSN	no	no
CGM [64]	network	WSN	no	no
GMR [99]	network	WSN	no	no
GMREE [100]	network	WSN	no	yes
HGMR [62]	network	WSN/MANET	no	no
[132]	network	WSN	no	yes
MERLIN [106]	mac/network	WSN	yes	no
StateSync [49]	network/transport	WSN	yes	no
[63]	network	802.11/MANET	no	no

A multicast protocol for wireless sensor nodes is VLM^2 (Very Lightweight Mobile Multicast) [106]. VLM^2 addresses on routing in that it provides both downstream routing in the form of multicast (and broadcast and unicast) as well as upstream routing in the form of unicast, which suites to the asymmetric topology of sensor networks rooted in a base station. Further, it focusses on mobility of wireless sensor nodes and its lightweight footprint. The evaluation of VLM^2 was made via simulation, but it was not indicated neither which simulator nor which simulation parameters were used. The performance of VLM^2 was also not compared to other protocols. Afterwards, VLM^2 was implemented on real sensor nodes, namely MICA sensor motes [27] to form a mobile multicast-based wireless sensor network. Besides the evaluation, the major drawback of VLM^2 is that there is no reliability mechanism implemented.

In [46] the authors present an effective all-in-one solution for unicasting, anycasting and multicasting in wireless sensor and mesh networks. The paper studies only routing schemes and proposes a distributed approximation algorithm to construct a distribution tree. The paper only describes the algorithm formally and provides no implementation or evaluation of it.

Another multicast routing protocol for wireless sensor networks, called RBMulticast, is proposed in [43]. The authors argue that in wireless sensor networks where traffic is bursty, with long periods of silence between the bursts of data, an apriori creation of multicast trees (where the sensor nodes have to hold state information) adds a large amount of overhead to the sensor nodes for no benefit to the application. Therefore, RBMulticast is a stateless, receiver-based multicast protocol, which simply uses a list of multicast members, integrated in packet headers, to enable receivers to decide the best way to forward multicast traffic. RBMul-

ticast was implemented on TinyOS's TOSSIM [68] simulator and for real-world experiments in TinyOS [66]. The simulation and experimental results showed that the protocol provides high success rates in a stateless operation. The evaluation does not include comparison to other protocols. Drawbacks of RBMulticast are that it is only focussed on routing and thus there are no reliability mechanisms implemented.

The authors of [95] adapt ADMR (Adaptive Demand-driven Multicast Routing), a multicast routing protocol for MANETS, for wireless sensor networks. They showed that adapting such a protocol into real-world implementations on wireless sensor nodes is a challenging task due to the resource-constraint limitations of sensor nodes. TinyADMR was implemented on MicaZ [28] nodes using the TinyOS operating system. The real-world impact of path selection metrics, radio link asymmetry, protocol overhead, and limited routing table size was shown. Since TinyADMR is an multicast routing protocol any reliability mechanisms were not addressed.

The paper [24] addresses source routing for overlay multicast in wireless sensor and ad-hoc networks. They state that the intuitive idea of using multiple unicasts to disseminate the message over the multicast overlay does not take advantage of the broadcast nature of radio transmission. Therefore, they argue that overlay multicast must be matched by the routing and propose Aggregation Overlay Multicast (AOM). The basic idea of AOM is to first build a data delivery tree from the source node to all the member nodes. The intermediate member nodes keep track of the parent and child relationship. When the source multicasts a message, the intermediate member nodes prepare their own header and relay the message through source routing. Thus, routing is stateless. The evaluation of AOM is done using the GloMoSim simulator [7]. They compared AOM with the Differential Destination Multicast (DDM) [61] and showed that AOM improves energy efficiency and balances the energy consumption better. Since AOM is a multicast routing protocol, there are no reliability mechanisms integrated.

The following five protocols can be classified as geographic multicast routing protocols. The construction of the distribution tree is based on the geographic position of the sensor nodes. All protocols do not take reliability into account.

Paper [64] addresses the challenges of multicast applications with large-scale groups in large-scale wireless sensor networks. An important issue for energy saving in geographic multicasting is to obtain location information of destination nodes and the efficient construction of the geographic multicast tree. To solve this problem the whole network is divided into many multiple small areas and a leader node in each area manages the location information of destination nodes in its area. Thus, hierarchical geographic multicast trees have a higher tree, consisting of the source node and the leader nodes, and a lower tree consisting of a leader node and destination nodes in the area of the leader node. The problem is that finding location information of such leader nodes through global location search at a source consumes a lot of energy. To avoid global location search the CGM (Consecutive Geographic Multicasting) protocol proposes a local search based member location

service and consecutive multicast tree expansion algorithm. The location service is efficiently minimizing the searching costs for location acquisitions of multicast members. The consecutive multicast tree expansion algorithm enables to avoid the data detour problem.The authors evaluate CGM using the Qualnet network simulator [10] and show that CGM is more scalable than other protocols in terms of network size and also constructs efficient multicast tree that reduces the data detouring. The protocol only focusses on geographic routing. There is neither a data dissemination scheme proposed nor any reliability mechanism.

The authors of [99] present geographic multicast routing (GMR), a multicast routing protocol for wireless sensor networks. The approach introduces the ideas of previous geographic unicast routing schemes to multicast routing schemes to be able to to efficiently deliver multicast data messages to multiple destinations. Since, GMR only needs information provided by neighbours, it is a fully-localized algorithm. Further, it does not require any type of flooding throughout the network. The main idea of GMR is that each node propagating a multicast data message only needs to select a subset of its neighbours as relay nodes towards destinations. The cost-aware neighbour selection at each routing step is based on a greedy set merging scheme. It allows to find a good trade-off between the optimality of the multicast tree, and the efficiency of data delivery. The evaluation of GRM is based on simulations. The results of the simulation show that GMR outperforms different variants of PBM [75] in terms of cost of the trees and computation time over a variety of networking scenarios. One drawback of the evaluation is that the simulation is based on a perfect MAC layer without any collisions.

The same authors extend GMR addressing energy-efficiency and propose an energy-efficient multicast routing protocol called GMREE. The construction of the multicast trees is based on a greedy algorithm using local information. The selection of relay nodes from the neighbourhood takes not only the minimization of the energy into account but also the number of relays selected. Nodes only select relays based on a locally built and energy-efficient underlying graph reduction such as Gabriel graph (GMREE_GG), enclosure graph (GMREE_EG) or a local shortest path tree (GRMEE_SPT). The evaluation of GMREE is done using a custom-made Java simulator for geographic routing. The simulation results show that the proposed protocol outperforms the traditional energy-efficient multi/unicast routing over a variety of network densities and number of receivers.

Another geographical multicast routing protocol is proposed in [62]. The proposed Hierarchical Geographic Multicast Routing (HGMR) protocol combines the key design concepts of GMR [99] and HRPM [31]. GMR improves the forwarding efficiency by exploiting the wireless multicast advantage as described above, but it suffers from scalability issues when dealing with large sensor networks. HRPM reduces the encoding overhead by constructing a hierarchy at virtually no maintenance cost via the use of geographic hashing but it is energy-inefficient due to inefficiencies in forwarding data packets. The proposed HGMR is a new location-based multicast protocol and is optimized for wireless sensor networks by providing both energy-efficient forwarding efficiency as well as scalability to large

networks. The evaluation of HGMR is done using the GloMoSim simulator [7] and showed that HGMR exhibits the strength of both GMR and HRPM. The authors state that HGMR is optimized for wireless sensor networks but the simulation scenario is using 802.11. This does not reflect the requirements for wireless sensor networks.

Another energy-efficient multicast routing algorithm, which is based on geographic routing is proposed in [132]. The algorithm defines the multicast region as a rectangle that has the smallest area but covers all destination nodes. An access point is selected from the rectangle as root of broadcast tree. The algorithm includes two parts. The first one is to seek a minimal energy path from the sink to the access point based on the idea of dynamic programming. The second one is to search for a broadcast tree between the access point and the destination nodes in the multicast region. The authors evaluate the algorithm analytically and compared it to pure flooding and SAM [134]. The results indicate that the proposed algorithm has better performance on energy consumption and success rate of tree setup.

Both following protocols take reliability mechanisms into account. To support reliability not only the network layer is affected but also the mac layer or the transport layer respectively.

The authors of [98] address the combination of energy-efficient MAC protocols and routing protocols in wireless sensor networks and propose the MERLIN (MAC and Efficient Routing integrated with Support for Localization) protocol. MERLIN integrates MAC and routing features into a single architecture. MERLIN employs a multicast upstream and multicast downstream approach to relaying packets to and from the gateway. On the MAC layer it utilizes Low Power Listening (LPL), Reception and transmission errors are notified by using asynchronous burst acknowledgements (BACK) and negative burst acknowledgements (BNACK). For the proper operation of MERLIN the sensor nodes need to be time synchronized, which leads to a division of the network into timezones. Together with an appropriate scheduling policy, this enables the routing of packets to the closest gateway. MERLIN has been evaluated against S-MAC [133] in combination of the Eyes Source Routing (ESR) [131] protocol using the OMNeT++ simulator. The evaluation showed that the use of MERLIN leads to both significant energy saving and latency reduction with respect to the node duty cycle. This paper [98] focusses on multicast routing and addresses the challenges of using a energy-saving MAC protocol. The reliability mechanisms are left at the MAC protocol. The evaluation only shows results about energy consumption and latency and left out any statements about reliability such as packet loss, etc.

The authors of [49] demonstrate StateSync, an abstraction for reliable dissemination of application state through a multi-hop wireless network. Using StateSync, the complexity of multihop wireless network applications and services can be reduced to processing a gradually evolving set of table entries, subject to minimal consistency checks. The StateSync abstraction defines a data model based on key-value pairs, a reliability model with probabilistic latency bounds, and an event-driven publish/subscribe API. The advantage of StateSync is that it defines a re-

liable transport that can protect a large collection of state variables with a single aggregate refresh. There are three variants of StateSync: SoftState, LogFlood, and LogTree. SoftState is a very simple implementation based on periodic re-flooding of the complete state with no retransmission mechanism. LogFlood introduces a log mechanism to enable publication of updates to existing state and implements a local retransmission protocol, while using a flooding mechanism to push data with low latency. LogTree introduces an overlay network consisting only of the most reliable bi-directional links, and forms distribution trees via that overlay. The evaluation of the three StateSync variants is done using simulation and a wireless testbed in terms of network traffic and latency. The results show that the LogTree variant outperforms the other two variants.

The last presented paper [63] compares two classical multicast routing protocols for MANETs the Multicast Ad-hoc On-Demand Distance Vector (MAODV) protocol and the On-demand Multicast Routing (ODMRP) protocol. While MAODV builds and maintains a multicast tree based on hard state information, ODMRP maintains a mesh based on softstate. The simulation results of this study show that in many scenarios ODMRP achieves a higher packet delivery ratio, but results in much higher overheads. In an environment where the network topology changes very frequently, mesh-based protocols outperform tree-based protocols, due to the availability of alternative paths, which allow multicast datagrams to be delivered to all or most multicast receivers even if links fail.

Most of the multicast routing protocols presented in this section are designed for wireless sensor networks. Although they focus on multicast routing aspects, they do not take reliability into account. In contrast, our proposed overlay multicast protocol focusses on reliable and energy-efficient multicast transport and leaves routing aspects out.

2.4.2 Multicast Transport

In this section we present studies on the topic of multicast transport. Table 2.2 shows the characteristics of the protocols in terms of the affected protocol layer, addressed type of network, and support of reliability and energy-efficiency. There are only few multicast transport protocols designed especially for wireless sensor networks. Therefore, we also choose protocols designed for core networks, for general wireless environments, and WLAN. The purpose of this section is to evaluate, if there are existing multicast transport protocols that can be used as a comparative base for the SNOMC protocol.

The first presented paper describes a reliable multicast protocol for core networks. It has been selected, because it addresses reliability and light-weight sessions.The paper [45] describes Scalable Reliable Multicast (SRM), a reliable multicast framework. Although the framework does not focus on wireless networks it addresses light-weight sessions. The proposed algorithms of this framework are efficient, robust, and scale well to both very large networks and very large sessions. The SRM design includes IP multicast group delivery, end-to-end and receiver-

Table 2.2: Overview over multicast transport protocols.

protocol	affected layer	addressed network	reliability	energy
SRM [45]	transport/application	core	yes	no
RMDP [93]	transport	core	yes	no
RM2 [25]	transport	wireless environment	yes	no
[23]	mac	wireless environment	no	no
MHARQ [71]	transport/application	WLAN	yes	no
SRB [111]	mac	WLAN	yes	no
HCP [80]	transport	MANET	yes	no
RMAC [107]	mac	MANET	yes	no
BAM [106]	mac/transport	WSN	no	yes
RMD [72]	mac/transport	WSN	yes	yes
[109], [108]	network	WSN	no	no

based reliability, and application level framing. It has been prototyped in wb, a distributed whiteboard application, which has been used on a global scale with sessions ranging from a few to more than 1000 participants. The paper has focused on SRM's request and repair algorithms for the reliable delivery of data but it has not proposed a complete set of algorithms for implementing local recovery. The evaluation via analysis and simulation of SRM showed that the performance of a reliable multicast delivery algorithm depends on the underlying topology and operational environment. Based on the results, the authors demonstrate an adaptive algorithm that uses the results of previous loss recovery events to adapt the control parameters used for future loss recovery. With the adaptive algorithm, the reliable multicast delivery algorithm provides good performance over a wide range of underlying topologies.

The next three papers present approaches for multicast transport for general wireless environment. It is not specified, for which wireless environment they are addressed. The authors of [93] present Reliable Multicast Data Distribution Protocol (RMDP), which also focusses on wireless environments. They argue that ARQ-based protocols perform very poorly as the number of receivers grows. Therefore, in the proposed protocol the reliability is based on the use of Forward Error Correction (FEC) techniques. In fact RMDP is a hybrid FEC+ARQ protocol. FEC encoding is used to improve the behaviour of the protocol in presence of large groups of receivers. Further, it reduces the amount of feedback from receivers. ARQ is used to deal with those cases, where the default amount of redundancy does not suffice to complete reception. The performance evaluation is done only analytically and there is not comparison to other protocols. The protocol has been implemented for the use in the MBone, there is no implementation for a wireless environment. However, the authors argue that the protocol is well-suited to the use with mobile equipment because of its simplicity, robustness to losses, moderate

demand for feedback, and scalability.

The authors of [25] focus on reliability of multicast transmissions in wireless environments. Existing multicast protocols adopt a static packet retransmission scheme (unicast or multicast) to retransmit lost packets. This often leads to performance loss due to wasting bandwidth resources. Thus, this paper analyses retransmission costs where two main parameters are used for the comparison: the network load and the amount of duplicate packets generated. Further, the authors mention that a careful continuous monitoring and control is fundamental in wireless mobile environments. Outcome of the analysis was a reliable multicast protocol called RM2. RM2 has been implemented in the ns-2 simulator and later compared to Scalable Reliable Multicast (SRM) for fixed networks. The evaluation showed that RM2 outperforms SRM in terms of link utilization, number of duplicated packets, and arrival times. Limitation of this evaluation is that it mixes fixed networks and wireless networks. Transmission errors are caused by buffer overflow in the routers (main source for Internet packet loss) and transmission packet error rate (reflects the link error probability). Furthermore, the approach focusses on wireless environments but not specially on wireless sensor networks. Therefore, it is not a candidate for a comparison with SNOMC.

The authors of [23] designed transmission strategies for medium access control (MAC) layer multicast. The goal of this work is to maximize the utilization of available bandwidth. Due to the multicast nature of transmissions, the throughput is not equivalent to attaining the stability region of the system or the minimization of loss. The authors show that threshold-based transmission policies maximize the throughput depending on stability and loss constraints. They present an adaptive approach to compute the parameters of the optimum policies without any knowledge of system statistics, the senders only need to know the number of ready receivers in each slot, and not the individual readiness states of the receivers. The limitations of this approach are that it is considered that each packet can be transmitted only once at the MAC layer. Thus, retransmission or any other reliability mechanism are not considered. Further, the evaluation is only made by a numerical analysis. There are no implementations of the approach to prove the performance in real-world against other protocols.

The following two protocols focus on multicast transport in WLAN/802.11. The authors of [71] propose a reliability mechanisms for video multicast over wireless LANs called Merged Hybrid ARQ with staggered FEC (MHARQ). MHARQ combines the advantages of receiver-driven staggered FEC and hybrid ARQ schemes to compensate the large dynamic range of WLAN channels and to achieve high reliability, scalability and wireless bandwidth efficiency for video multicast. Receivers can dynamically join or leave FEC multicast groups based on the channel conditions. In addition, when delayed FEC packets are not enough to recover the lost packets, the receivers can send a hybrid ARQ request (ARQ NACK) to the video server. The protocol has been evaluated using the ORBIT radio grid testbed [91]. Three error recovery schemes for video delivery have been compared: Staggered Adaptive FEC (SAFEC), Hybrid ARQ (HARQ), Merged Hybrid

ARQ (MHARQ). The results show that MHARQ improves wireless bandwidth efficiency and scalability for reliable video multicast, compared with existing reliable multicast schemes.

Also in the area of 802.11 is the following protocol located. The authors of [111] address that the IEEE 802.11 MAC layer does not support reliable multicast, which is the limiting factor in the efficacy of multicast applications. They propose a Slot Reservation based Reliable Multicast (SRB) protocol that adds a reliability components to the existing multicast protocol in the 802.11 MAC. It is based on the existing DCF support in the IEEE 802.11 MAC to seamlessly incorporate an efficient reliable multicast mechanism. The protocol uses RTS-CTS-DATA-ACK exchange with a slot reservation based scheduling mechanism to ensure reliable multicast data delivery. The protocol has been evaluated using the NS-2 [78] simulator and compared to Batch Mode Multicast MAC (BMMM) [115] protocol. The results show that SRB protocol outperforms BMMMM in terms of delivered throughput in various scenarios.

Another group of protocol are designed for multicast transport in MANETs.

The authors of [80] address the challenges of multicast transport of multicast transport in MANETs because the source must provide congestion control and reliability not only for a single but for a distribution tree. In MANETs this problem increases due to contention, spatial reuse, and mobility. To solve the problems, the authors design HCP, a *H*op-by-hop multi*C*ast transport *P*rotocol, which pushes transport functionality into the core of the network. Because HCP uses credit-based, hop-by-hop congestion control, it can quickly determine the appropriate sending rate when network conditions change. One problem, which the authors address is that multicast packets are forwarded using broadcast. This leads to a high packet loss rate when it must contend with TCP traffic, which uses often the RTS/CTS exchange. Instead, HCP uses a form of semi-reliable broadcast. Each time it forwards a packet it chooses one of the one-hop receivers and forwards the packet using the same RTS/CTS exchange as a unicast packet. This ensures that one receiver gets the packet reliably, and the other receivers attempt to receive the packet through overhearing the packet. Thus, many packets are transmitted reliably by the MAC layer. Furthermore, caching provides local recovery and supports different rates within the same multicast tree. The evaluation of HCP is done using the NS-2 simulator. They compare HCP to an application-layer multicast that uses an overlay of ATP [116]. The results show greater efficiency as application-layer multicast.

A reliable MAC protocol called RMAC for wireless ad hoc networks is presented in paper [107]. One of the main ideas of the protocol is the usage of variable-length control frame to stipulate an order for the receivers to respond. With this the problem of feedback collision is solved. Beside this, the usage of receiver busy tone (RBT) for preventing data frame collisions is extended to multicast scenarios. Further, acknowledgement busy tone (ABT) is used to acknowledge the data frames. Additionally, RMAC provides both reliable and unreliable services for unicast, multicast, and broadcast communication. RMAC has been evaluated us-

ing GloMoSim [7]. RMAC has been compared to BMMMM [115] and it has been shown that RMAC not only provides higher reliability but also involves lower cost.

The only two approaches about multicast transport for wireless sensor networks found in the literature are presented as follows.

In [79] a multicast protocol called BAM (Branch Aggregation Multicast) is presented. The approach focusses on multicast for wireless sensor networks. BAM is composed of two aggregation techniques. One is single hop aggregation (S-BAM) and the other is multiple paths aggregation (M-BAM). S-BAM supports single-hop link-layer multicast and is designed to reduce redundant communication at every branch. It aggregates radio transmission within a single-hop and enables just single transmissions to multiple intended receivers. M-BAM supports multi-hop multicast and is designed to reduce the number of branches. It aggregates multiple paths into fewer ones and limits the range of radio transmission. The combination of S-BAM and M-BAM is called SM-BAM. The authors evaluated BAM in three ways, qualitative evaluation by theoretical analysis, quantitative evaluation through computer simulations, and experiments using CrossBow's MICA2 [27]. In the real-world experiments the authors compare the three variants of BAM with Directed Diffusion [60] with B-MAC [86] as underlying MAC protocol and show that they outperform Directed Diffusion in terms of energy-consumption. Limitations of BAM are that they do not provide any reliability mechanisms. BAM inspired us to use aggregation as well as link layer multicast on the branching nodes in the design of SNOMC. The link layer multicast is called broadcast optimization in SNOMC.

The paper [72] addresses the problem of reliability of data dissemination, particularly in the case of total or partial network reconfiguration. The authors propose RMD (Reliable Multicast Data Dissemination) protocol targeting multicast groups of collaborating objects. The protocol focuses on guaranteeing the data transmission rather than improving the delivery ratio. The multicast data dissemination is ensured using tree-based reliable multicast protocols (TRMP) [65]. As reliability mechanism local ACKS and NACK are used to trigger retransmissions. Aggregated ACKs indicate correct overall reception to the source. To improve reliability a cross layer design is proposed. The MAC layer provides neighbourhood information, local (one-hop) ACKs and retransmissions (timeout or ARQ based) and interaction points (callbacks) for signalling local ACKs to the dissemination/transport layer. The dissemination protocol fragments the message, ensures end-to-end delivery through aggregate ACKs, and controls the MAC layer for the listening phase in order to save energy. RMD has been evaluated analytically and through simulation. The results show the benefits of using cross-layer interactions between dissemination and MAC. Further the authors compare RMD with PSFQ [129] using the OMNeT++ simulator. It has been shown that RMD ensures the data delivery to all recipients even under high error rates, while consuming 2-3 times less energy, and maintaining a comparable delay. Similar to SNOMC, RMD also uses NACKs to request retransmissions. RMD would be a fitting candidate for a comparison with SNOMC. But there is no real-world implementation available we left it out and focussed on the comparison with common data dissemination protocols

(see Section 2.5).

The last presented paper does not present a protocol but a study about using IP multicast in wireless sensor networks.

The authors of [109] and [108] address general challenges and benefits of using IP and IP multicast in wireless sensor networks. They made an evaluation from two multicast routing protocols, namely MAODV (Multicast Ad hoc On-Demand Distance Vector) [96], SSM (Source-Specific Multicast) [17], one unicast routing protocol, namely AODV (Ad hoc On-Demand Distance Vector) [82], and pure broadcast. They used the NS-2 [78] simulator and showed that both multicast protocols outperform the AODV and pure broadcast in terms of number of forwarded packets and energy consumption. The results clearly point to the benefits of the use of IP multicast in processing and energy-restricted environments such as in wireless sensor networks. This paper is just a study about the potential usage of IP multicast in wireless sensor networks. There is no own approach for a multicast protocol.

The presented multicast transport protocols are mainly designed for other networks as wireless sensor networks. It can be stated that besides BAM and RMD no multicast transport protocols for wireless sensor networks can be found in literature. RMD is the only protocol which addresses the same problems as SNOMC does (reliability and energy-efficiency). Thus it would be the only candidate for a comparison with SNOMC, our overlay multicast transport protocol. Due to there is not real-world implementation of RMD at the time of the SNOMC evaluation available, we decided to evaluate SNOMC against common data dissemination protocols, such as Flooding, Multipoint Relay (MPR), Directed Diffusion, and PSFQ. These data dissemination protocols are presented in the next Section 2.5.

2.5 Data Dissemination Protocols

In this section we present a subset of the many data dissemination protocols existing in the literature on wireless sensor networks. These protocols has been selected for the evaluation of SNOMC, because they are common data dissemination protocols used in wireless sensor networks.

2.5.1 Directed Diffusion

One data dissemination scheme, commonly used in wireless sensor networks, is Directed Diffusion [60]. This communication protocol follows different principles than other protocols. The main difference is the data-centric routing in that all communication is for named data. Data generated by sensor nodes is named by an attribute-value pair. Directed diffusion consists of several elements: naming, interest propagation, gradient establishment, data propagation, reinforcement, and repair.

Naming is realized as follows. The task descriptions are named by a list of

attribute-value pairs that describe a task. Attributes are, e.g. the *type* of data, *interval*, *duration*, and *rectangle*. The *type* of data describes the type of the data to be collected. The *interval* attribute indicates the event data rate. The *duration* describes the time period of the data collection. The *rectangle* attribute describes the area of interest.

The task description leads to an **Interest Propagation**. An Interest is transmitted from a sink node. For each task the sink periodically broadcasts the Interest. Figure 2.9 shows the propagation of Interests within the sensor network.

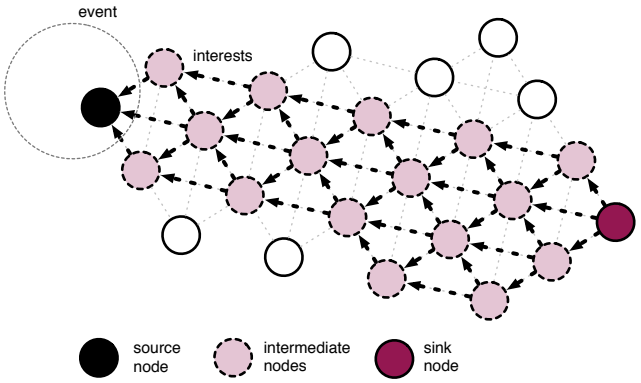


Figure 2.9: Directed Diffusion: interest propagation.

Every node maintains an interest cache, where each item in the cache corresponds to an interest. Interest entries in the cache do not contain information about the sink but about the immediate previous hop. An entry in the interest cache has several fields. A *timestamp* field indicates the timestamp of the last received matching interest. Further, it contains an *interval* field and an *expiresAt* field. Additionally, it contains one or more *gradient* fields, one field per neighbour. Each gradient contains a field *datarate*. The value for the *datarate* is requested by the specified neighbour and derived from the *interval* attribute of the interest. It also contains a *duration* field, derived from the *timestamp* and *expiresAt* attributes of the interest. It indicates the approximate lifetime of the interest.

When an interest arrives at a node, it checks if the interest exists in the cache. If not, the node creates an interest entry. The fields from the interest cache entry are taken from the receiver interest. This entry has a single gradient for the neighbour from which the interest was received including the specified event data rate. If there exists an interest entry, but no gradient for the sender of the interest, the node adds a gradient with the specified value. Further, it updates the entries and fields for the *timestamp* and the *duration* accordingly. If there exists both an entry and a gradient, the node simply updates both fields.

Gradient Establishment happens after the interests are flooded through the wireless sensor network. Each pair of neighbour nodes establishes a gradient for each other. Further, a gradient specifies both a data rate and a direction in which to send events. Thus, the interest propagation sets up routing information in the

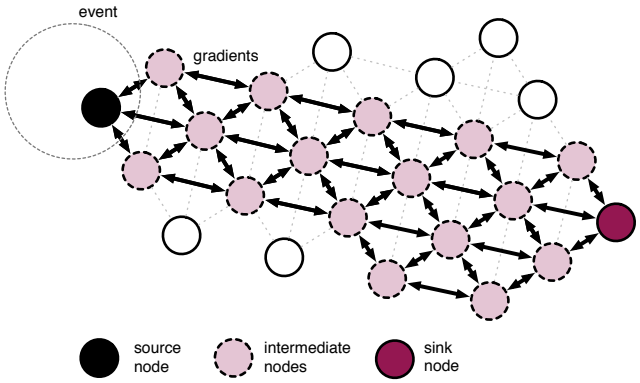


Figure 2.10: Directed Diffusion: gradient establishment.

wireless network sensor network to transmit data towards the sink. Figure 2.10 shows the established gradients.

The next step is **Data Propagation**. A sensor node located within the *rectangle* propagated by the interest starts sensing and collecting the samples in the given interval. If an event is detected the interest cache is checked if there is a fitting interest for this *type* of event. Then it creates a *data* message, which will be transmitted to the relevant neighbour using unicast transmission. A node that receives a data message from its neighbours tries to find a matching interest entry in its data cache. This cache keeps track of recently seen data items to avoid loops. To determine the data rate of an event the sensor node can determine this checking the stored data in the data cache. To forward a received data message, a sensor node needs to examine the matching gradient in the interest cache.

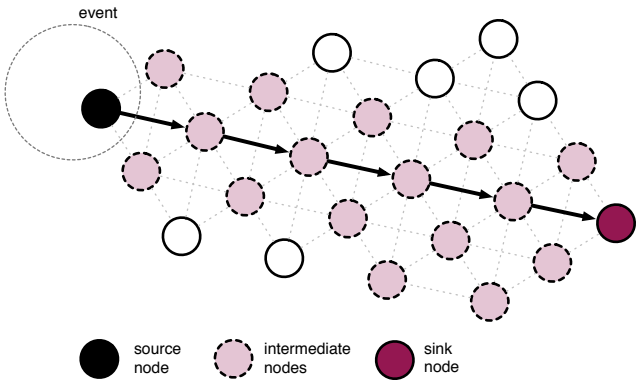


Figure 2.11: Directed Diffusion: reinforcement.

The path from the source to the sink can be established using **Reinforcement**. The data from the sources to the sink arrive at the sink via potentially different paths. Thus, the different paths cause high traffic and lead to collisions and lost packets. To avoid this the sink reinforces the neighbour from which the sink wants to get the data by resending the original interest message but with a smaller *interval*

(higher data rate). When the neighbour node receives this interest, it notices that it already has a gradient for this neighbour but with a lower data rate. Thus, this node must also reinforce at least one of its neighbours. The so selected path has a low-delay. An example of reinforcement is shown in Figure 2.11.

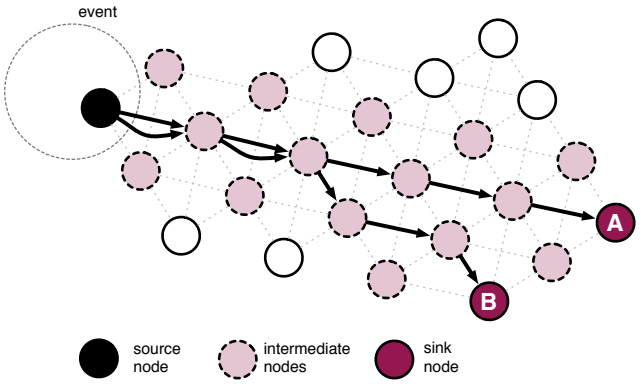


Figure 2.12: Directed Diffusion: multiple sinks.

When two sinks express identical interests, interest propagation, gradient establishment, and reinforcement work in the same way as described above. The situation is shown in Figure 2.12. Sink **A** has already reinforced a path to the source. When another sink **B** propagates an identical interest, the new sink can also use reinforcement to establish the path towards the source. Directed Diffusion has no rule to combine paths when they go (partly) through the same sensor nodes.

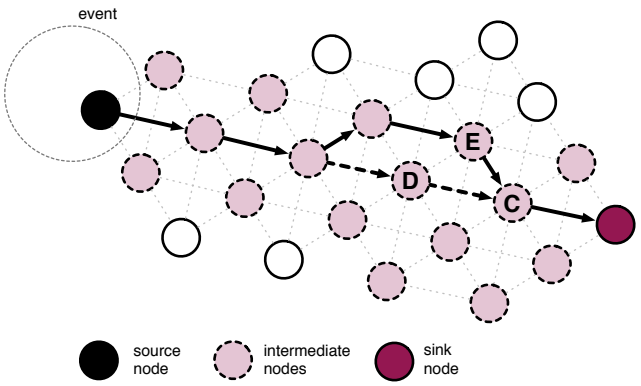


Figure 2.13: Directed Diffusion: local repair.

If an intermediate node on the established path fails, a **Repair** of the path is needed. The situation is shown in Figure 2.13. Sensor node **C** detects a poor link quality from its neighbour node **D** by detecting that the data rate decreases. To solve the problem sensor node **C** reinforces the neighbour node **E**. Sensor node **E** then also reinforces its neighbour node and so a new path has been established.

2.5.2 Pump Slowly, Fetch Quickly (PSFQ)

Contrary to many data dissemination protocols that focus on many-sensors-to-one-sink communication PSFQ [129] addresses the case of one-sink-to-many-sensors communication. This communication direction is more common for network management tasks such as code updates and configuration of sensor nodes. While the sensors-to-sink path may tolerate information loss due to the highly correlated data, the reverse path requires end-to-end reliability to ensure the successfully dissemination of, e.g., a code update to each sensor node. The PSFQ protocol provides three main operations: pump operation, fetch operation, and status reporting.

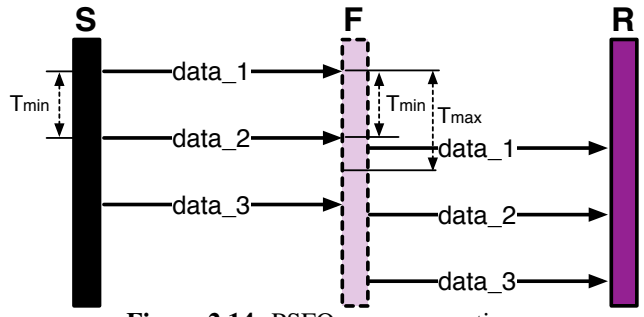


Figure 2.14: PSFQ: pump operation.

Since reliability is generally more important than transmission time, the main idea of PSFQ's **pump operation** is to *slowly* inject packets into the network to avoid congestion situations. The operation is shown in Figure 2.14. A sensor node **S** transmits packets to its neighbour node **F**. Sensor node **F** relays this information to sensor node **R**. To schedule the transmission times of the sensor nodes along the path two timers are used. A sensor node *broadcasts* packets every T_{min} to its immediate neighbours. After sensor node **F** receives a packet it relays it after a random wait time, which is selected between T_{min} and T_{max}. This duration between packet transmissions allows sensor nodes to recover missing packets. Moreover, the random delay allows a reduction in the number of redundant broadcasts of the same packet by the neighbours. If the packet is forwarded by one of the nodes, other neighbours suppress their transmissions.

In case of packet errors, PSFQ's **fetch operation** performs aggressive hop-by-hop recovery at each sensor node to fetch the lost packets from neighbour nodes. The fetch operation is illustrated in Figure 2.15. When sensor node **F** detects a lost packet, it immediately broadcasts a NACK message to its neighbours. If the NACK message gets lost and sensor node **F** does not receive any reply within a period of T_r, where T_r ¡ T_{max}, it continues sending NACK messages. If sensor node **S** has the packet cached, it rebroadcasts the packet to sensor node **F** in an interval between $(1/4)T_r$ and $(1/2)T_r$. To prevent message implosion, PSFQ limits NACK message transmission to the one-hop neighbourhood. Thus, NACK messages are not transmitted via multiple hops. PSFQ relies on the sequence number of the transmitted packets to detect a loss packets. Since, a data message does not include the overall

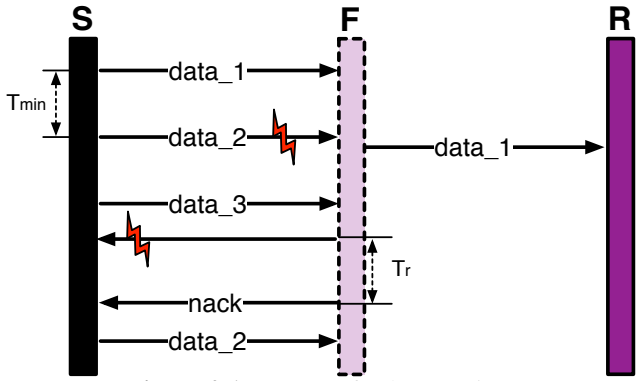


Figure 2.15: PSFQ: fetch operation.

number of packets in the flow, a loss of the last packet can not be detected. For this case, PSFQ employs a proactive fetch operation, where the receivers follow a timer-based fetch operation. The operation is shown in Figure 2.16. Sensor node **F** does not receive data packet number three within a time period T_{pro}, it broadcasts a NACK message for this packet to its neighbours.

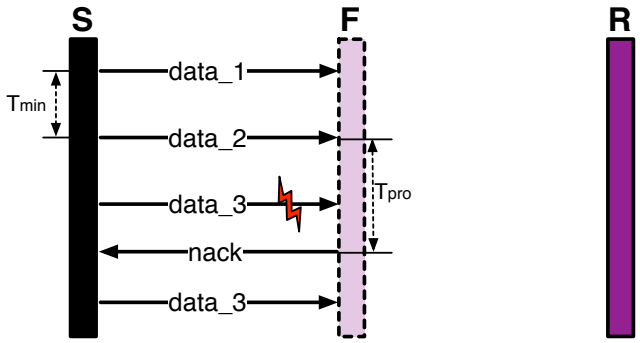


Figure 2.16: PSFQ: pro-active fetch operation.

Finally, there is the **report operation**, which introduces an end-to-end error control mechanism. The sink node requests feedback from the sensor nodes by setting a report bit in the packet header. When the sensor nodes receive this packet, they respond immediately with a report message. The report message is forwarded towards the sink node and each node on the path adds its status information to this report message.

To summarize, PSFQ provides with its pump operation an effective mechanism to limit congestion. Although, the fetch operation introduces hop-to-hop reliability, end-to-end reliability is not provided.

2.5.3 Flooding

Several strategies can be used for data delivery from one sender node to many receiver nodes. The simplest strategy is flooding, where data is transmitted using

broadcast communication mechanisms.

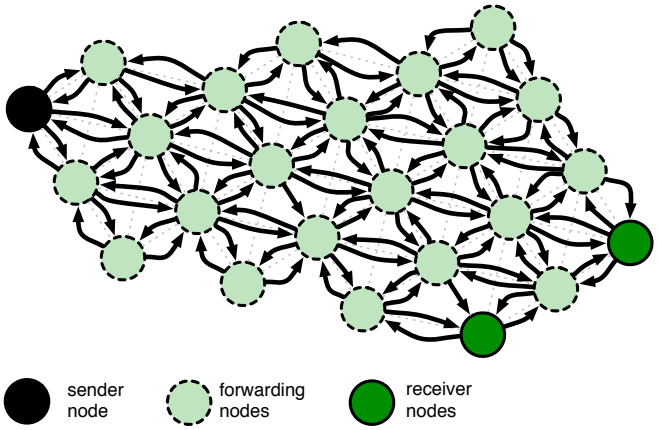


Figure 2.17: Flooding.

An example of flooding is shown in Figure 2.17. The sender starts to broadcast the first packet. Each intermediate node re-broadcasts the packet until the packet arrives at the receiver nodes. To avoid infinite re-broadcasting of packets, resulting in a broadcast storm, loop prevention is needed. The loop prevention mechanism for data messages is very simple. Each packet has an identifier (sequence number). If a packets arrives at an intermediate node and was already forwarded it has been dropped.

Flooding is, however, inherently very inefficient, energy consuming, and unreliable.

2.5.4 MPR

Multipoint Relay (MPR) is a data dissemination protocol, which is also broadcast-based like Flooding. In contrast to Flooding, MPR tries to reduce the number of forwarding intermediate nodes with the goal of reducing the amount of collisions. In general it can be said that less collisions lead to a faster data transmission.

In MPR only a subset of sensor nodes (so called multipoint relays) rebroadcast messages. The relays are chosen based on local knowledge at each node of its two-hop neighbourhood. Each sensor node calculates its own set of relay nodes, that is the set of its one-hop neighbours, reaching most of its two-hop neighbours.

Starting from sensor node **0** the set of relay nodes has to be found to reach the nodes **19** and **23**. Via sensor node **1** sensor node **0** can reach one two-hop neighbours (node **4**) and via sensor node **2** it can reach two two-hop neighbours (nodes **5** and **6**). Most two-hop neighbours can be reached via node **3**, namely nodes **4**, **5**, **6**, and **7**. Therefore, sensor node **3** joins the set of relay nodes and selects itself as the next one-hop neighbour with which the most two-hop neighbours can be reached. The next node in the relay set would be node **7**. Node **16** is required to reach node **17**. Figure 2.18 shows the result of the algorithm.

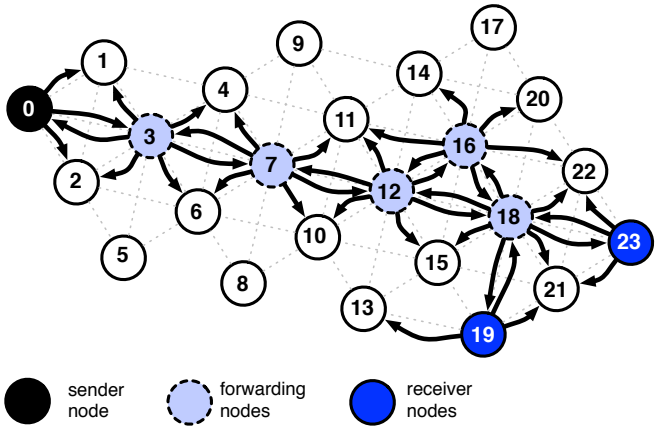


Figure 2.18: Multipoint Relay.

As preliminary requirement for the algorithm each node has to know its one-hop and two-hop neighbourhood. To gather this information each node broadcasts Hello messages for a certain interval. A node, which receives this message knows that the sending node is a one-hop neighbour. Then it sets flag *rebroadcast* to indicate that it forwards the hello message and sets the address of the original node. All nodes receiving the rebroadcasted message know that the this node is in the two-hop neighbourhood.

Similarly to Flooding, Multipoint Relay (MPR) does not support any reliability mechanisms such as retransmission of lost packets.

TinyCubus

TinyCubus [73], [74] is an adaptive cross-layer framework for wireless sensor networks. It is also broadcast-based and deploys a role-based code distribution algorithm using cross-layer information such as role assignments to decrease the number of messages needed for code distribution to specific nodes. While the management part of the framework is discussed in Section 2.7 we discuss the data dissemination protocol of TinyCubus in this section.

The data dissemination protocol is an integral part of the role-based code distribution algorithm. It is very simple and requires some assumptions. The authors discuss in [73] that in wireless sensor networks the roles of all the sensor nodes are defined in advance. Thus, a message can be broadcasted through the wireless sensor network not affecting all nodes of the network, but only a subset of them. Further, the authors define that the nodes with the same role have to be connected that all nodes with the same role can be reached. An example is shown in Figure 2.19. The nodes with the same roles build a group and have the same colour. The receiver nodes are also in the same group. Only the nodes in the same group forward a message. Starting from the gateway node, the message is broadcasted and the forwarding nodes rebroadcast the message.

The authors make several important assumptions. First, the roles of the nodes

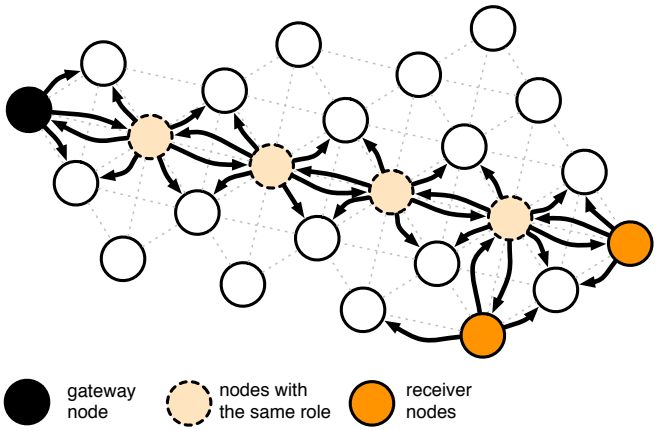


Figure 2.19: TinyCubus.

are assigned and the nodes have knowledge about their neighbours before the data dissemination starts. Following this assumption we can take an optimal set of nodes with the same role to reduce the number of hops from the gateway to the receivers. Secondly, the communication is done using broadcasts. In addition, if reliability is necessary the distribution algorithm uses implicit acknowledgements. If a sensor node transmits a message the next node receives the message and forwards it. The forwarded message can be overheard by the previous node. If the neighbour node does not forward the message within a certain time, the message will be retransmitted.

2.6 Contiki Protocol Stack

In this section we present the protocol stack of the Contiki OS, because the real-world implementation of SNOMC is integrated into the Contiki protocol stack. The protocol stack includes two important MAC protocols, namely NullMAC and ContikiMAC. Additionally we present BEAM (Burst Aware Energy Efficient Adaptive MAC) protocol, which is an energy-efficient MAC protocol integrated in the Contiki OS protocol stack. BEAM has been developed in the CDS group of the University of Bern. In the centre of the Contiki protocol stack the µIP stack and the RIME protocol stack are placed at the network layer. The µIP stack includes an UDP and TCP implementation. As data dissemination protocols we present a number of commonly used protocols for wireless sensor networks, namely Directed Diffusion and PSFQ (Pump Slowly, Fetch Quickly). Further, we describe Flooding as simplest possible data dissemination protocol and with MPR (Multipoint Relay ...) an optimization of flooding. We present also TinyCubus because it combines a management framework for wireless sensor networks and a broadcast-based data dissemination protocol.

2.6.1 Link Layer Protocols

In Contiki a number of common MAC protocols are implemented. Examples are X-MAC [21, 114] and its successor ContikiMAC [33] as well as NullMAC as a minimalistic MAC protocol. In general, a MAC protocol is mainly responsible for sharing the transmission medium between multiple nodes that want to access the medium at the same time. Besides this, the main functions are frame delimiting and recognition, addressing, transfer of data from upper layers, error protection and correction. MAC protocols for wireless sensor networks must be additionally energy efficient to maximize the lifetime of the nodes. Radio communication is typically the biggest consumer of energy. Energy consumption for reception (including idle listening) is often as high as for transmission. Therefore, the MAC protocol must turn off the radio as often as possible, while being awake long enough to receive packets from other sensor nodes. Contiki splits the traditional MAC protocol into two smaller, task-specific protocols:

- a MAC protocol: responsible for accessing the transmission medium using CSMA/CA
- a RDC protocol: the Radio Duty Cycle (RDC) protocol is responsible for power saving.

In terms of Contiki, the MAC protocol is responsible of checking the transmission medium whether someone else is transmitting data or not. This mechanism is called Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA). If the transmission medium is free, the data can be transmitted. If the transmission medium is occupied, the MAC protocol has to wait for a certain time and then to try again to send the data. The other task of the MAC protocol is handling the radio duty cycling. This mechanism is called Radio Duty Cycle (RDC) protocol. The RDC protocol takes care of the energy saving mechanisms. This includes turning off the radio transmitter as much as possible without decreasing the transmission performance too much.

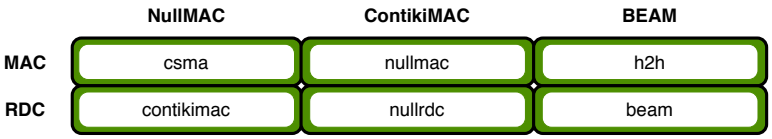


Figure 2.20: MAC and RDC protocols of NullMAC, ContikiMAC and BEAM.

The separation of a common MAC protocol into a MAC and a RDC part can be very confusing (cf. Figure 2.20). `NullMAC`, `ContikiMAC` and `BEAM` also consist of a MAC and a RDC part. The MAC part of `NullMAC` is called *nullmac*, and the RDC part is called *nullrdc*. For `ContikiMAC`, the MAC part is called *csma*, and the RDC part is called *contikimac*. In case of `BEAM` the MAC part is called *h2hr* (hop-to-hop reliability), and the RDC part is called *beam*.

NullMAC

NullMAC is the simplest possible MAC protocol, which is implemented in Contiki. Both the MAC part (*nullmac*) and the RDC part (*nullrdc*) are very simple. *nullmac* only checks, if the transmission medium is free to send the packet. If the transmission medium is occupied, the packet is dropped without any attempt to resend. Analogously, *nullrdc* never turns off the radio transmitter. The packets are transmitted with the maximum power but also with the maximum energy consumption.

ContikiMAC

ContikiMAC is an energy optimized MAC protocol and is the default MAC protocol of Contiki since release 2.5. It is the successor of X-MAC. ContikiMAC supports energy saving mechanisms, such as periodic wake-ups, wake-up strobes and phase-lock optimization. These energy saving mechanisms are implemented in the RDC protocol (*contikimac*). A sensor node using ContikiMAC turns on the radio transmitter periodically only for a very short time (*periodic wake-up*). Within this time period, *contikimac* checks the radio channel. This is called Clear Channel Assessment (CCA) and uses the Received Signal Strength Indicator (RSSI) to give an indication for radio activity on a certain channel. If the RSSI value is higher than a given threshold, the CCA is positive and a packet transmission from another sensor node is detected. In this case the radio transceiver keeps turned on until the whole packet has been received. To acknowledge the successful reception of a packet, the receiver transmits an acknowledgement back the sender and turns off the radio. If the sender transmits a packet and the receiver's radio transmitter is turned off at the same time, the transmitted packet gets lost. To avoid this the receiver uses *wake-up strobes* to notify an upcoming transmission. If the sender wants to transmit a broadcast packet, it has to be rebroadcasted for the full wake-up interval. Thus, the sender can be sure that each neighbour node has once turned on the radio transceiver within this time period and can receive the wake-up strobe. Figure 2.21 and 2.22 illustrate these two situations.

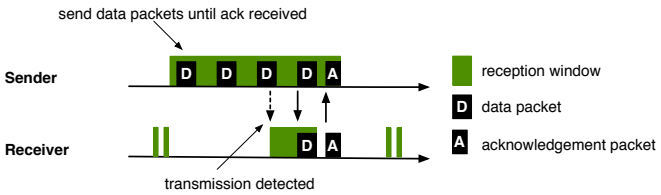


Figure 2.21: ContikiMAC: unicast transmission[33].

In the third energy-saving mechanism, namely *phase-lock optimization*, the sender tries to learn when a certain receiver wakes up. Since the wake-up of a certain receiver is periodic, the sender can estimate the next wake-up of the receiver

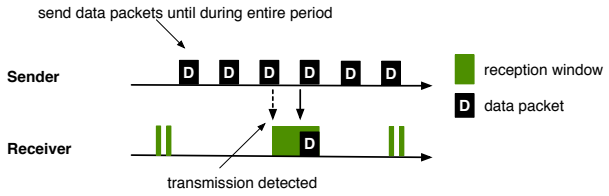


Figure 2.22: ContikiMAC: broadcast transmission[33].

after the first successful packet transmission. With this estimated sleep interval, the sender is able to time the first transmission of a wake-up strobe more accurately.

BEAM

Burst-Aware Energy-Efficient Adaptive MAC (BEAM) [13] protocol is an energy efficient link layer protocol designed to maximize the lifetime of a wireless sensor network. BEAM employs all advanced features of the energy efficient IEEE 802.15.4 compliant radio module CC2420 to optimize energy efficiency. To save energy it uses adaptive and unsynchronized radio duty cycle periods. The radio transceiver is turned on periodically to check the radio channel for incoming packets. Between these channel checks, the radio module is turned off. The wake-up periods of the different sensor nodes are unsynchronized. Thus, no synchronization messages are required. It was designed to operate under variable traffic load conditions by dynamically adapting the duty cycles. To support hop-to-hop reliability BEAM implements an ARQ mechanism. BEAM has two operation modi: using short beacon strobes and using beacon strobes including payload.

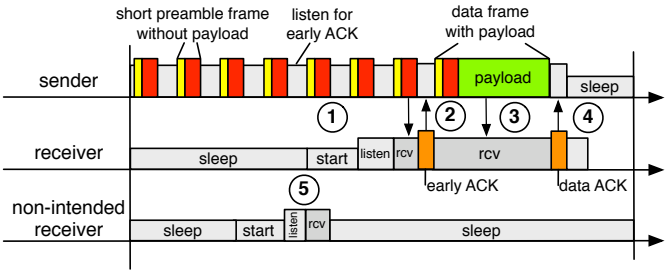


Figure 2.23: BEAM: using short beacon strobes [13].

The BEAM modus using **short beacon strobes** is depicted in Figure 2.23. The sender periodically transmits short beacon strobes to indicate transmission intention and waits for an early ACK frame **(1)**. The receiver wakes up, listens to the channel, and with the destination address stored in the short preamble frame header, it recognizes that it is the intended receiver. Then, it transmits back the early ACK frame **(2)**. Upon reception of the early ACK frame, the transmitting node is aware of the intended receiver being awake and transmits the data frame **(3)**. Successful transmission is acknowledged by the receiver via a data ACK frame **(4)**. Non-intended receivers can thus go to sleep much earlier as in the second

operation mode, because they know that they are not the intended receivers **(5)**. The main disadvantage here is that at least four transmissions are required in the optimal case.

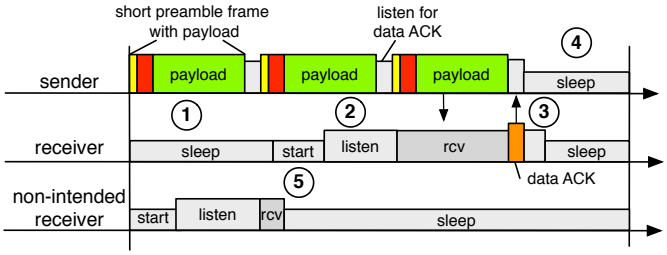


Figure 2.24: BEAM: using beacon strobes including payload [13].

The second mode of BEAM using **short beacon strobes including payload** is shown in Figure 2.24. The sender node transmits periodically short beacon strobes including payload **(1)**. The receiver wakes up and listens to the channel **(2)**. Since the destination address is stored in the short preamble frame header, it can recognize that it is the intended receiver and sends a data ACK frame **(3)**. The sender receives the data ACK frame, stops transmitting the data frame, and goes to sleep **(4)**. If the sender does not receive the data ACK (e.g. due to interferences) it continues transmitting the short preamble frames including payload. In the optimal case, two transmissions for each transmitted MAC frame are needed. Clearly, longer duty cycles (as long as the data frame) of the receiver affect the energy-efficiency negatively. The non-intended receivers then have to listen longer to conceive that the data is not destined for them **(5)**.

2.6.2 Network Layer Protocols

In this section we describe Contiki's µIP stack and Contiki's RIME stack.

Contiki µIP Stack

Part of the Contiki protocol stack is a lightweight TCP/IP stack called µIP, shown in Figure 2.25. Its architecture was especially developed to handle the hardware constraints of 8-bit or 16-bit micro-controllers. Despite its small code size and small memory footprint, µIP attends the subset of RFC1122 [18] needed for full host-to-host interoperability and supports TCP flow control, fragment reassembly, and retransmission time-out estimation [34]. To use as little memory as possible, µIP uses one global packet buffer. This buffer holds incoming and outgoing packets but can store only one packet at a time. Therefore, packet buffering has to be implemented either by the underlying MAC protocol or in the application using µIP. Another challenge is the calculation of 32-bit checksums used in TCP and UDP protocol headers. Calculating a 32-bit checksum on an 8-bit architecture takes many more clock cycles than on a 32-bit architecture, since the 32-bit checksum has to be stored in the main memory and cannot be processed directly in the 8-bit

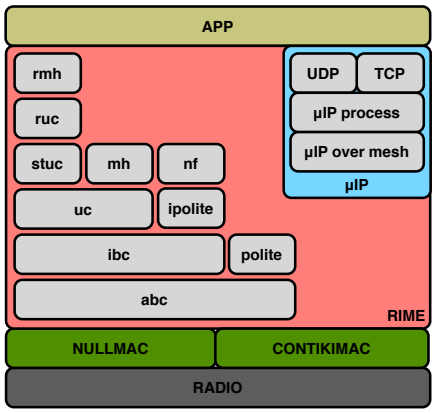


Figure 2.25: The protocol stack in Contiki.

wide CPU registers. µIP solves this challenge by dividing its functionality into two parts: one generic part and one architecture specific part.

A number of Internet protocols are implemented on top of the µIP protocol suite. Besides IP [89] and the two major transport protocols TCP[22] and UDP[87], and also ARP[85], SLIP[94], and ICMP[88] have been implemented. Figure 2.26

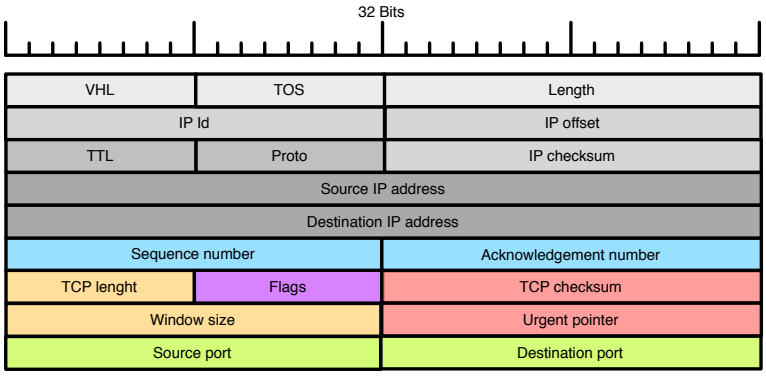


Figure 2.26: TCP and µIP header.

shows the TCP and µIP headers. The headers include the fields for the IP address and the port of the source and destination. Furthermore, it includes the IP and TCP checksums. The main feature of TCP is the end-to-end reliability. Each packet has to be acknowledged using the sequence number and the acknowledgement number. The windows size is used for congestion control. 16-bits are reserved to store the length of the TCP packet.

The TCP implementation in µIP has some limitations compared to a ”standard” TCP implementation like TCP Reno. Due to the constraints of available space some mechanisms for packet delivery to the application are not implemented, for instance, the soft error reporting mechanism and dynamically configurable type-of-service bits for TCP connections. Such functionalities are not usually used by the applications and skipping them would not cause a big impact in a real scenario.

Furthermore, due to the µIP stack is not able to process more than one packet at the same time, the transmission of multiple unacknowledged TCP segments is not supported and the congestion window is set to the value of one. This leads to a decrease of the achievable throughput but prevents nodes from allocating precious memory for unacknowledged segments.

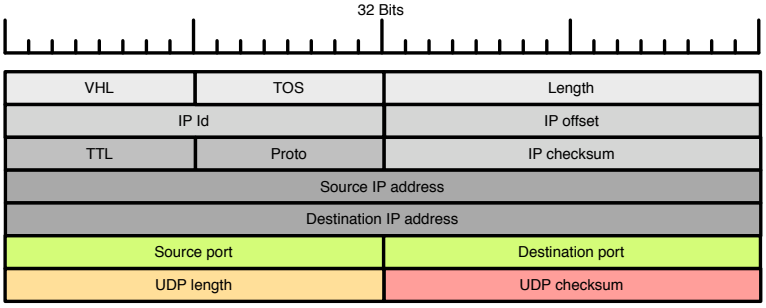


Figure 2.27: UDP and µIP header.

Contrary to TCP, UDP has no support for end-to-end reliability. It is connectionless and does not retransmit packets as TCP does. Therefore, it has a lower complexity. This simplicity helps to use it on wireless sensor nodes. The UDP implementation in Contiki has basically the same header of the Internet standards and is shown in Figure 2.27. The header includes among others the source and destination ports, the checksum and 16 bits to store the package length.

µIP was implemented as a protothread on top of Contiki. Therefore, it uses events to signal the arrival of new packets. To use µIP, the application protothread has to open a new socket for the desired IP address. µIP then stores the process ID (PID) of the application protothread to the connection state data structure. Every time a new packet arrives, µIP posts a `tcpip_event` to the registered protothread, which is unblocked by the scheduler as soon as it is its turn. Listing 2.1 shows how to open a UDP unicast connection to port 2500 of the host with the IP address `192.168.1.20` and the transmission of one UDP packet with the payload HELLO:

Listing 2.1: Sending an UDP packet

```
static struct uip_udp_conn *udp_conn;
static uip_ipaddr_t ipaddr;
static char* msg="HELLO";
uip_ipaddr(&ipaddr, 192,168,1,20);

udp_conn = udp_new(&ipaddr, UIP_HTONS(2500), NULL);
uip_udp_packet_send(udp_conn, &msg, sizeof(msg));
```

The data structure `uip_udp_conn` stores the PID of the application protothread, to which the `tcpip_event` is posted. Additionally, the destination port and destination IP address is stored by this data structure. The µIP function `udp_new`

opens the UDP unicast flow and the function `uip_udp_packet_send` triggers the transmission of the data found at the supplied memory address *&msg*. Listing 2.2 illustrates the code on the receiver part for listening for a `tcpip_event` and processing the arrived packet:

Listing 2.2: Receiving an UDP packet

```
while(1){
  PROCESS_WAIT_EVENT_UNTIL(ev == tcpip_event)

  if (uip_newdata()){
    uint8_t len = uip_datalen();
    void    *ptr = uip_appdata;
  }
}
```

The statement `PROCESS_WAIT_EVENT_UNTIL(ev == tcpip_event)` blocks this protothread until a `tcpip_event` arrives. The pre-processor macro `uip_newdata()` checks if a new packet arrived, and the `uip_datalen()` function returns the size of the arrived data in bytes. `uip_appdata` is a void pointer to the packet payload.

Contiki RIME Stack

Contiki RIME [35, 40] is a lightweight layered communication stack for wireless sensor networks and is shown in Figure 2.25. The RIME stack differs from traditional network stacks in such a way, that each of its layers is very thin. It contains of different node-to-node communication services. These services are implemented in a hierarchical manner with lower layer services providing basic features and more complex services building on top of them. Each layer introduces very limited functionality and accomplishes only one certain task, which leads to a lower implementation complexity. RIME uses very short headers from one layer to the next. A connection between two sensor nodes is usually established by copying the data to the RIME buffer and initiating the actual transmission. The RIME stack offers methods for unicast and broadcast communication, that are optimized for the needs of embedded systems. These are the methods:

- **abc**: Anonymous Best-effort Single-hop Broadcast protocol
- **ibc**: Identified Best-effort Single-hop Broadcast
- **uc**: Best-effort Single-hop Unicast protocol
- **stuc**: Stubborn Single-hop Unicast protocol
- **ruc**: Reliable Single-hop Unicast protocol
- **mh**: Best-effort Multi-hop Unicast

Table 2.3: Overview over management frameworks.

framework	monitoring	configuration	code update/reprogramming
Mate [67]	no	no	yes
MANNA [97]	yes	yes	no
TinyCubus [73]	no	yes	yes

- **rmh**: Hop-by-hop Reliable Multi-hop Unicast
- **polite**: Polite (Anonymous) Single-hop Broadcast protocol
- **ipolite**: Polite (Identified) Single-hop Broadcast protocol
- **nf**: Best-effort Network Flooding

One possible realization is to use the RIME stack as a standalone communication stack without µIP. Another possibility is to use µIP on top of RIME. The former reduces communication overhead, but in this case there is not support of Internet communication. The default configuration of Contiki is using RIME in combination with µIP. IP packets are tunnelled over the RIME stack, which results in an additional 2 byte header for each IP packet. The combination of RIME and µIP gives a strong flexibility. To communicate with any device in the Internet µIP can be used. For lightweight communication that only takes place inside the wireless sensor network the RIME stack can be used, e.g., for neighbourhood discovery.

2.7 Management of Wireless Sensor Networks

In this section we discuss different aspects of the management of wireless sensor networks. In the following, first we discuss between general management frameworks and later focus on the aspect of code dissemination.

2.7.1 Management Frameworks

In this section we only consider management frameworks for deployed wireless sensor networks. Management frameworks for testbeds are described in Section 2.3. Table 2.3 shows an overview of the selected management frameworks.

Mate

The authors of [67] present Mate, a tiny communication-centric virtual machine designed for sensor networks. Sensor nodes have to be reprogrammable over the air, because nodes may be physically unreachable. Reprogramming nodes, thus, code dissemination of the code, can be a significant factor for energy costs. Focus of this paper is not the code dissemination but the reduction of the code. The

main idea of Mate is to provide a high-level programming interface allowing implementing complex programs very short (under 100 bytes). Code is broken into small capsules of 24 instructions, which can self-replicate through the network. The code is interpreted in a virtual machine on the sensor node. The virtual machine additionally offers a safe execution environment, because sensor nodes have usually no hardware protection mechanisms. Another advantage of Mate is that it simplifies programming for non-expert users.

Compared to MARWIS, Mate focus only reprogramming of sensor nodes using a high level programming interface. In MARWIS binary code of applications are transmitted and no byte code interpreted in a virtual machine on the sensor node. Thus, the reprogramming of a sensor node with Mate can be compared with the reconfiguration of a sensor node in MARWIS. Besides this difference, MARWIS also supports monitoring of sensor nodes.

MANNA

The MANNA architecture [97] considers three management dimensions: functional areas, management levels, and wireless sensor network functionalities. In the MANNA architecture, the execution of management services (composed of functions) is dependent on the information obtained from the wireless sensor network models. The definition of functions that compose these services is based on the three architectural planes: functional architecture, physical architecture, and information architecture. The functional architecture allows the establishment of all possible configurations for the management entities, such as manager, agent, and management information base (MIB). The physical architecture is the implementation of the functional architecture. In doing this, physical aspects such as the management protocol, the physical location of agents, agent functionalities, management service implemented, and supported interfaces for wireless sensor networks are defined. The MANNA information architecture is based on the object-oriented information model. Basically, the system is decomposed into two categories of modules, which play the role of managers and agents exchanging management information.

MANNA is a theoretical approach how to design a management architecture for wireless sensor networks, based on a distributed architecture using agents. The agents in MANNA can be compared to the mesh nodes in MARWIS.

TinyCubus

TinyCubus [73] is a flexible and adaptive cross-layer framework, which was developed for TinyOS-based wireless sensor networks. It supports capabilities for flexible reconfiguration, optimization and adaptation. TinyCubus consists of three parts: a data management framework, a cross-layer framework, and a configuration engine. The data management framework allows the dynamic selection and adaptation of both system and data management components. The cross-layer framework

Table 2.4: Overview over code dissemination protocols.

framework	dissemination scheme	reliability
Impala [70]	unicast	no
Trickle [69]	broadcast	no
Deluge [54]	broadcast	yes

supports data sharing and other forms of interaction between components in order to achieve cross-layer optimizations. The configuration engine allows code to be distributed reliably and efficiently by taking into account the topology of sensors and their assigned functionality. This part of TinyCubus has been described in Section 2.5.4. TinyOS is primarily used as a hardware abstraction layer. For TinyOS, TinyCubus is the only application running in the system. All other applications register their requirements and components with TinyCubus and are executed by the framework.

The main difference between TinyCubus and MARWIS is that TinyCubus is a middleware between the operating system (TinyOS) and the applications. In contrast, in MARWIS the applications are running directly in the operating system. A commonality between MARWIS and TinyCubus is that both offer a combined management architecture and data dissemination protocol.

2.7.2 Code Dissemination Protocols

In this section we present selected code dissemination protocols. These protocols are independent from management framework. Table 2.4 compares the protocols in terms of dissemination scheme and support of reliability.

Impala

Impala [70] is a middleware architecture that enables application modularity, adaptivity, and repairability in wireless sensor networks. It provides functionality to distribute new applications received via the sensor node's wireless transceiver and to be applied to the running system dynamically. For this purpose, abstractions between the operating system and the application are created. New code is only transmitted on demand if there is a new version available on a neighbour node. Furthermore, if certain parameters change and an adaptation rule is satisfied, the system can switch to another protocol. However, this adaptation mechanism only supports simple adaptation rules. Although it uses crosslayer data, Impala does not have a generic, structured mechanism to share it and so, is not easily extensible.

In contrast to MARWIS the code dissemination of Impala is based on unicast transmissions. Furthermore, Impala does not support a reliability mechanism whether the code dissemination was successful or not.

Trickle

Trickle [69] is an algorithm for code distribution in wireless sensor networks. It periodically broadcasts meta-data about the software version nodes are using. A sensor node receiving the meta-data can check if it has an older or newer version of the software. Thus, the node can decide if it requires itself an update or the neighbour node has the older software version and need an update. Since, the code update is broadcasted also nodes, which have not explicitly requested an update receive the new software version. Thus, all sensor nodes in the network get the newest version of the software.

The code dissemination of Trickle relies on broadcast transmissions and in contrast to MARWIS, does not support any reliability mechanism.

Deluge

Deluge [54] disseminates the code updates in wireless sensor networks using an epidemic routing algorithm, which uses a three-way handshake based on advertisement (ADV), request (REQ) and actual code (CODE) transfer. In Deluge, the code is subdivided into fragments (called pages), which are disseminated using a NACK-based ARQ protocol. The code is transmitted, fragment by fragment, via broadcast. So called pipelining is implemented allowing a node that correctly receives a fragment from a neighbour node to directly start the dissemination of this page to the next-hop sensor node. The randomization of the transmission of the advertisement within predetermined time windows, as well as advertisement suppression, are implemented to reduce the congestion in the propagation of the code updates through the wireless sensor network.

While MARWIS code updates is based on multicast (in combination with SNOMC), the code dissemination scheme of Deluge relies on broadcast. Deluge is the only of the selected protocols which supports reliability.

2.8 Conclusions

This chapter provides the necessary background to understand the design choices made in the development of SNOMC (our proposed overlay multicast transport protocol) and MARWIS (our proposed management architecture for heterogeneous wireless sensor networks). Most importantly we motivate the need for the new developments presented in this thesis and we argue about the contribution to the scientific community. In the context of SNOMC we presented a comprehensive overview of multicast routing protocols (Section 2.4.1), multicast transport protocols (Section 2.4.2), and traditional data dissemination protocols for wireless sensor networks (Section 2.5). In the context of MARWIS, Section 2.7 describes other management frameworks and code dissemination platforms found in the literature. Not as last, we describe the hardware (Section 2.1) and software platforms (Section 2.2) used for the work in this thesis. Proper understanding of these platforms

is needed in the argumentation of our implementation and design choices.

The thesis is further organised in two parts. The first part focusses on the design, implementation, and evaluation of our protocol for our proposed multicast protocol for reliable data transport (SNOMC). These are the corresponding chapters for design (Chapter 3), implementation (Chapter 4), and evaluation (Chapter 5) of SNOMC . In the second part MARWIS a novel management architecture for heterogeneous wireless sensor networks is presented, in design (Chapter 6) and implementation (Chapter 7). Finally we address integration of SNOMC and MARWIS and their common performance (Chapter 8).

Part I

SNOMC: A Overlay Multicast Transport Protocol for Wireless Sensor Networks

Chapter 3

Protocol Design and Architecture

This chapter presents the design and architecture of the SNOMC (Sensor Node Overlay Multicast) protocol [126, 125]. SNOMC is an overlay multicast transport protocol for wireless sensor networks that supports reliable and time- and energy-efficient transport of bulky data.

Section 3.2 describes the problem we address with SNOMC and the general design decisions for SNOMC. Section 3.3 presents the detailed description of the protocol, including the several protocol phases, such as joining and transmission phase, and the reliability mechanisms. Finally, Section 3.4 concludes the chapter.

3.1 Introduction

Wireless sensor nodes running different applications for the support of various tasks such as event detection, localization, tracking, and monitoring. Independently of the performed task an application should be configured and continuously updated for the lifetime of the sensor nodes. These processes, i.e., configuration and updating, should be done over the air [127] and their traffic pattern differs considerably from the predominant traffic pattern in wireless sensor networks. Typically traffic in wireless sensor networks is from many-to-one communication, e.g., many sensor nodes transmitting their data to a single sink. On the contrary, during a code update one-to-many communication takes place, e.g., the code update needs to reach multiple sensor nodes. Moreover, the traffic nature of the updates is bulky. Since node updates are rather crucial, transmissions should be reliable.

Distributing data from one sender node to many receivers can be done in different ways (cf. [128]). The simplest one is flooding where many independent unicast connections are established from the source to all selected receivers. Flooding, however, is inherently very inefficient, energy-consuming and unreliable. Another way is to rely on multiple unicast connections between the source and any of the desired receivers. In the context of wireless sensor networks, however, redundancy translates to higher probability of collisions, which can cause long transmission times. This leads to an inefficient and unreliable code distribution. A more efficient distribution scheme, which better fits the requirements set by configuration

and code updating, is multicast. Multicast is able to propagate data from a single sender to many receivers by affecting smaller number of sensor nodes in the network. It is easily extendable with any kind of reliability mechanism and it has better support for bulky traffic patterns, which typical occur in code update scenarios. Thus, multicast communication fits very well for any code update task in wireless sensor networks. Currently, to our knowledge, there is no single multicast protocol able to meet the combined set of requirements for reliability and efficiency (in both time and energy consumption) for bulky traffic patterns.

The question arises how can we design such a multicast protocol in a wireless sensor network. How can we support end-to-end reliability (necessary for code updates) while still keeping energy consumption and delays low? In the design process several choices need to be made including how is bulky traffic best propagated and what is an appropriate underlying MAC protocol. Further, we would like the multicast communication to be IP-based in order to access the wireless sensor network via the Internet [12].

3.2 Protocol Design on Application Layer

We wish to design a protocol that supports multicast in wireless sensor networks in an efficient and energy-conserving way. The protocol should operate with bulky traffic, which is characterized by data arrivals in bursts of, e.g., 1000 bytes (15 packets with a payload of 70 bytes). As mentioned before, a typical example of bulky traffic is a code update when the update originates from one sender and is intended to reach many destinations.

There are several approaches towards the design of multicast in wireless sensor networks. On the one hand, we can distinguish between overlay multicast [52] and IP Multicast, which differ in the protocol layer of the implementation. IP Multicast is implemented on the network layer. Its distribution tree is built between routers in the Internet. Overlay multicast is implemented on the application layer, and therefore, its distribution tree is built between the participating nodes. Furthermore, we can choose between a sender-driven and a receiver-driven formation of the multicast group. In the former, the sender decides on the receiving nodes (the participants in the multicast group) while in the latter the receiving nodes decide themselves whether they want to be part of the multicast group and receive data or not. In the sender-driven approach the multicast group only exists until the data is transmitted to the receivers (which is a typical scenario for code updates). Thus, it is not foreseen that later other receiver nodes can join the multicast group. In case of the receiver-driven approach receivers nodes can join later to a multicast group.

In case of overlay multicast, different transport protocols (UDP or TCP) can be used for the overlay connections. Due to its stateless character UDP does not support any reliability, but it benefits from low complexity. TCP can support end-to-end reliability but on the cost of certain communication overhead. Acknowledgments can be positive or negative and inform on the reception status of messages.

A possible multicast scenario in a wireless sensor network with the different node roles is shown in Figure 3.1. In such a scenario the sensor nodes have different roles. There is one sender node, which sends the data and is the root of the distribution tree. There are forwarding nodes, which just forward the data to the next-hop neighbor. Furthermore, there are branching nodes, which duplicate the data packet and forward it to two or more next-hop neighbor nodes. And finally, there are the receiver nodes.

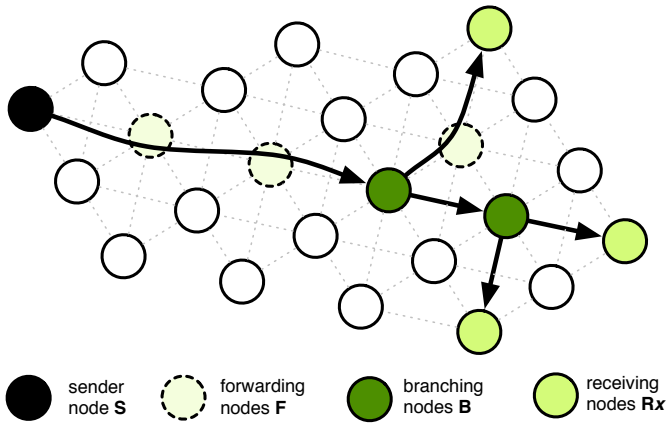


Figure 3.1: SNOMC: possible scenario.

Caching is a convenient way to have content closer to the requesting side and hence decreases delays. There are three possibilities on where to cache data: on sender nodes, on branching nodes or on all intermediate nodes (forwarding and branching). Furthermore, nodes that cache data fragments can also detect gaps in the fragment sequence and pro-actively request the missed fragments.

In order to meet our design requirements from Section 3.2 we chose an overlay multicast approach, which is able to operate in both sender-driven mode and receiver-driven mode. The main argument for an overlay approach is that it is easier to implement at the application layer than at the network layer. We are using UDP as transport protocol and to ensure reliability we are using a simple NACK-based mechanism on application layer with all three caching modes, pro-actively or passively. Our protocol - Sensor Node Overlay Multicast (SNOMC) - is described in the following section.

3.3 Protocol Description

This section describes the technical details of how SNOMC operates, including joining a multicast group and transmitting data to the group. Further, the NACK-based reliability mechanism and each of the three caching variants are described.

3.3.1 Joining Phase

To distribute data from one sender node to many receiver nodes we need to build a distribution tree. A distribution tree can be built only after the receiver nodes have joined a multicast group. As mentioned above, this can happen in two different ways, namely, sender-driven or receiver-driven. We begin with description of the **sender-driven** approach.

Sender-Driven Join Approach

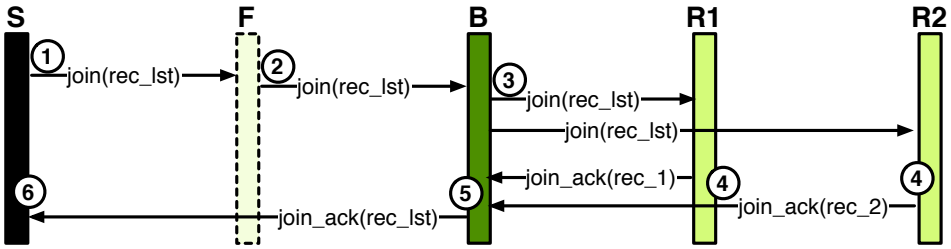


Figure 3.2: SNOMC: Joining phase, sender driven mode.

In a sender-driven approach the sender decides, which nodes should be in the multicast group as receivers. The result of the joining phase is a distribution tree, including the sender node **(S)**, forwarding nodes **(F)**, branching nodes **(B)**, and receivers **(R*x*)**. The joining phase for the sender-driven mode is shown in Fig. 3.2 and operates as follows.

(1) First, the sender **S** creates a `join` message (cf. Figure 3.3(a)), which contains amongst others the list of receivers (`rec_lst`), the multicast group id (`mc_id`) created by the sender node, and the sender address (`sender_id`). Next, the sender node transmits this `join` message to this next-hop neighbour, from which all receivers can be reached, according to the routing table.

(2) The next-hop node, upon reception of the `join` message, checks via which next-hop nodes all receivers in the list can be reached, according to its routing table. If all receiver can be reached via one next-hop neighbour, it simply forwards the full receiver list in a `join` message to this particular neighbour and acquires a forwarding role **(F)**.

(3) If the receiver nodes can be reached only via different next hop neighbours, the node acts as a branching node **(B)**. In such case the node splits the initial list accordingly and transmits the new partial lists to the respective next-hop neighbours (also according to the routing tables).

(4) When a `join` message reaches a receiver **(R*x*)** the latter confirms its role as a receiver by transmitting a `join_ack` message with its address in the `rec_lst` field back to the last branching node (cf. Figure 3.3(b)).

(5) The branching node waits for the `join_ack` messages of all subordinate receivers, combines these messages into one (by combining the `rec_lst` field from the several received `join_ack` messages) and transmits it back to the sender node (or to the next branching node up on the path towards the sender node). If after a certain time the branching node has not received all `join_ack` messages from all receivers (in case a `join_ack` message gets lost), the branching node will resend the `join` message to the according receivers.

(6) After the `join_ack` message reaches the sender node, it knows that the join procedure was successful and the data transmission phase can start.

The used caching strategy (cf. Section 3.3.2) affects the joining procedure. The intermediate nodes have to be informed about the selected caching strategy so that each nodes can decide if they have to cache the data fragments or not. This happens through the `caching_strat` field in the `join` message (cf. Figure 3.3(a)). In case of caching on every intermediate node, each forwarding node and branching node caches additionally the data fragments **(2/3)**. Moreover, each node stores its address in the `last_node_id` field. The next-hop node knows which node it has to request for lost fragments. In general, each node knows its predecessor and its successor node. Between them the overlay connections are established. In case of caching on branching nodes, only these nodes cache the data fragments and write their addresses into the `last_node_id` field **(3)**. In case of caching only on the sender node, the branching nodes do not cache the data fragments, but still writes its own address into the `last_node_id` field **(3)**, because the receivers have to know where it has to transmit the `join_ack` message. In the last two cases, each branching node knows its predecessor and its successor branching node. Between them the overlay connections are established. The caching strategy has to be defined in the joining phase, because it defines between which nodes the overlay links are established. If the caching strategy has to be changed the joining phase has to be repeated.

The structure of the `join` message is shown in Figure 3.3(a). The field `type` contains the message type, in this case *join*. The `rec_lst` field contains a list of all addressed receivers. The address of a receiver is 2 bytes long so that approximately a number of maximum 30 receivers can be in the `rec_lst` field. The `mc_id` field contains the multicast group id generated by the sender node and is used to identify the connection later at the data transmission phase. The field `caching_strat` contains the selected caching strategy (cf. Section 3.3.2); possible values are *no_cache*, *cache_branching*, *cache_branching_proactive*, *cache_intermediate*, and *cache_intermediate_proactive*. The field `sender_id` contains the address of the sender node and the field `last_node_id` the address of the last forwarding node or branching node according to the selected caching strategy. The latter information is necessary so that the nodes know their predecessor and successor to establish later the overlay connections for the data transmission phase and the reliability mechanism.

(a) join message

(b) join_ack message

Figure 3.3: SNOMC: messages for sender-driven joining.

The structure of the `join_ack` message is shown in Figure 3.3(b). It also contains the field `type` with the value *join_ack*. The second field `rec_lst` contains the list of the receivers. When the receiver creates a `join_ack` message it only contains its own address. Later, when these messages are combined at the branching nodes it contains list of all receivers subordinated to this branching node. Finally, when the message arrives at the sender node, the list contains all addressed receivers.

Receiver-Driven Join Approach

In the **receiver-driven** approach the receivers themselves decide whether they want to be in a multicast group and receive data from the sender. In this case they have to know which node is the sender node. In this approach it is also possible that receivers join the multicast group later. The joining phase is shown in Figure 3.4.

(1) Each receiver **R*x*** transmits a `join` message (cf. Figure 3.5(a)) to the next-hop neighbor in the direction towards the sender node according to the routing table. The message contains the field `rec_lst`, which initially carries only the receiver's address.

(2) A neighbor node takes the incoming `join` message and decides on its role (e.g., forwarding node or branching node). If there is just one subordinate receiver it acts as a forwarding node **F** and writes its address in the `last_node_id` field; if there are more than one receiver, it acts as a branching node **B**. In the latter case it adds its address to the `last_branch_node_id` field in the `join` message to inform the sender node (or the follow-up branching node) about its own role as branching node. This information is later used for the overlay connections. Afterwards, it transmits the `join` message further to the sender node.

(3) The sender node collects all `join` messages and creates a `join_ack` message with the receiver list, its own address as `sender_id` and decides the multicast group id for the `mc_id` field. Then it transmits the message towards the receivers, either to the next forwarding node or the next branching node, according to the selected caching strategy.

(4) The branching node splits the `join_ack` message and sends out adapted copies to the corresponding receivers

(5) When the `join_ack` message arrives at the receivers they know that their joining to the multicast group was successful.

The distribution tree is built implicitly through the path of the `join` messages and the data transmission phase can start.

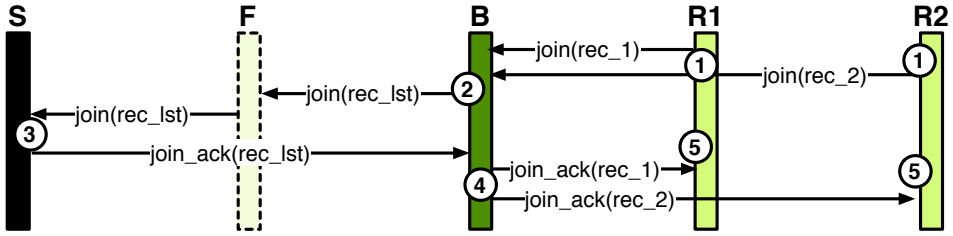
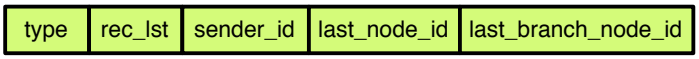


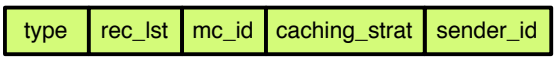
Figure 3.4: SNOMC: Joining phase, receiver driven mode.

The used caching strategy (cf. Section 3.3.2) affects the receiver-driven joining phase as well. Again, in this approach the intermediate nodes have to be informed about the selected caching strategy. The sender node decides about the used caching strategy before it transmits the `join_ack` messages towards the receivers. It writes the selected strategy in the `caching_strat` field in the `join_ack` message (cf. Figure 3.3(b)). If caching on each intermediate node is selected it forwards this message to the next forwarding node. In case of caching on branching nodes or in case of no caching, it forwards the `join_ack` message to the next branching node.

The structure of the `join` message for the receiver-driven approach is shown in Figure 3.5(a). The field `type` contains the message type, in this case *join*. The `rec_lst` field contains a list of all addressed receivers. The field `sender_id` contains the address of the sender node, the field `last_node_id` the address of the last forwarding node and the field `last_brach_node_id` the address of the last branching node. This information is necessary for the nodes to know their predecessor and successor, which helps to establish later the overlay connections for the data transmission phase and the reliability mechanism according to the selected caching strategy.



(a) join message



(b) join_ack message

Figure 3.5: SNOMC: messages for receiver-driven joining.

The structure of the `join_ack` message is shown in Figure 3.3(b). It contains also the field `rec_lst` with the list of the joined receivers. The `mc_id` field contains the multicast group id generated by the sender node and used to identify the connection later at the data transmission phase. The field `caching_strat` contains the selected caching strategy (cf. Section 3.3.2) with the possible values *no_cache*, *cache_branching*, *cache_branching_proactive*, *cache_intermediate*, and *cache_intermediate_proactive*. Finally, it contains the field `sender_id` with the address of the sender.

3.3.2 Data Transmission Phase and Caching

Propagating data from sender to receivers is done using overlay connections established as result of the join phase. It depends on the chosen caching strategy between which nodes the overlay connections are established. If the data are cached on every intermediate node (sender, forwarding, and branching nodes) the overlay connections are established between them. Thus, an overlay connection corresponds to each hop in the distribution tree. If the data are cached on a sender or branching nodes, the overlay connections are established between sender node, branching nodes, and receivers.

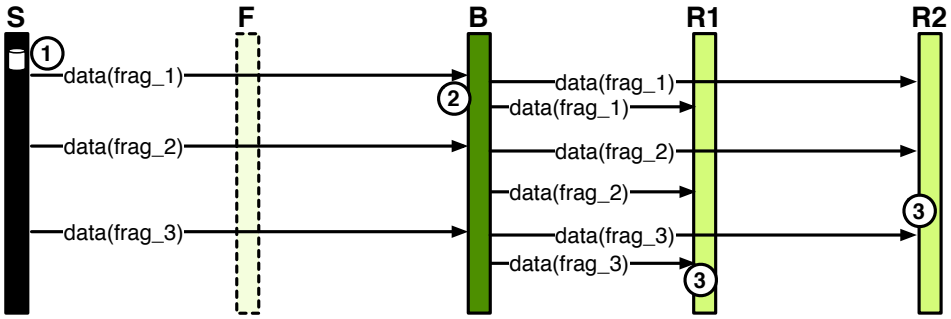


Figure 3.6: SNOMC: Transmission phase, no caching.

The data transmission in case of no caching is depicted in Figure 3.6 and operates as follows.

(1) The sender node fragments the data and caches them. Afterwards it transmits them in `data` messages (cf. Fig. 3.10) to the next branching node using the overlay connection.

(2) The branching node duplicates the `data` message and transmits the duplicates to the receivers or to the next branching nodes respectively. It does not cache the fragments. Using broadcast transmissions on branching nodes is described below and depicted in Figure 3.9.

(3) Eventually, a receiver collects all fragments.

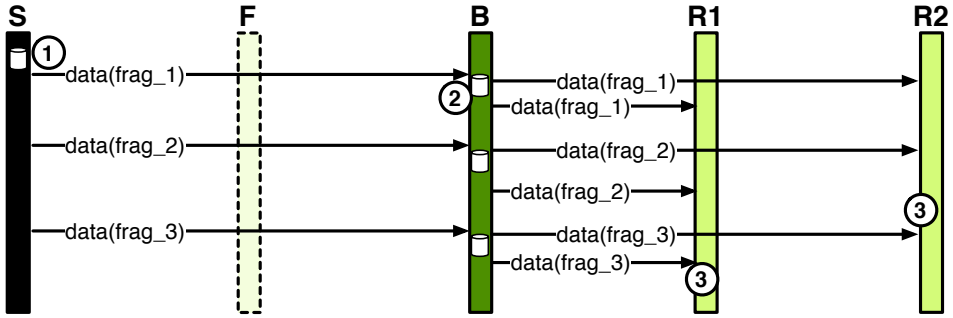


Figure 3.7: SNOMC: Transmission phase, caching on branching nodes.

Data transmission in case of caching on the branching node works quite similar, with the difference that the branching nodes cache the fragments **(2)**. It is depicted in Figure 3.7.

In case of caching on each intermediate node the procedure works a little bit different. The overlay connections are not only established between the sender node, the branching nodes and the receivers, but also between the forwarding nodes. The procedure is depicted in Figure 3.8 and works as follows.

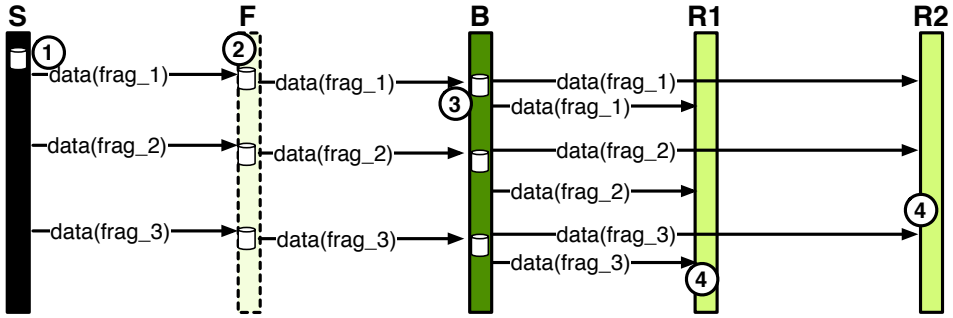


Figure 3.8: SNOMC: Transmission phase, caching on forwarding nodes.

(1) The sender node fragments the data and caches them. Afterwards it transmits them in `data` messages (cf. Fig. 3.10) to the next forwarding node using a one hop overlay connection.

(2) The forwarding node caches the fragment and forwards it to the next follow-up node.

(3) The branching node duplicates the `data` message and transmits the duplicates to the receivers or to the next branching nodes respectively. It does not cache the fragments.

(4) Eventually, a receiver collects all fragments.

If caching on each intermediate node is selected as caching strategy an optimization using broadcast transmissions is possible. On the branching nodes the

data messages have to be duplicated and transmitted to two or more successor nodes. In this case we can use a broadcast transmission instead of two or more unicast transmissions. This case is depicted in Figure 3.9 **(3)**.

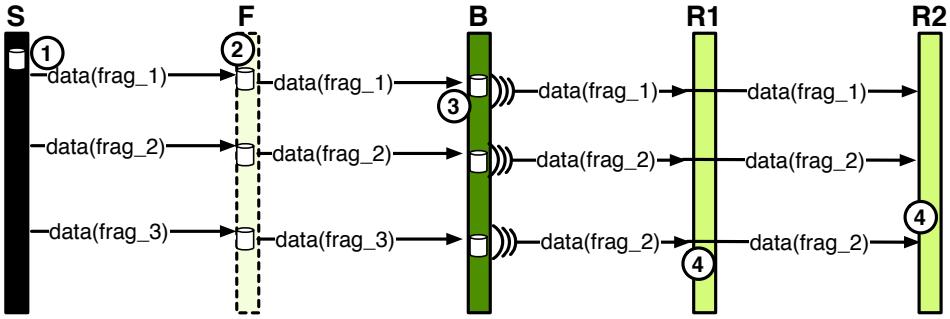


Figure 3.9: SNOMC: Transmission phase, using broadcast transmissions.

The `data` message is depicted in Figure 3.10. It contains the field `type` with the value *data*, the field `mc_id` to identify the multicast group. Then it includes two fields for the fragment numbers. The field `frag_no` contains the number of the current fragment and the field `last_frag_no` contains the number of the last fragment to indicate the end of the transmission. And finally, it contains the fragment in the `frag` field.

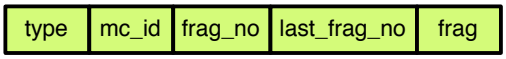


Figure 3.10: SNOMC: data message.

3.3.3 End-to-End Reliability

End-to-end reliability is ensured using a mechanism based on negative acknowledgments. In case a `data` fragment gets lost a receiver recognizes it and requests the lost `data` fragment by transmitting backwards a `nack` message. A `nack` message, arriving at a sensor node that has the missing `data` fragment cached, triggers retransmission of the fragment. There are three caching strategies: caching only on the sender node, caching on branching nodes, and caching on every intermediate node. Additionally, a node that caches data fragments can also pro-actively request lost `data` fragments. The success of a transmission is indicated using a `data_ack` message transmitted by the receivers towards the sender node. Hence, end-to-end reliability can be ensured. Below we describe each of the caching strategies.

The caching strategy only the sender node is depicted in Figure 3.11.

(1) The first transmission phase is the same like in Figure 3.6. The sender splits the data into fragments, creates the according `data` messages and transmits them to the next sensor node.

(2) A `data` fragment gets lost.

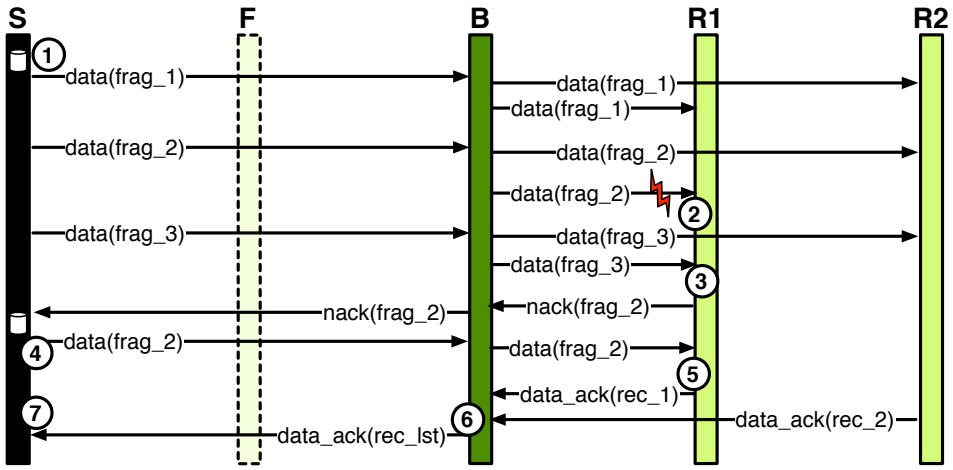


Figure 3.11: SNOMC: Reliability, caching only on sender node.

(3) If a `data` fragment gets lost the receiver detects a gap in the fragment sequence and requests the missing fragment. The receiver generates a `nack` message (cf. Figure 3.14(a)), which is transmitted towards the sender node via the according branching node.

(4) Since the sender node has the fragment in the cache, it retransmits it towards the requesting receiver.

(5) If the receiver finally gets all fragments successfully, it confirms this with a `data_ack` message (cf. Figure 3.14(b)).

(6) A branching node accumulates all incoming `data_ack` messages into a combined `data_ack` message towards the sender node. The branching node knows from the joining procedure how many `data_ack` messages have to be received.

(7) Finally, the sender node gets notified about the successful transmission of the data.

In case that the data is cached not only at the sender node, but also at the branching node the protocol operation changes slightly as shown in Figure 3.12.

(1)-(3) The first three steps are as in Figure 3.11.

(4) A missed `data` fragment will be directly retransmitted towards the receivers by the branching node, since it hast the `data` cached.

(5)-(7) Also the end of the procedure is the same as depicted in Figure 3.11.

In case the data is cached at every intermediate node (forwarding and branching nodes), which is shown in Figure 3.8, the protocol does not change very much.

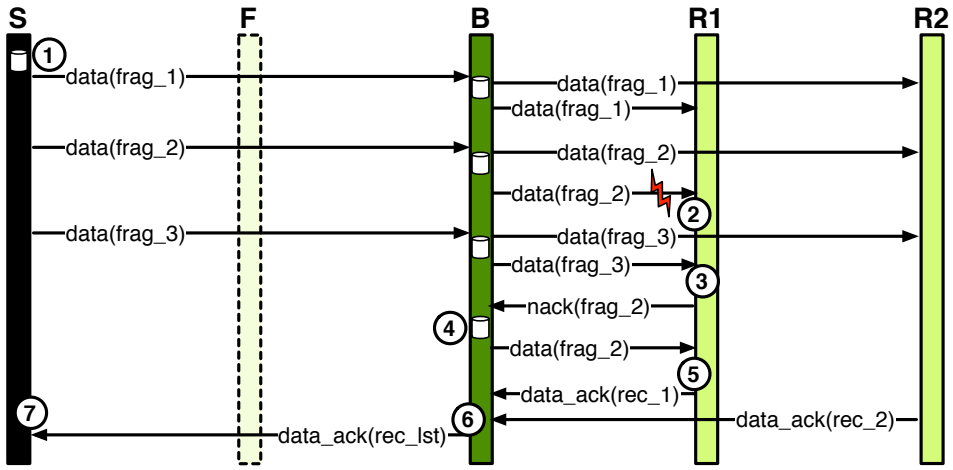


Figure 3.12: SNOMC: Reliability, caching on branching node.

The request for a lost `data` fragment is transmitted back to the last forwarding node instead of the last branching node. If the node has cached the fragment it retransmits it; otherwise it forwards the `nack` message up the distribution tree until a node is found where the fragment was cached.

The nodes that cache the `data` fragment also can pro-actively request fragments, if they detect a gap in the fragment sequence. In this case a missed fragment is detected earlier and not only at arrival at the receiver nodes. The pro-active mode is shown in Figure 3.13 with the caching strategy on each intermediate node.

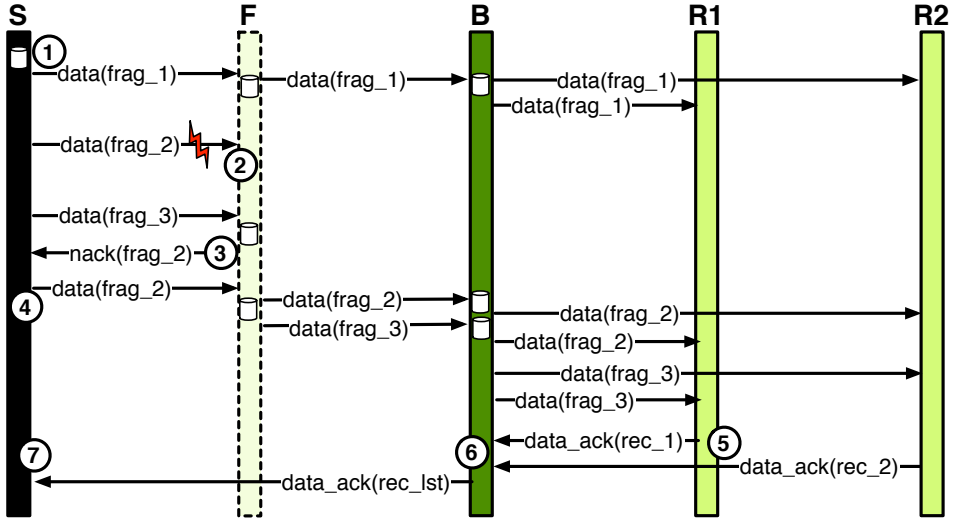


Figure 3.13: SNOMC: Reliability, caching on each intermediate node, pro-active.

(1) The first transmission phase is the same like in Figure 3.6, with the difference that the forwarding node caches the `data` fragment.

(2) A `data` fragment gets lost.

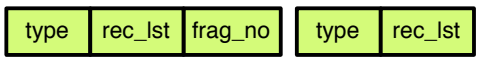
(3) The forwarding node detects the loss when the third `data` fragment arrives and requests pro-actively the missed second fragment. It does not forward fragment *3* in order to preserve the sequence of the fragments. The forwarding node then generates a `nack` message (cf. Figure 3.14(a)), which is transmitted towards the sender node.

(4) Since the previous node in the distribution tree has the fragment in the cache, it retransmits it towards the requesting forwarding node. After receiving the retransmitted fragments the forwarding node transmits fragments *2* and *3* to the next node towards the receivers.

(5)-(7) The last three steps (data acknowledgement) are the same like in the previous caching strategies.

If sensor node that requests a missed fragment starts also a retransmission timer. If the retransmission timer expires and the requested fragment does not arrived, the sensor node requests the fragment again.

To ensure end-to-end reliability we are using two types of messages, the `nack` message (shown in Figure 3.14(a) and the `data_ack` message shown in Figure 3.14(b). While the `nack` message is used for requesting missed fragments, the `data_ack` message is used to acknowledge the successful receive of all fragments at the end of the transmission (not of each fragment). The `nack` message contains the field `type` with the value *nack*, and further the field `rec_lst` with the list of the receivers. It also contains the field `frag_no` with the list of the missed fragments. The message `data_ack` contains also the field `type` with the value *data_ack* and the list of the receivers, which successfully received all fragments (field `rec_lst`).



(a) nack message (b) data_ack message

Figure 3.14: SNOMC: reliability messages.

3.4 Conclusions

This chapter introduced the Sensor Node Overlay Multicast (SNOMC) protocol, which is specifically designed for overlay multicast transport in wireless sensor networks. It supports a reliable dissemination of bulky data from one sender node to many receiver nodes.

A distribution tree is used to transmit the data efficiently from the sender node to the receiver nodes using overlay connections. We described the joining phase,

which results in a distribution tree, composed of one *sender node*, *forwarding nodes*, *branching nodes*, and the *receiver nodes*. The overlay connections are UDP flows between the sensor nodes and are described in Section 4.3.4. As an optimization we introduced a broadcast transmission at the branching nodes to avoid unnecessary unicast transmissions to two ore more following nodes.

To ensure end-to-end reliability we designed a NACK-based reliability mechanism, combined with a data acknowledgement after the successful transmission of all data fragments to the receiver nodes. To avoid costly end-to-end retransmissions we propose different caching strategies: caching on each intermediate node, caching on branching nodes, or only caching on the sender node. Further, the nodes which cache data also can pro-actively request missing fragments.

The following chapter 4 presents the SNOMC implementation in the OMNeT++ simulator and a real-world implementation in the Contiki OS. A quantitative evaluation of SNOMC is further presented in chapter 5.

Chapter 4

SNOMC Implementation

This chapter describes the implementation details of SNOMC. After a short introduction in Section 4.1, we describe the implementation details of SNOMC in the OMNeT++ simulator [9] in Section 4.2 and the implementation in the Contiki OS [38] in Section 4.3. Finally, a concise summary is given in Section 4.4.

4.1 Introduction

SNOMC was designed as a multicast transport protocol to perform efficient management tasks in MARWIS. Before integrating the two we decided to independently evaluate SNOMC. In order to make a proper performance evaluation we took the two-step approach.

First, for a proof-of concept evaluation we implemented SNOMC in the OMNeT++ simulator. Later, for real-world experiments in the Wisebed testbed and the integration in MARWIS, we implemented SNOMC in the Contiki OS.

4.2 SNOMC Implementation in OMNeT++

In this section we describe the implementation of SNOMC in the OMNeT++ simulator. SNOMC has been implemented in the INET framework provided by OMNeT++. The INET framework is suited for simulations of wired, wireless and ad-hoc networks. Beyond IP and UDP/TCP there are implementations of 802.11, Ethernet, PPP, IPv6, OSPF, RIP, and several other protocols.

In this section, first the OMNeT++ protocol stack is described. Afterwards, we describe the implementation of the protocol operations of SNOMC and the implementation of CC2420 radio transceiver. Moreover, the implementation of important SNOMC data structures and messages is discussed.

4.2.1 Protocol Stack

OMNeT++ is very appropriate for the evaluation of complex systems. Figure 4.1 shows the OMNeT++ protocol stack. Our protocol stack covers the applica-

tion, SNOMC, UDP, µIP, NullMAC and ContikiMAC as MAC protocols and the CC2420 radio.

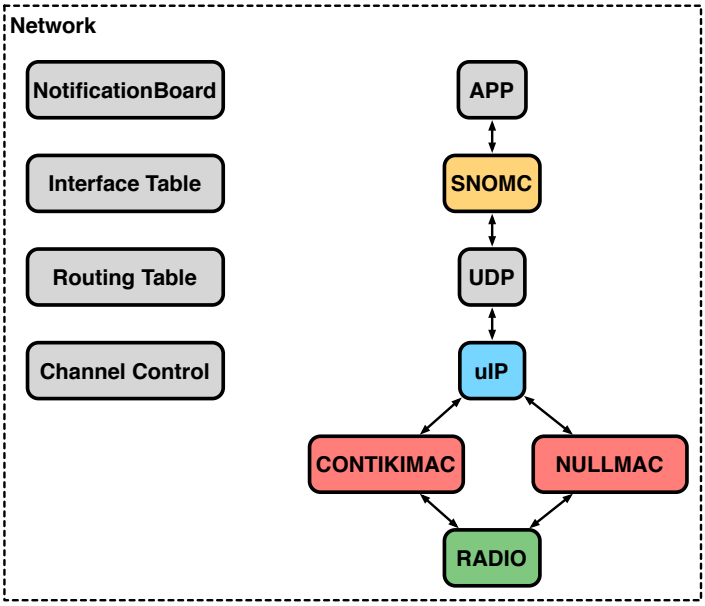


Figure 4.1: OMNeT++: protocol stack.

The application module is responsible to start the joining phase and later the data transmission. After finishing the data transmission it collects statistically data such as transmission time. The **SNOMC** protocol has been implemented as a state machine, which is described detailed in Section 4.2.2. The **UDP** module is taken unchanged from the OMNET++ INET framework. The **µIP** module is adapted from the original IP module, provided by the INET framework. All functionality to fragment packets are removed - in method fragmentAndSend(), because the maximum size of a fragment is fixed to 78 (due to of the MTU of 802.15.4 MAC-FRAMES of 128 Bytes). Since, we only want to support overlay multicast the method for routing multicast packets is removed (routeMulticastPacket()).

As MAC protocols we implemented **NullMAC** and **ContikiMAC**, which are integral part of Contiki OS. The description of the protocols can be found in Section 2.6.

The **CC2420 radio transceiver** is implemented in the class `CC2420Radio`. The state machine (shown in Figure 4.5 and described in Section 4.2.3) is implemented as a class `CC2420RadioState` and the states are defined as *enums*, such as `CC2420_IDLE`, `CC2420_TX_CAL`, or `CC2420_SLEEP`. The transitions between the states are implemented as method calls, such as `setRadioToIDLE()`, `setRadioToTX_CAL()`, or `setRadioToSLEEP()`. Some transitions are initiated by the upper MAC protocol via commands (such as `STXON` or `SRXON`). Autonomous state transitions (such as `TX CAL` → `TX Preamb Send` → `TX Send`) are handled via timers which expire according the state transition dura-

tions in the real CC2420 radio transceiver. The timers are handled as self messages via `handleSelfMessages()`. There are two buffers, one for incoming messages called RX Buffer, and one for outgoing messages called TX Buffer. To empty the RX Buffer and the TX Buffer the methods `flushRXBuffer()` and `flushTXBuffer()` are used. When a frame arrives at the radio it is delivered by the `ChannelControl` module. After finishing the receiving process the radio transmits the command `RECEPTION_COMPLETE` which initiates processing of the `handleLowerMsgEnd()` method. Within this method the frame is handled - if there is a collision (`handleCollision()`), if there is a bit error (`handleBiterror()`), or the frame received correctly (`handleCorrectFrame()`). In the latest case, the frame is written in the RX Buffer.

There are a number of global modules, such as `notificationBoard`, the `interfaceTable` as well as the `routingTable`. Using the `notificationBoard` all protocol layers can exchange information. Thus, a module is able to notify that one of their values or states have changed. When a change occurs, the module has to post the change using the method `fireChangeNotification()`. Every other module that wants to be notified about a change can subscribe to the `notificationBoard` using the method `subscribe()`. The module has to be a child from the class `INotifiable`. The subscribed module gets notified when the function `receiveChangeNotification()` is called. The `interfaceTable` contains all interfaces such as the radio or serial interface. The `routingTable` contains all routes between the sensor nodes. The `ChannelControl` module includes the radio model and calculates with the help of parameter, such as bitrate, bandwidth, carrier frequency, modulation, thermal noise, radio sensitivity, and other parameter the signal-to-noise ratio (SNR) of a signal at the receiving node. Further, the `ChannelControl` module delivers the received frame to the radio module of the node, if there is no bit error or collision.

4.2.2 Protocol Operation

The SNOMC protocol has been implemented in OMNeT++ as a state machines. Depending on whether the node acts as a sender node, a receiver node, or a forwarding/branching node the state machine differs.

Figure 4.2 shows the simplified state machine of the sender node. When it starts the joining procedure it creates the `join` message, checks the receiver list and sends the message to the according neighbour. Then it waits for incoming `join_ack` messages. When such a message arrives, it checks if all receivers have notified the joining to the multicast group. If this is not the case, the sender node stays in the `waiting for join_acks` state. If all `join_ack` messages arrived, the data transmission procedure starts. The data gets fragmented, the fragments are buffered and packed into the `data` messages and transmitted to the according neighbour. Afterwards, the sender node waits for next events. When a `nack` message arrives, it creates a `data` message with the requested fragment and transmits it to the according neighbour. When a `data_ack` message arrives, it

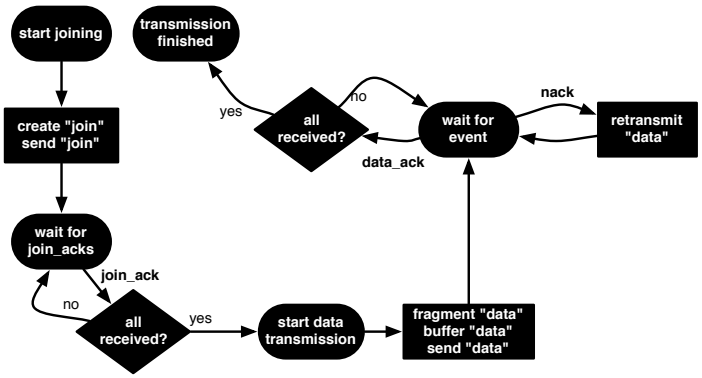


Figure 4.2: OMNeT++: SNOMC state machine, sender node.

checks, if all receivers acknowledged the successful data transmission. If this is not the case, it goes back to the `wait for event` state. If all `data_ack` messages arrived, the data transmission finished successfully.

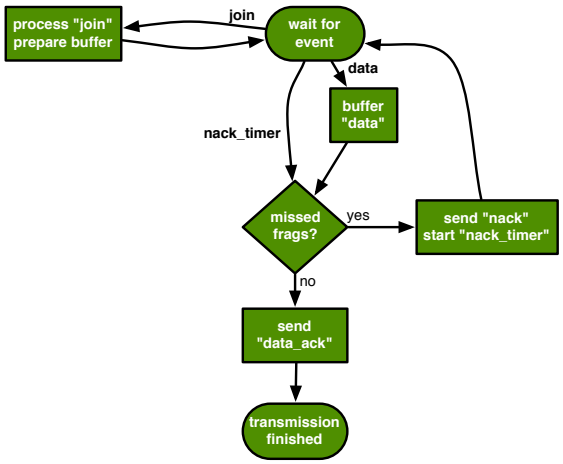


Figure 4.3: OMNeT++: SNOMC state machine, receiver nodes.

Figure 4.3 shows the simplified state machine of the receiver node. The only state is the `wait for event` state. The behaviour of the receiver depends on the incoming messages. When a `join` message arrives, the receiver processes it, defines its own role as a receiver and prepares the buffer for the incoming data messages. Afterwards, it goes back to the `wait for event` state. When a `data` message arrives the receiver node buffers it and checks if fragments are missing. If so, a `nack` message is created and transmitted towards the sender node. Afterwards, a `nack_timer` is started and the state machine goes back to the `wait for event` state. When the `nack_timer` expires, again the buffer is checked for missed fragments. If fragments are missing the timer is started again. If all fragments have successfully arrived the receiver a `data_ack` message is sent towards the sender node and the transmission is finished.

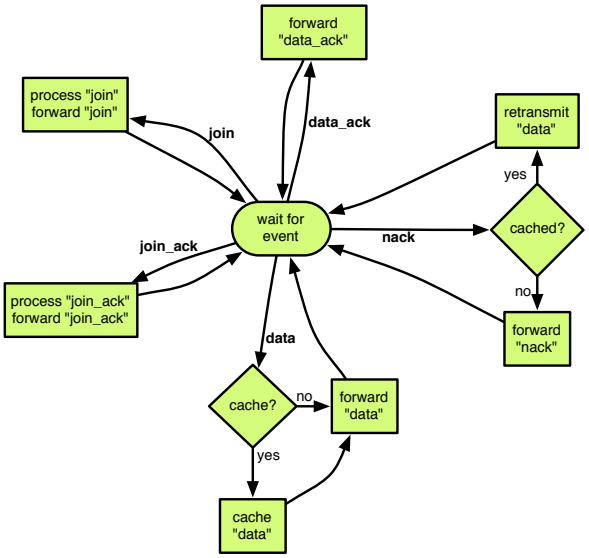


Figure 4.4: OMNeT++: SNOMC state machine, forwarding/branching nodes.

Figure 4.4 shows the simplified state machine of a forwarding/branching node. The behaviour of both node types does not differ much. A forwarding node has only to forward a message to one neighbour while a branching node has to forward the message to two or more neighbours. The state machine has only one state, the `wait for event` state. When a `join` message arrives the node decides on its role (as a forwarding or branching node) and the message is forwarded to the next neighbour. Similar is the behaviour when a `join_ack` message arrives. The message gets processed and forwarded towards the sender node. When a `data` message arrives the behaviour depends on the caching strategy. If the node does not cache fragments, the message is just forwarded. If the node caches fragments, the fragment is cached before the message is forwarded. When a `nack` message arrives, the behaviour also depends on the caching strategy. If the requested fragment is cached on the node, the fragment is taken from the buffer and transmitted as `data` message to the requesting receiver. If this is not the case, the `nack` message is transmitted towards the sender node.

4.2.3 CC2420 Radio

Figure 4.5 shows the state machine of a CC2420 radio transceiver as it was implemented in the `CC2420Radio` class. The radio transceiver has two buffers, one for outgoing messages (`RX Buffer`) and one for incoming messages (`TX Buffer`). Each of them stores 128 bytes. The radio transceiver is realized as a built-in state machine, which is used to switch between different operational states. The change of state is done either by using commands or by internal events such as SFD (Start Frame Delimiter). The CC2420 provides an `AUTO-ACK` mode. An

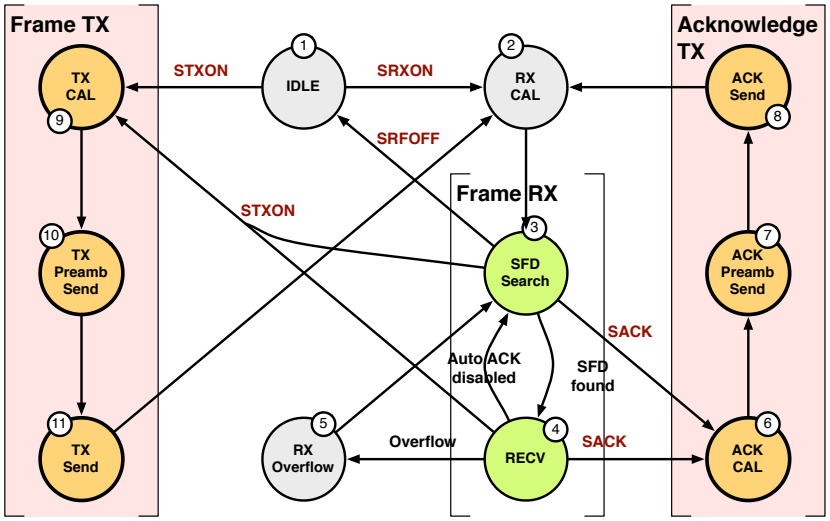


Figure 4.5: OMNeT++: CC2420 state machine.

acknowledgement frame is transmitted for all incoming frames accepted by the address recognition with the acknowledge request flag set and a valid CRC. The address recognition is part of the CC2420 transceiver.

The red transitions are commands, the black ones are internal events. The states are numbered as follows:

(1) In the IDLE state the radio is switched off. If the MAC protocol gives the STXON or SRXON command the radio transceiver switches to the corresponding state.

(2) The frequency synthesizer has to be calibrated to receive the frames. This calibration (RX CAL) takes 192 µs. Afterwards, the radio transceiver switches to the SFD Search state.

(3) In the SFD Search state the radio transceiver listens for the SFD byte (Start of Frame Delimiter). If the SFD is found the radio transceiver switches to the RECV state. If the MAC protocol transmits a SACK (Send Acknowledge) command, the radio transceiver switches to the ACK CAL state. If the radio transceiver receives a SRFOFF command, it disables the receiving/-transmission of messages and the frequency synthesizer and switches to the IDLE state.

(4) The radio transceiver receives the frame. If the RX buffer overflows, the next state is the RX Overflow. If the AUTO-ACK flag is disabled, it returns to the SFD Search state. Otherwise, the radio transceiver sets the SACK command and thus switches to the ACK CAL state.

(5) If there is an overflow (RX Overflow), the RX buffer is emptied (by MAC

command SFLUSHRX and the radio transceiver switches to the SFD Search state.

(6) The frequency synthesizer has to be calibrated to generate the signal for the auto-acknowledgment. This calibration (ACK CAL) takes 192 µs. Afterwards, the radio transceiver switches to the ACK Preamb Send state.

(7) The preamble and SFD byte are transmitted (ACK Preamb Send) and the radio transceiver switches immediately to the ACK Send state.

(8) The acknowledgment frame (MAC-ACK or AUTO-ACK) is transmitted (ACK Send) and the radio transceiver switches to the RX CAL state.

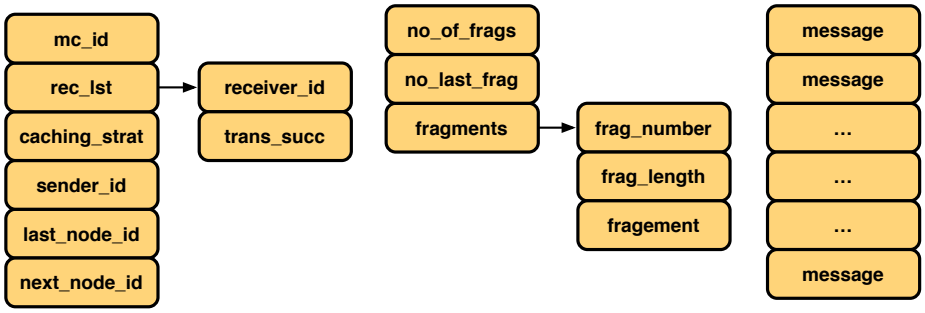
(9) The frequency synthesizer has to be calibrated to generate the signal. This calibration (TX CAL) takes a period of 192 µs. After that the radio transceiver switches to the TX Preamb Send state.

(10) The preamble and SFD byte are transmitted (TX Preamb Send) and after that the radio transceiver switches to the TX Send state.

(11) Transmitting the frame (TX Send) and switching to RX CAL.

4.2.4 Data Structures and Messages

The SNOMC implementation has four data structures to hold all necessary information: SNOMC control structure (`snomc_ctrl_struct`), SNOMC buffer (`snomc_buffer`), SNOMC incoming queue (`snomc_in_queue`), and SNOMC outgoing queue (`snomc_out_queue`). Figure 4.6 shows the four data structures.



(a) SNOMC control structure (b) SNOMC buffer (c) SNOMC in/out queue

Figure 4.6: SNOMC: data structures.

The `snomc_ctrl_struct` shown in Figure 4.6(a) contains the id of the multicast group (`mc_id`), the list of the receivers (`rec_lst`), the caching strategy (`caching_strat`), and the addresses of the sender node (`sender_id`), of the previous neighbour node (`next_node_id`), and of the next neighbour node

(next_node_id). The list of receivers is realized as a STL (Standard Template Library) list. Each entry of the list contains the address of the receiver (receiver_id) and a flag if the receiver successfully received all data (trans_succ).

The snomc_buffer shown in Figure 4.6(b) contains all buffered data fragments. It contains the total number of cached fragments (no_of_frags), the sequence number of the last fragment (no_last_frag), and a list of the fragments (fragments). The list of cached fragments is realized as a STL list. Each entry of the fragment list contains the sequence number (frag_number) and the length of the fragment (frag_length) and the data of the fragment (fragment). The SNOMC buffer has the same structure on the sender node, the receiver node and all nodes that cache the data.

The queues of the incoming packets and the outgoing packets have the same structure and are shown in Figure 4.6(c). They are also realized as a STL list and contain any kind of messages. It is organized as a FIFO queue so that the messages stay in the same order.

The SNOMC messages are also realized as C++ classes. The fields of each message are attributes of this class. Additional information and parameters can be attached to the message using the addPar function. The message handling is done using the functions handleMessage, handleLowerMessage, handleUpperMessage, and handleSelfMessage, depending on from which layer the message arrives. The following SNOMC messages have been implemented: join, join_ack, data, data_ack, and nack. The messages are described in Section 3.3 and shown in the Figures 3.3, 3.5, 3.10, and 3.14.

4.3 SNOMC Implementation in Contiki OS

In this section we present details of the SNOMC implementation in Contiki OS. In contrast to the implementation in the OMNeT++ simulator the implementation took a bigger effort, because we had to solve many challenges regarding the limitations of the sensor nodes and the operating system. We implemented the sender-driven mode and the three caching strategies, including pro-actively requesting missed fragments.

First, we describe the joining and the data transmission procedure on the implementation level. The procedures are depicted using flowcharts, which are better suited for intuitive presentation than pseudo-code listings. Afterwards we show implementation details of the caching, the packet queues, the SNOMC sender and control process, timers and variables.

4.3.1 Joining Procedure

Figure 4.7 shows the joining procedure on the sender node. It depicts the start of the joining procedure and the behaviour when a join_ack messages receives the

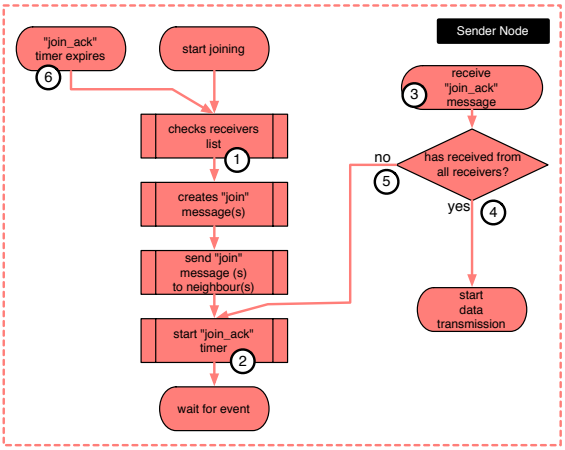


Figure 4.7: SNOMC control process: joining procedure, sender node.

sender node.

(1) The sender node starts the joining procedure. First, it checks the receivers list and the routing table and decides via which neighbour nodes the receivers can be reached. Afterwards, it creates the `join` message(s) and transmits them to the neighbours accordingly.

(2) Then, it starts the `join_ack_timer` and awaits the next event.

(3) When the sender node receives a `join_ack` message, it checks if it got it from all receivers.

(4) If it did not get the notifications from all receivers it restarts the `join_ack_timer` again and waits for the next event.

(5) If it got the notifications from all receivers the joining procedure is finished and the data transmission can start.

(6) When the `join_ack_timer` expires it checks in the receiver list which receivers have not notified the joining in the multicast group. Then, the procedure continuous with step **(1)**.

Figure 4.8 shows the joining procedure for the forwarding nodes, branching nodes, and receiver nodes. Initially, the nodes will be not know which role they will play and which caching strategy is used.

(1) The node receives the `join` message. First, it checks the receiver list in the `join` message and define its own role as follows. If all receivers can be reached via one single neighbour node, it acts as a forwarding node. If the receivers can be reached via different neighbour nodes, it acts as a branching node.

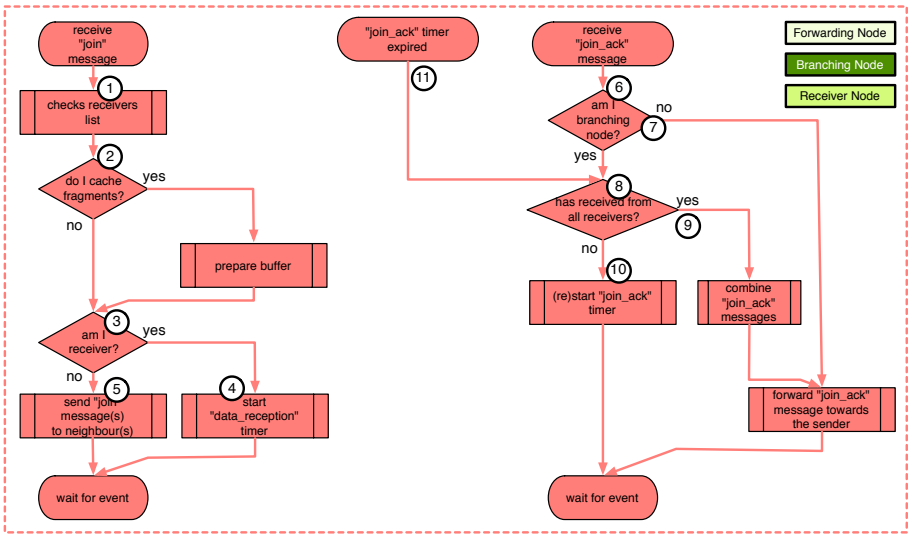


Figure 4.8: SNOMC control process: joining procedure, other nodes.

(2) Then, it checks in the `join` message which caching strategy is used. If the node has to cache the data fragments it prepares the buffer. If not, it continues without the buffer preparation.

(3) In the receiver list of the `join` message the node sees if it is a receiver node or not.

(4) If it is a receiver node, it starts the `data_reception_timer` and waits for the next events.

(5) If the node is not on the receivers list, it just forwards the `join` message to the one or more neighbour nodes (depending on whether the node is a branching node or a forwarding node). Then, it waits for the next event.

(6) When the node receives a `join_ack` message, it checks first its own role.

(7) If the node is a forwarding node it just forwards the `join_ack` message towards the sender node.

(8) If the node is a branching node, it checks if it got notifications from all receiver nodes.

(9) If it got all notifications, it combines these notifications to a new `join_ack` message and forwards it towards the sender node.

(10) If not all notifications from all receivers have arrived the node starts the `join_ack_timer` and waits for incoming events.

(11) When the `join_ack_timer` expires, it checks if all notifications arrived in the meanwhile and the procedure continuous with step **(8)**.

4.3.2 Data Transmission Procedure

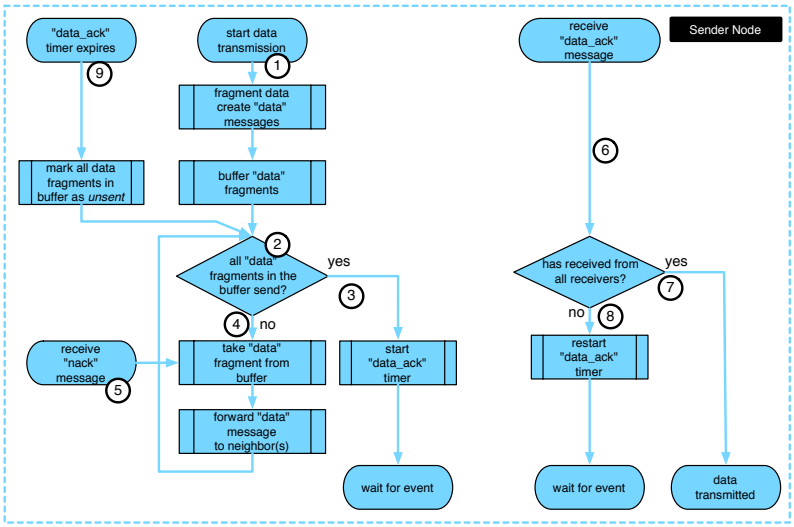


Figure 4.9: SNOMC control process: data transmission, sender node.

After the joining procedure the data transmission procedure starts. Figure 4.9 shows the procedure for the sender node. The sender node is responsible for the fragmentation of the data and for the transmission to the receiver nodes. The data transmission is finished when all receivers are notified that all data arrived successfully.

(1) As first step of the data transmission procedure the data has to be fragmented. This procedure is described in detail in Section 4.3.3. Then, the data fragments are stored in the buffer. In the beginning all data fragments in the buffer are marked as *unsent*.

(2) A standard check up is made, if all data fragments in the buffer were sent. In the first check after the initial data fragmentation clearly no data fragment has been sent so far.

(3) If all fragments were sent, the `data_ack_timer` is started and the process waits for the next events.

(4) If not all fragments are sent, the next unsent fragment is taken from the buffer and forwarded to the next neighbour(s). Afterwards, the procedure continues with step **(2)**.

(5) If a `nack` message arrives at the sender node, the requested missed fragment will be taken from the buffer and transmitted to the neighbour(s). Since the sender node created the fragments, it has all of them in the buffer for sure. It is also independent of the caching strategy, because the sender node anyway caches all fragments.

(6) If a `data_ack` message arrives the sender node this means that one or more receivers successfully got all data fragments. The sender node checks if it got notification from all receivers.

(7) If it got a notification from all receivers, the data transmission finished successfully.

(8) If it does not got a notification from all receivers, the `data_ack_timer` is restarted and the sender node waits for next events.

(9) The expiration of the `data_ack_timer` means that the data transmission was not successful to all receivers or a subset of them. In this case all fragments in the buffer are marked as *unsent* and the data will be retransmitted to all receiver nodes. The procedure starts again with step **(2)**.

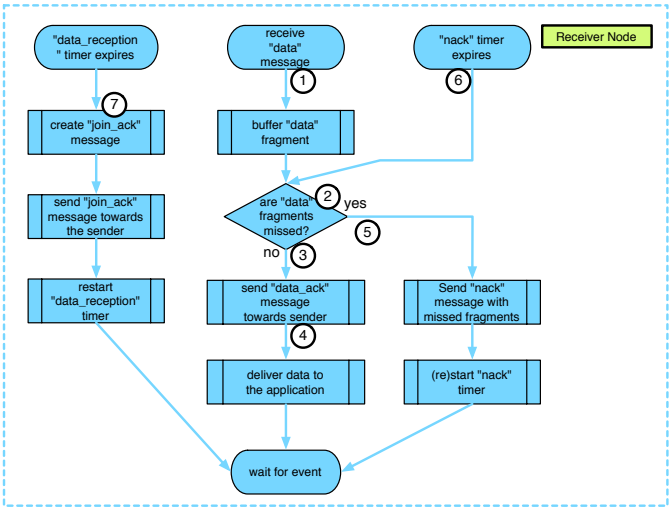


Figure 4.10: SNOMC control process: data transmission, receiver nodes.

Figure 4.10 shows the situation at the receiver nodes. The receiver node gets the data fragments and they request missed fragments. When all data fragments have arrived the receiver notifies the sender node about this.

(1) When the receiver node gets a `data` message with a data fragment it buffers it in its buffer.

(2) Then, the receiver node checks, if there are missed fragments.

(3) If this is not the case, the receiver notifies the sender node about this by transmitting a `data_ack` message.

(4) After processing case **(3)**, the receiver reassembles the data, deliver it to the application and waits for next events.

(5) If the receiver did not get all fragments, it creates a `nack` message with the missed fragments and forwards it towards the sender node. Then, it (re)starts the `nack_timer` and waits for new events.

(6) When the `nack_timer` expires the receiver node checks again which fragments are missing. The procedure continuous with step **(2)**.

(7) The `data_reception_timer` is started after the `join` message received (cf. Figure 4.8). If data only consists of one fragment and this fragment gets lost, the receiver node cannot know if there is any data transmission after the joining procedure. This timer ensures that the receiver node does not wait infinitely after the joining procedure. When the `data_reception_timer` expires a second `join_ack` message is generated and transmitted towards the sender node. Afterwards, the `data_reception_timer` is restarted.

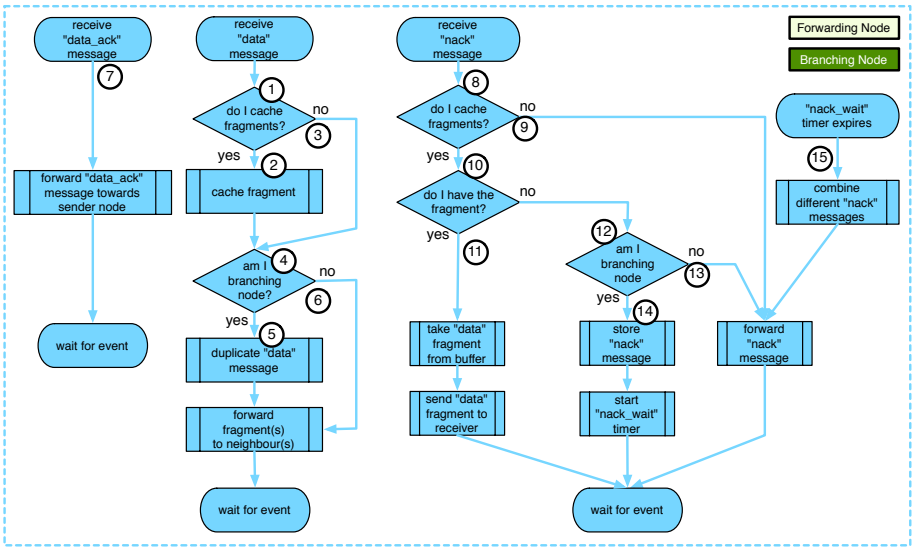


Figure 4.11: SNOMC control process: data transmission, other nodes.

Figure 4.11 shows the situation at the forwarding and branching nodes. Both nodes are responsible for forwarding of data messages, for caching, and for retransmission of missed fragments.

(1) When a node receives a `data` message it first checks, if it should cache the data, depending on the role of the node and the selected caching strategy.

(2) If yes, the node stores the data fragment in the buffer, if it is not already cached (in case the `data` message was a retransmission).

(3) If it should not cache the data (according to the caching strategy), it just continues with the procedure.

(4) Next, the node checks if it is a branching node.

(5) If it is a branching node it duplicates the `data` message and forwards it to two or more neighbour nodes. If the broadcast optimization is enabled the branching node broadcasts the `data` message.

(6) If the node is a forwarding node, it forwards the `data` message with the duplication and waits for further events.

(7) When the node receives a `data_ack` message, it forwards the message towards the sender node.

(8) When a `nack` message arrives, the node first checks if it generally should cached fragments, depending on the role of the node and the selected caching strategy.

(9) If the node should not cache fragments, it only forwards the `nack` message towards the sender node.

(10) If the node does caching it checks the requested fragment is in the buffer.

(11) If the fragment is in the buffer, then it copies the fragment from the buffer and sends it to the requesting receivers.

(12) If the node does not have the requested fragment in the buffer, the nodes further behaviour depends on whether the node is a forwarding node or a branching node.

(13) If it is a forwarding node it only forwards the `nack` message towards the sender node.

(14) If it is a branching node it stores the `nack` message and waits for certain time (`nack_wait_timer`) for further `nack` messages to combine them.

(15) When the `nack_wait_timer` expires it checks if in the meanwhile other `nack` messages arrived. It combines them and forwards the combined `nack` message towards the sender node.

When the node pro-actively requests missed fragments it checks if fragments are missed in its own buffer and pro-actively requests the missed fragments. The procedure is the same as shown in Figure 4.10, numbers **(2)** to **(5)**.

4.3.3 Fragmentation, Caching, and Buffer

The fragmentation of data of 1000 bytes into smaller fragments of 70 bytes each is accomplished by the function `buildFragment(uint8_t size)`. This function builds a `data` message (fragment) with a given size. Afterwards, this function stores the fragment in the buffer.

To implement the cache discussed in Section 3.3.2, a simple buffer was implemented as cache. The caching strategy is defined in the `snomc_ctrl_struct`

shown in Figure 4.6(a). All types of nodes (sender node, forwarding nodes, branching nodes, receiver nodes) have the same buffer implementation. The structure of the buffer is called `buffer_t` and includes, besides the storage of the fragments, also information about the number of fragments (`no_of_frags`) and the number of the last fragment (`no_last_frag`). The size of the memory is dynamically allocated using the Managed Memory Allocator (`mmem`) provided from Contiki OS. The structure of the buffer is depicted in Figure 4.12.

Figure 4.12: Buffer structure.

To cache a fragment, an intermediate node simply copies the *frag* field of the `data` message to the right position in the cache buffer. This position in the cache buffer is characterized by a start byte, which is calculated from the size of the fragment and the *frag_no*.

4.3.4 SNOMC Control/Sender Process and Packet Queues

As discussed in Section 2.6, the µIP TCP/IP stack of Contiki OS uses a single buffer for incoming and outgoing messages. Therefore, µIP does not have a queue for incoming or outgoing packets and only one packet is stored. Thus, packet loss also happens because the sensor node can not store an incoming packet while the µIP buffer is occupied, since the processing of the current packet is not finished and µIP does not overwrite the global packet buffer. That is problematic, since due to the preconfigured backoff time of the MAC protocol, a node waits before it actually transmits a packet. During this time period, the node is not able to receive new packets. To overcome this issue we implemented the SNOMC protocol as four Contiki processes: SNOMC control process (described above) with the incoming packet queue process and the SNOMC sending process with the outgoing packet queue process. Both packet queues act as proxies between the SNOMC control/sender processes and µIP. Thus, the SNOMC control/sender processes do not talk directly to µIP, instead they use the packet queues for receiving new packets and transmitting packets.

The SNOMC sending process is shown in Figure 4.13. The SNOMC sender process does not transmit a message directly via the µIP buffer. Instead, it adds the

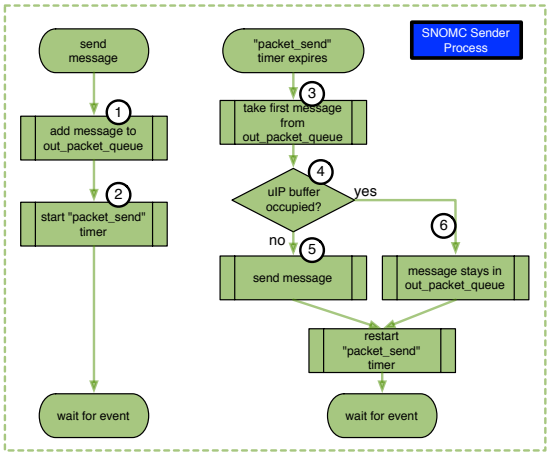


Figure 4.13: SNOMC sender process: sending messages

message to the outgoing packet queue (`out_packet_queue`). The `out_packet_queue` is polled frequently by the SNOMC sender process (randomly between 20ms and 84ms) and the first message in the queue is send.

(1) When SNOMC wants to send a message, it does not simply send it via the µIP buffer. It stores the message in the outgoing packet queue (`out_packet_queue`). The queue is in fact a FIFO queue, the first packet stays on the first position. Thus, the order of the packets is ensured.

(2) After storing the packet, the `packet_send_timer` is started. Then, the process waits for next events.

(3) When the `packet_send_timer` expires, the first packet is taken from the `out_packet_queue`.

(4) Then it is checked, whether the µIP buffer is free.

(5) If the buffer is free, the message is sent. Afterwards the `packet_send_timer` is restarted.

(6) If the buffer is occupied the packet stays in the `out_packet_queue` and the `packet_send_timer` is restarted.

The outgoing packet queue is implemented using managed memory allocator provided by the Contiki OS. Thus, we have a dynamic sized queue. The SNOMC sender process uses the function `addToQueue(* msg)` to add a packet to the queue. Since, the queue is organized as a FIFO memory, the function `getFromQueue()` always returns the first (and oldest) packet of the queue.

Figure 4.14 shows the interaction between the SNOMC control process and the incoming packet queue .

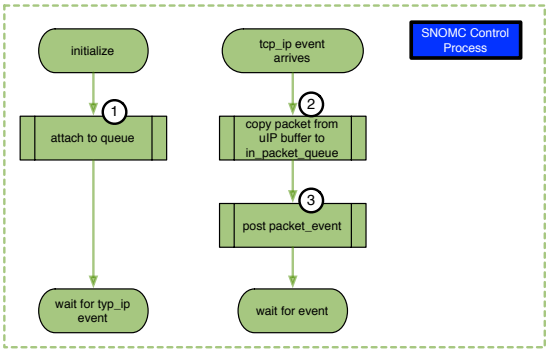


Figure 4.14: SNOMC control process: receiving messages

(1) First, the process of the communication protocol has to tell the incoming packet queue that it wants to use the queue. The process does that by executing the `attachToQueue(handle h)` function. This function stores the process ID of the process of the SNOMC receiver process in the data structure `h` of type `handle`. Next, the `attachToQueue` function starts the incoming packet queue process, which is listening for `tcp_ip` events posted by µIP.

(2) If new packets arrive, µIP posts the `tcp_ip` event, signalling that a new packet was copied into the global packet buffer. The packet queue now checks for a free packet slot.

(3) If so, the new packet in the global packet buffer is copied to this slot, otherwise, the packet is dropped. The packet queue now posts the `packet_queue` event to the process with the ID stored in the data structure `handle`. The receiver process of the SNOMC protocol has to listen for the event `packet_event`. If such an event arrives, the process uses the `getPacket` function to access the new packet. This function returns a pointer to the data of the new packet.

4.4 Conclusions

This chapter focussed on the details of the SNOMC implementation in the OMNeT++ simulator and in the Contiki OS. Based on the SNOMC design described in Chapter 3 we implemented SNOMC for both platforms. The protocol operations are implemented as state machines. The behaviour depends on the state and the incoming message. Due to the limitations of the µIP buffer incoming and outgoing messages are queued in two separate queues. An important property of SNOMC is caching, which is implemented using a buffer structure. In OMNeT++ we additionally implemented the underlying MAC protocols NullMAC and ContikiMAC and the CC2420 radio transceiver.

With these two implementations of SNOMC we had the possibility to make a proper performance evaluation of SNOMC against other popular data dissemi-

nation protocols as well as the possibility to integrate SNOMC in the MARWIS architecture. This integration allows us to perform management tasks such as code update, reconfiguration and monitoring in an efficient way. This has been evaluated in Chapter 8.

The following Chapter 5 presents the quantitative evaluation of SNOMC. To make a proof-of-concept evaluation we first use the implementation in the OMNeT++ simulator. Later, to determine its performance in a real-world environment we use the Contiki OS implementation in the Wisebed testbed.

Chapter 5

SNOMC Evaluation

In this section we evaluate the SNOMC (Sensor Node Overlay Multicast) protocol and compare its performance against a number of other data dissemination protocols for wireless sensor networks, such as Flooding, MPR (Multipoint Relay) [90], PSFQ (Pump Slowly, Fetch Quickly) [129], TinyCubus [73], and Directed Diffusion [59, 60]. For a detailed description of the protocols we refer to Section 2.6. Additionally, we compared it to two unicast protocols, namely UDP and TCP.

After Introduction in Section 5.1 we present the results of the evaluation using the OMNeT++ simulator in Section 5.2. The evaluation of SNOMC in the Wisebed testbed is presented in Section 5.3. Section 5.4 compares the simulated results and the results of the real-worlds experiments. Section 5.5 concludes this chapter.

5.1 Introduction

The performance of SNOMC in terms of transmission time, number of totally transmitted packets and energy consumption is compared to other data dissemination protocols often used in wireless sensor networks. We chose broadcast-based protocols such as Flooding, MPR (Multipoint Relay), and TinyCubus. Furthermore, we consider typical data dissemination protocols especially designed for wireless sensor networks, such as PSFQ (Pump Slowly, Fetch Quickly), and Directed Diffusion. Since we are using Contiki OS, which supports a µIP protocol stack, we also compare SNOMC to the unicast-based distribution using UDP and TCP.

We implemented the above protocols in the OMNeT++ simulator and made evaluations to proof the general concepts of the SNOMC protocol, such as joining procedure, data transmission, reliability mechanisms, and the different caching strategies. Later, we implemented them in Contiki OS to evaluate SNOMC under real-world conditions running experiments in the Wisebed testbed.

To ensure a fair evaluation we implemented on top of each of the protocols (except for TCP) a simple NACK-based reliability mechanism including different caching strategies as used in SNOMC. Furthermore, we used SNOMC on top of different MAC protocols to show the performance of SNOMC, independently of

the underlying MAC protocol. In the OMNeT++ implementation we used BEAM [13, 122], and implementations of NullMAC and ContikiMAC, which are integrated in the Contiki OS. The latter two we used for our real-world experiments in the Wisebed testbed. We could not use BEAM for the real-world experiments, because the BEAM implementation in Contiki OS does not support broadcast transmissions.

5.2 SNOMC Evaluation of Simulation Results

This section presents the evaluation of the SNOMC protocol in the OMNeT++ simulator. First, we describe the protocol stack. Then, we introduce the different simulation scenarios. Finally, we discuss the results of our measurements.

5.2.1 Protocol Stack

To evaluate the performance of SNOMC, we compare it to a number of data dissemination protocols commonly found in wireless sensor networks in combination with different underlying MAC protocols. More specifically, these protocols are: Flooding, Multipoint Relay, TinyCubus, Directed Diffusion, UDP, and TCP as data dissemination protocols as well as BEAM [13], ContikiMAC [33] and NullMAC as MAC protocols. For the description of the protocols we refer to Section 2.6. All protocols have been implemented in the OMNeT++ simulator [9]. The protocol stack is shown in Fig. 5.1 and is based on the µIP stack from Contiki OS. To enable a fair comparison we had to ensure end-to-end reliability for all protocols and implement the same simple NACK-based reliability mechanism as used in SNOMC.

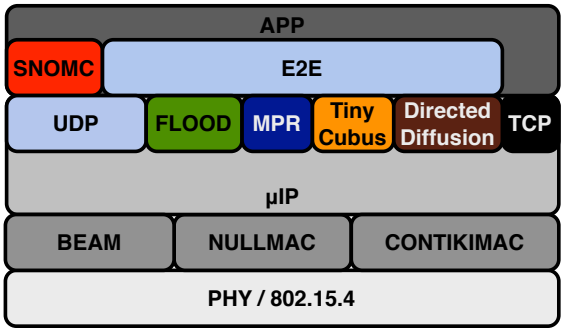


Figure 5.1: Simulation protocol stack.

The underlying MAC protocol plays an important role for the transmission of data between neighbour nodes. Hence, different MAC protocols can lead to significantly different results, irrespectively of the used transport or multicast protocol. We chose three MAC protocols with different support mechanisms for reliability and energy-efficient operation. The Burst-aware Energy-Efficient Adaptive MAC Protocol (BEAM) [13] uses an adaptive duty cycle mechanism, which reacts

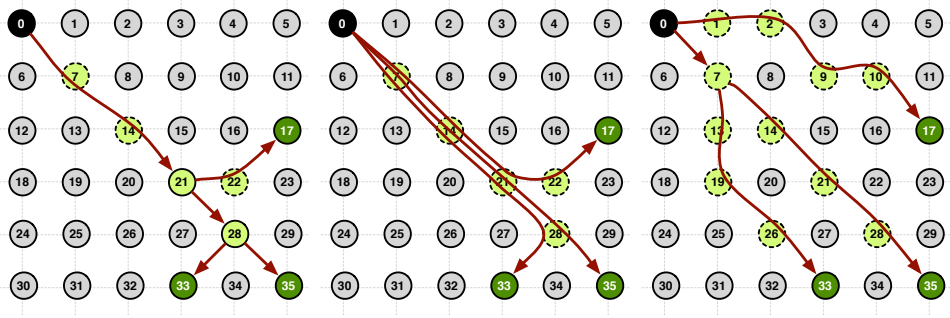
Table 5.1: MAC Protocol Parameters.

	acknowledgements	max. retransmissions	energy-saving
BEAM	positive ack	5	yes
ContikiMAC	early ack	2	yes
NullMAC	no	0	no

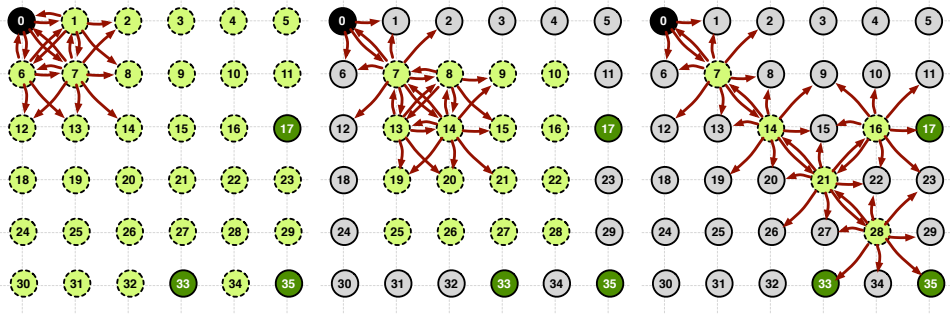
quickly to changes in both traffic loads and traffic patterns and ensures hop-to-hop reliability. ContikiMAC [33], which is part of the Contiki operating system, also supports energy-saving radio duty cycling mechanisms and reliability based on an acknowledgement mechanism. NullMAC, also part of Contiki, has no energy-saving mechanisms and does not support reliability. Table 5.1 shows an overview of the parameter of the used MAC protocols.

5.2.2 Simulation Scenarios

We arranged 36 sensor nodes in a grid of 6x6 nodes with a distance of 100 meters between each two nodes as shown in Fig. 5.2. Since we are interested in a multicast scenario, we chose a sender (node 0) with three receivers (node 17, 33, and 35).



(a) SNOMC (b) UDP and TCP (c) Directed Diffusion



(d) Flooding (e) MPR (f) TinyCubus

Figure 5.2: Simulation scenarios.

Table 5.2: Simulation Parameters.

carrierFrequency	bit-rate	sensitivity	thermalNoise	TX power	modulation
2.4 GHz	250 kbps	-94 dBm	-110 dBm	1mW	O-QPSK

Table 5.3: Energy Consumption of the CC2420 Radio Transceiver.

mode	sleeping	receiving	transmitting
consumption in mA	0.4 mA	18.8 mA	17.4 mA

Given the chosen simulation scenario, each of the compared protocols affects a different set of nodes. In case of SNOMC there are two branching nodes (21, 28) and three forwarding nodes (7, 14, 22) as shown in Fig. 5.2(a). For UDP and TCP the same nodes are affected but there are three independent unicast connections (cf. Figure 5.2(b)). As shown in Fig. 5.2(c) a different set of nodes participates in the distribution tree in Directed Diffusion, which is a result of the different Interest message routing compared to the static routing of SNOMC. In Flooding all nodes are affected (cf. in Fig. 5.2(d)). In the chosen grid scenario the Multipoint Relay protocol calculates a rather high number of multipoint relay nodes (cf. Fig. 5.2(e)). In the case of TinyCubus, the same set of nodes as in SNOMC is affected. Due to design decisions the protocol does not distinguish between receivers and intermediate forwarders (cf. Fig. 5.2(f)). Hence, all nodes in the set (7, 14, 21, 22, and 28) will rebroadcast the packets.

We created two evaluation scenarios, which differ in the size of the transmitted messages - 20 bytes and 1000 bytes. The size of 1000 bytes is typically associated with software updates on the sensor nodes; the size of 20 bytes is related to a short configuration message for the sensor nodes. For each scenario, 50 simulation runs are used for evaluation. We measured three parameters: (i) transmission times from the sender to all receivers (including the joining phase), (ii) the total number of packets it takes to ensure the successful reception of the data by all receivers, and (iii) the energy consumption of the nodes in the network. The energy consumption is measured according to the CC2420 state machine with real state switching times and energy consumption according to [120] and [83] (values for sleeping, receiving, and transmitting, see Table 5.3) and is calculated per node and per transmitted byte. The energy consumption of the CPU and writing the flash memory is much smaller than the energy consumption of the radio transceiver. Therefore, only the energy consumption of the radio transceiver is taken into account.

In order to get realistic simulation results a radio model in OMNeT++ is implemented according to the data CC2420 manual [120] and the Castalia Simulator [83]. It is used to calculate the signal to noise ratio (SNR) based on parameters shown in Table 5.2. Using the SNR and real measurements with a CC2420 radio transceiver the bit error rate (BER) is calculated. In addition, a packet error rate of

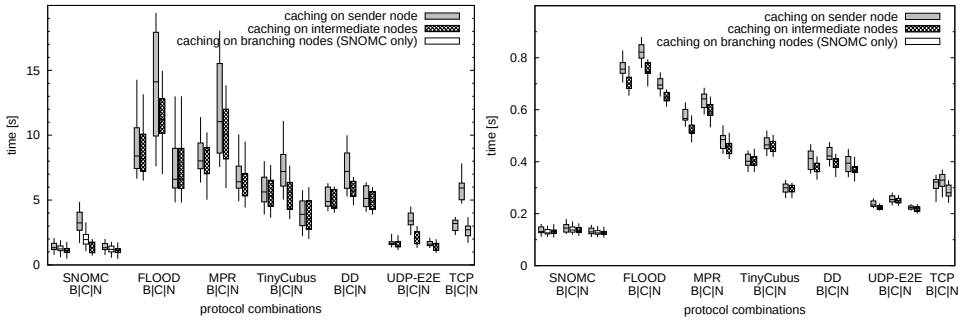
5% is assumed to represent random noise and external interferences.

5.2.3 Transmission Times

In this section, we present our findings on transmission times. In all figures on the x-axis the combinations of transport and MAC protocols are shown. In our notation **B** stands for BEAM, **C** for ContikiMAC, and **N** for NullMAC.

First, in Fig. 5.3(a), we discuss results for the time required to transmit 1000 bytes from the sender node to the three receiver nodes. SNOMC requires the shortest time to transmit 1000 bytes to the receivers. As expected, unicast data distribution based on UDP-E2E requires more time due to the redundant unicast flows that need to be established for each of the three receiver nodes. TCP performs even worse since every packet has to be acknowledged. This kind of traffic pattern caused by simultaneous data and acknowledgements increases collision probability and hence affects delay negatively. Flooding, Multipoint Relay, and TinyCubus are all broadcast protocols and are much worse in performance. On the one hand, broadcasting affects usually more nodes, which leads to a higher number of transmissions. Consequently, the probability of collisions increases and more retransmissions are necessary, pushing delay up. On the other hand, to avoid collisions higher random back-off times are necessary compared to unicast-based protocols. This leads to longer transmission times.

Further we can see that using caching in intermediate nodes performs better than caching only on sender node. It avoids long end-to-end transmissions, which cost time. Further, it reduces the number of packets in the network, which reduces the probability of collisions.



(a) 1000 bytes (b) 20 bytes

Figure 5.3: Evaluation: transmission time.

Let us now compare the performance of the MAC protocols (BEAM, ContikiMAC, NullMAC) in combination with the higher level transport protocols. In the diagrams, shown in Fig. 5.3(a), the MAC protocols are indicated with the letter underneath the transport protocol name. In our notation **B** stands for BEAM, **C** for ContikiMAC, and **N** for NullMAC. We see that BEAM has a little worse performance than NullMAC (for SNOMC, Directed Diffusion, UDP-E2E and TCP).

However, BEAM outperforms ContikiMAC, which is the result of two factors. First, BEAM is optimized for bulky traffic while ContikiMAC focuses only on constant (or slowly changing) traffic. Bulky traffic is characterised by transmissions of a large number of packets within a short time period. Second, BEAM [13] has better congestion control and duty cycle mechanisms. The latter is also the reason why caching at intermediate nodes affects the performance of both protocols differently, i.e., for BEAM the effect is much smaller than for ContikiMAC. NullMAC in combination with broadcast-based protocols (Flooding, MPR, TinyCubus) works better than BEAM in combination with these protocols. Since BEAM has energy-saving mechanisms, the radio transceiver can be in sleep mode. If so, longer time is needed to transmit a packet from sender to receiver. On the contrary, the radio transceiver in NullMAC is always on and therefore the sender can immediately transmit the packet.

In case of 20 bytes of data a single packet has to be transmitted. Transmission times are shown in Fig. 5.3(b). We see that SNOMC achieves the best performance. Ideally, TinyCubus requires a smaller number of transmissions compared to SNOMC (due to using broadcast transmissions). SNOMC, however, has the additional benefit of smaller random-back off times (see implementation of the packet queues in Section 4.3.4). UDP and TCP need more transmissions and longer transmission times due to the three independent flows. Due to just one packet has to be transmitted in scenario 2, also TCP requires only one acknowledgement for the transmitted packet, explaining the much smaller differences between TCP and UDP compared to the scenario with 1000 bytes.

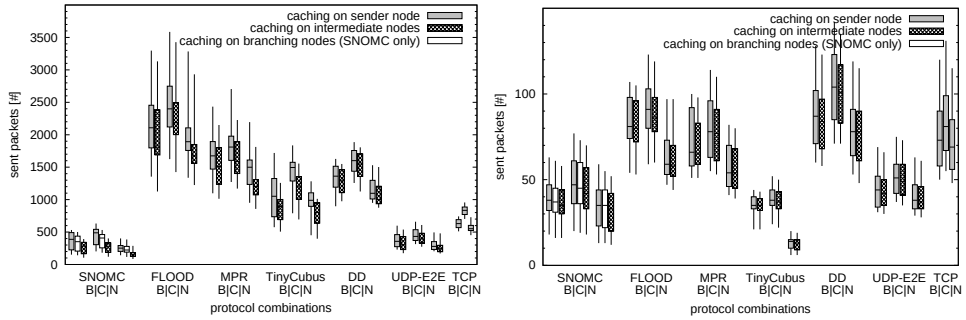
Further, in SNOMC, TinyCubus, Directed Diffusion and UDP collisions among data and acknowledgements generally do not occur and hence no retransmissions are required. Therefore, the corresponding boxplots in Fig. 5.3(b) are quite compact and do not have big outliers. Finally, the differences between Flooding, Multipoint Relay and Directed Diffusion are similar as for the 1000 bytes scenario.

5.2.4 Number of Transmissions

Fig. 5.4(a) shows the number of total transmissions needed for the successful transfer of 1000 bytes. As we can see, SNOMC requires the fewest number of packets, followed by UDP, TCP and Directed Diffusion. The results of broadcast-based protocols (Flooding, Multipoint Relay, and TinyCubus) are considerably worse and are compliant with our observations on transmission times. Flooding requires most transmissions (inherent to its communication style), followed by Multipoint Relay (result of the disadvantageous set of multipoint relays) and, with the best performance of the three, TinyCubus.

Looking at the MAC protocols, BEAM requires more packets to ensure hop-to-hop reliability than NullMAC, irrespectively of the transport protocol. This is due to the fact that the receiver can be in sleeping mode and multiple attempts may be required before the packet is transmitted successfully. ContikiMAC always needs more packet retransmissions on link layer due to a worse duty cycle mechanism

compared to the adaptive duty cycle mechanisms of BEAM. Thus, a higher number of necessary end-to-end retransmissions on the transport layer is required.



(a) 1000 bytes (b) 20 bytes

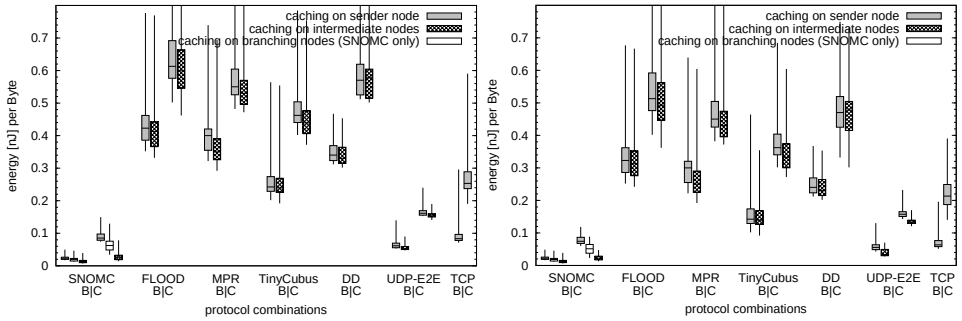
Figure 5.4: Evaluation: number of transmitted packets.

The results for the transmission of 20 bytes are shown in Fig. 5.4(b). TinyCubus achieves the best performance in combination with NullMAC. It has an optimal set of forwarding nodes and NullMAC keeps the radio transceiver always awake. Hence, this combination reaches the minimal number of packets (6) to ensure the successful transmission of 20 bytes. The other broadcast protocols (Flooding and Multipoint Relay) show an improved performance as well, considering the scenario with 1000 bytes. Just one packet has to be transmitted, which causes less collisions and retransmissions. Out of all non-broadcast-based protocols, SNOMC has the best performance while Directed Diffusion has the worst. Directed Diffusion requires a larger number of packets because of the more extensive hop connectivity, i.e., more connections compared to UDP and TCP. If Directed Diffusion would use the same sensor nodes as, e.g., UDP or TCP, it would perform better. Further, the differences between the three caching modes are quite small, a result of the smaller number of required retransmissions.

5.2.5 Energy Consumption

We now discuss the energy consumed per node and per transmitted byte. Results for the scenario with 1000 bytes are shown in Fig. 5.5(a). We compare only the performance of BEAM and ContikiMAC, since NullMAC does not have an energy saving mechanism. In general, it can be seen that for broadcast-based protocols there is a stronger relation between the consumed energy and the number of transmitted bytes. More specifically, the more bytes are transmitted the higher is the energy consumption per byte due to higher packet loss. The energy consumption of unicast-based protocols is generally good with the exception of Directed Diffusion, which performs rather poor due to additional maintenance messages, e.g., for path reinforcement or the propagation of new interest. Concerning the impact of the MAC protocol, BEAM offers higher energy-efficiency than ContikiMAC since the latter has a worse duty cycle mechanism compared to the adaptive duty cycle

mechanism of BEAM. Furthermore, caching does not significantly influence the performance of broadcast-based protocols from an energy point of view.



(a) 1000 bytes (b) 20 bytes

Figure 5.5: Evaluation: energy consumption per node and per transmitted byte.

In Fig. 5.5(b) corresponding results on energy consumption are shown for the scenario with a single packet (20 bytes). Due to the transmission of 20 bytes implies a lower number of collisions and retransmissions, the energy consumption per transmitted byte is lower for 20 bytes (1 packet) than for 1000 bytes (15 packets).

5.3 SNOMC Evaluation in Real-World Testbed

In this section we present the results of the evaluation under real-world conditions. We measure the transmission time, the number of transmitted packets and the consumed energy transmitting 1000 bytes from one sender node to three receiver nodes.

For the evaluation we are using our in-house testbed developed in the Wisebed project. It consists of 40 TMoteSky sensor nodes, which are distributed in the building of the Institut für Informatik und angewandte Mathematik. More detailed information about Wisebed can be found in Section 2.3.

First, the used protocol stack is described. Afterwards, we show the three used evaluation scenarios followed by the measurements for time consumption, transmitted packets, and energy consumption.

5.3.1 Protocol Stack

To evaluate the performance of SNOMC, we compare it to a number of data dissemination protocols commonly found in wireless sensor networks in combination with different underlying MAC protocols. In addition to the implementation and evaluation in OMNeT++ we implemented with PSFQ (Pump Slowly, Fetch Quickly) protocol another transport protocol. Moreover, we do not evaluate SNOMC and the other data dissemination protocols with the BEAM MAC protocol. The reason is that BEAM implemented in Contiki OS does not support

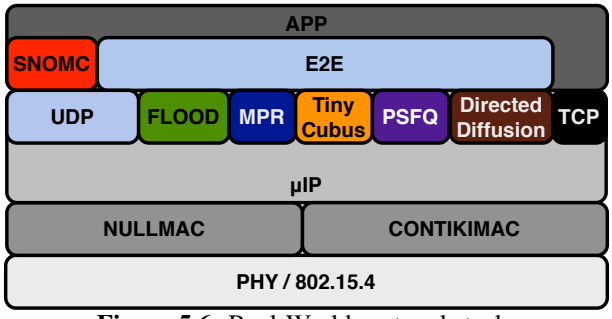


Figure 5.6: Real-World protocol stack.

broadcast transmissions, which is necessary for the broadcast-based protocols and the broadcast optimization of SNOMC.

Thus, the used protocols for the evaluation of SNOMC are: Flooding, Multipoint Relay, TinyCubus, PSFQ, Directed Diffusion, UDP, and TCP in combination with NullMAC and ContikiMAC. For the description of the protocols we refer to Section 2.6. All protocols have been implemented in Contiki OS [38]. The protocol stack is shown in Fig. 5.6 and is based on the µIP stack from Contiki OS. To ensure a fair comparison with SNOMC, we also implemented in the real-world evaluation the same simple NACK-based reliability mechanism, used in SNOMC, on top of all data dissemination protocols (except for TCP).

5.3.2 Experimentation Scenarios

For the evaluation of SNOMC we used three different scenarios in the in-house testbed . The first scenario, shown in Figure 5.7, consists of all 40 sensor nodes in the in-house testbed. We have one sender node (number 7), and three receiver nodes (numbers 16, 19, and 21). The SNOMC distribution tree spans over the nodes 4, 40, 23, and 1. The rest of the nodes are passive in transmitting the data using SNOMC but they play an active role while transmitting data using Flooding or Multipoint Relay.

Since most of the broadcast-based protocols did not terminate and thus failed in scenario 1 with all 40 nodes (cf. Section 5.3.3), we choose a second scenario (see Figure 5.8). This scenario has the same sender node, the same receiver nodes, and the same spanned distribution tree for SNOMC. But it does not include all the other passive sender nodes. This leads to much less traffic and transmitted packets using the broadcast-based protocols and a better performance.

The third scenario, shown in Figure 5.9 is used to compare SNOMC specifically with UDP-E2E. The sender node is again node number 7, but receiver node 16 changed to receiver node 21. Thus also the spanned distribution tree changed and consists of the nodes number 4, 40, and 1. This scenario shows the advantages of a multicast-based data distribution scheme compared to a unicast-based one.

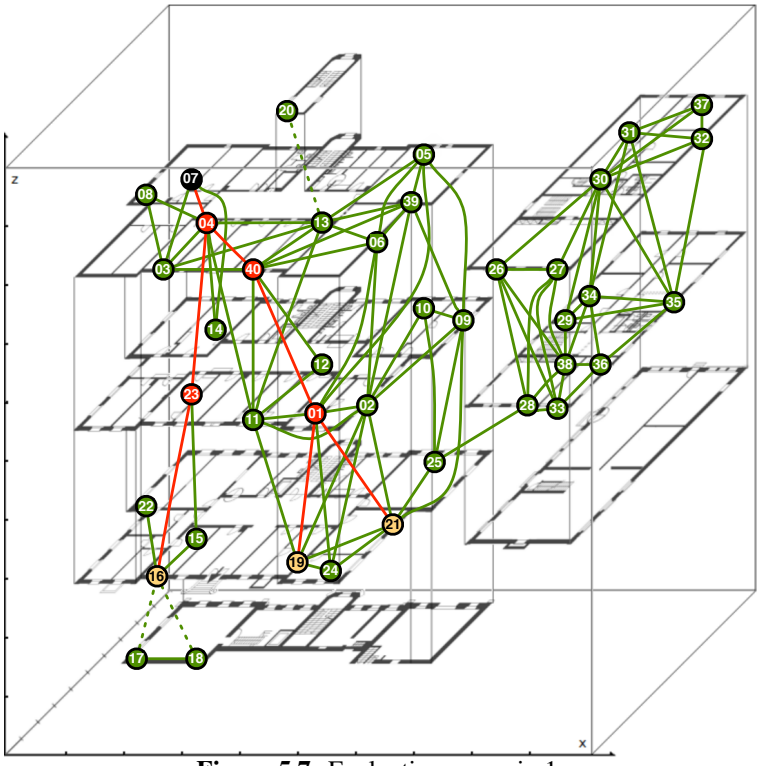


Figure 5.7: Evaluation scenario 1.

5.3.3 Transmission Times

In this section we present the results for the transmission time in the three scenarios. In all figures on the x-axis the combinations of data dissemination protocols with the different caching strategies are shown. In our notation **s** stands for caching only on sender node, **b(-pa)** for caching on branching nodes (with pro-active request), **i(-pa)** caching on each intermediate sensor node (with pro-active request), and **i-bc** stands for the broadcast optimization of SNOMC. With pro-active mode each intermediate node that caches data, requests actively missing fragments. Caching on branching nodes and the broadcast optimization appears only in SNOMC. The diagrams on the left side show the measured values using NullMAC, and the diagram on the right side shows the values using ContikiMAC. Both diagrams have the same scale on the y-axis to ease the comparison between NullMAC and ContikiMAC.

Figure 5.10(a) shows the transmission times transmitting 1000 bytes from one sender node to the three receiver nodes using NullMAC. Compared to the other protocols SNOMC has the best performance.

Only the performance of UDP-E2E comes close to SNOMC. Broadcast-based protocols such as Flooding and MPR work generally worse than unicast-based protocols except for TCP. The reason is obvious. All packets (`data` as well as `nack` messages) are broadcasted which leads to a huge amount of messages in the network and causes high interferences. The three broadcast-based protocols differ in

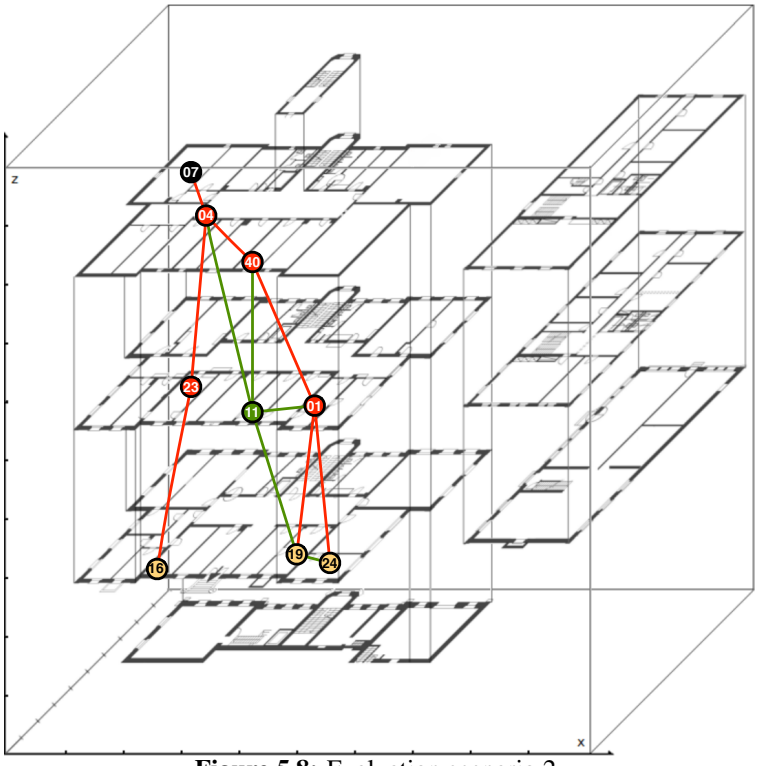


Figure 5.8: Evaluation scenario 2.

how many sensor nodes rebroadcast the packets. Therefore, Flooding has the worst performance, followed by MPR, and TinyCubus. In case of using TinyCubus only the sensor nodes 4, 40, 23, and 1 rebroadcast the packets. Thus, there is less traffic in the network and TinyCubus performs better.

Directed Diffusion does not work and does not deliver results. The reason is that the found paths are always very bad and in combination with the bad links there are no stable connections. TCP performs poorly because each packet is acknowledged, and not only the missed one, leading to higher traffic and interferences.

Comparing the various caching strategies, we see that caching on each intermediate nodes show the best results. In case of SNOMC caching on each intermediate node performs better than caching on branching nodes. Thus, we can say that the more often the packets are cached the better the performance is. Proactively requesting missed packets results on higher packet traffic in the network. Thus, always caching with pro-active requests has a worse performance compared to caching strategies without pro-active requests.

If we have a look on Figure 5.10(b), we see the results using ContikiMAC. As mentioned above, we see that a number of protocols does not work in this scenario. Only SNOMC, UDP-E2E, TinyCubus and partly Flooding show meaningful results at all. The other protocols fail. The reason is how ContikiMAC handles broadcasts. In the unicast case ContikiMAC sends a frame and gets back an acknowledgement.

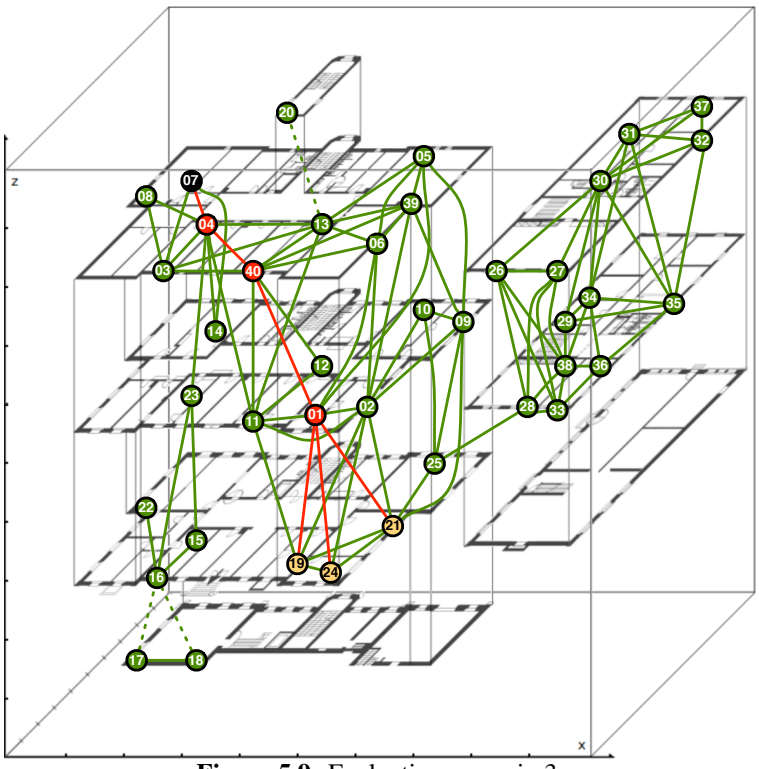


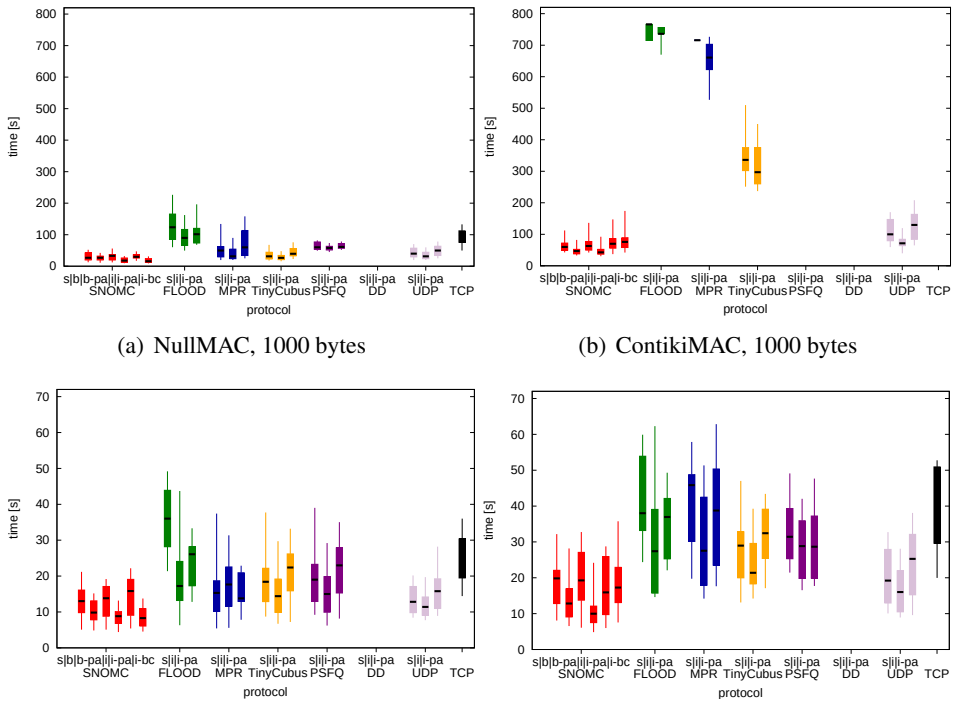
Figure 5.9: Evaluation scenario 3.

If the neighbour sleeps then there is no acknowledgement. ContikiMAC sends the frame until the neighbour wakes up and answers. In the broadcast case there are no acknowledgements. Hence, ContikiMAC sends as long as for the maximum duty cycle among the neighbours so that it can be ensured that every neighbour was awake during this time. This produces a huge amount of traffic and interferences.

Figures 5.10(c) and 5.10(d) show the results transmitting 20 bytes from one sender to the receiver nodes using NullMAC and ContikiMAC respectively. The results are similar to the 1000 byte scenario. SNOMC has a smaller advantage compared to UDP-E2E and TCP because the joining phase takes additional time before the delivery of the data can start. Using NullMAC (see Figure 5.10(c)) the transmission times are lower than the transmission times using ContikiMAC (see Figure 5.10(d)). Directed Diffusion also does not deliver results using both NullMAC and ContikiMAC, because it was not possible to establish stable routes in this scenario.

Since Directed Diffusion and PSFQ have phases in the protocol operation that use broadcast transmission, also theses protocols fail using ContikiMAC as MAC protocol. Therefore, we run the same experiments on a smaller scenario, which only consists of a sub-set of nodes, namely only 9 instead of 40. The results are shown in Figure 5.11.

Figure 5.11(a) shows the results using NullMAC. The results from SNOMC,



(a) NullMAC, 1000 bytes

(b) ContikiMAC, 1000 bytes

(c) NullMAC, 20 bytes

(d) ContikiMAC, 20 bytes

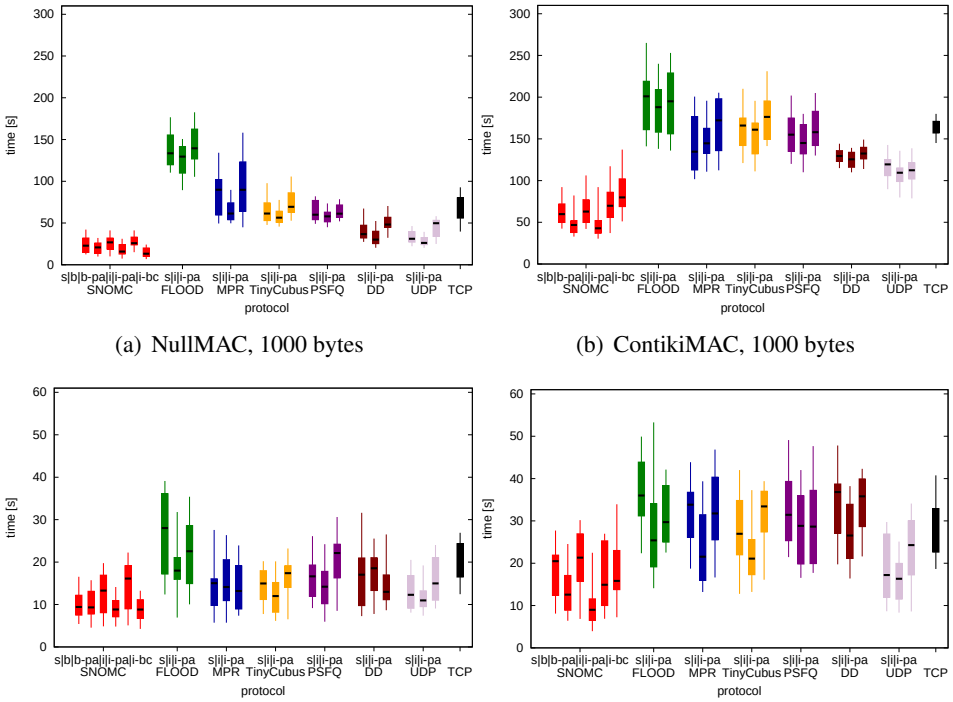
Figure 5.10: Evaluation: transmission time, scenario 1.

UDP-E2E, and TCP are almost the same as in the large 40-node scenario, because the same set of nodes are affected as in scenario 1. We can see that SNOMC outperforms all protocols.

Also in this smaller scenario 2 broadcast-based protocols such as Flooding, MPR, and TinyCubus perform poor. The reasons are still the same as in scenario 1. A lot of packets in the network leading to high interferences and thus many retransmissions. Directed Diffusion and PSFQ perform better than the broadcast-based protocols. Directed Diffusion finds good paths in the interest phase to deliver the data efficiently.

Also the results in this scenario show that the caching strategy has significant influence. Caching only on the sender node performs worse than caching on each intermediate node and pro-active requests of missing packets cause worse performance due to higher traffic and more interferences.

While most of the data dissemination protocols failed in scenario 1 using ContikiMAC, we have results for all of them in scenario 2, as shown in Figure 5.11(b). SNOMC outperforms all other protocols, broadcast-based protocols perform worst, and the different caching strategies have a certain influence. If we look at the broadcast optimization of SNOMC we see the strong influence of the underlying MAC protocol. In combination with NullMAC the broadcast optimization performs very well. It shows the overall best performance of all protocol and caching combina-



(a) NullMAC, 1000 bytes

(b) ContikiMAC, 1000 bytes

(c) NullMAC, 20 bytes

(d) ContikiMAC, 20 bytes

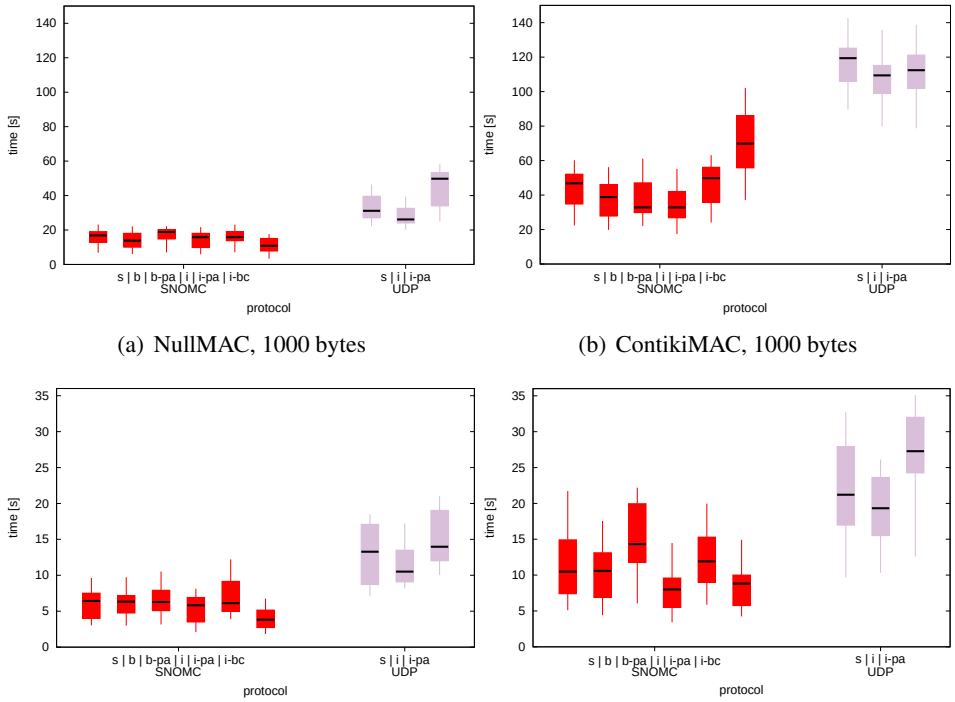
Figure 5.11: Evaluation: transmission time, scenario 2.

tions. In combination with ContikiMAC the broadcast optimization has an opposite impact, since ContikiMAC handles broadcast transmissions inefficiently.

Figures 5.11(c) and 5.11(d) show the results transmitting 20 bytes in scenario 2 using NullMAC and ContikiMAC respectively. The difference in performance between the broadcast-based protocols and both SNOMC and UDP-E2E is smaller in scenario 2 than in scenario 1, because just a small number of nodes are affected. Additionally, only one packet has to be transmitted which leads to less traffic and thus less packet loss and necessary retransmissions.

To have a comparison to the hardest competitor of SNOMC, UDP-E2E, we choose a third scenario to compare both protocols. Figure 5.12 shows diagrams with the results using NullMAC and ContikiMAC. We see that SNOMC outperforms UDP-E2E by factor of 2 to 2.5 with both MAC protocols due to the reduced number of hops. With UDP-E2E at least 18 hops are necessary to transmit a packet to all three receiver nodes. In case of using SNOMC just 6 hops, or even 4 hops with the broadcast optimization. Thus, we see the advantage of the overlay multicast communication scheme compared to a unicast-based one. UDP-E2E has very low overhead, because it does not have a joining phase but the reduced number of hops in the distribution tree compensates the joining overhead.

In both diagrams we again see the effect of the caching strategy and the effect of ContikiMAC for the broadcast optimization of SNOMC.



(a) NullMAC, 1000 bytes

(b) ContikiMAC, 1000 bytes

(c) NullMAC, 20 bytes

(d) ContikiMAC, 20 bytes

Figure 5.12: Evaluation: transmission time, scenario 3.

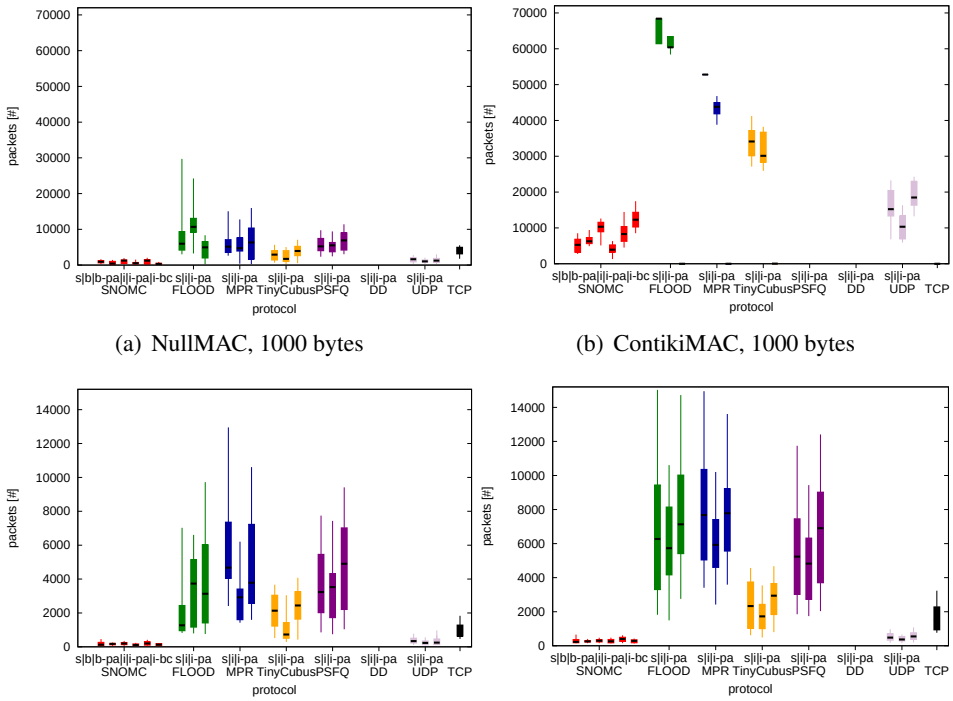
5.3.4 Number of Transmissions

The number of transmissions is another metric that shows how efficiently a protocol operates. The more packets are necessary to deliver a certain number of bytes the more inefficient is the protocol.

Figure 5.13 shows the number of transmitted packets with 1000 bytes from one sender to three receivers. These are results for scenario 1 with the combination of NullMAC or ContikiMAC. We can see that the broadcast-based protocols require much more transmissions to deliver the data to the receivers compared to the unicast-based protocols. SNOMC outperforms all other protocols also in this metric.

Directed Diffusion, PSFQ, TCP, and partly TinyCubus, MPR and Flooding fail in combination with ContikiMAC in scenario 1. In combination with NullMAC only Directed Diffusion fails.

Figures 5.13(c) and 5.13(d) show the results of transmitting 20 bytes in scenario 1. Also in the 20 bytes scenario the broadcast-based protocols require a significantly large number of transmissions because the whole network with all 40 nodes get flooded. In contrast to that, SNOMC and UDP-E2E just require a small number of transmission to deliver the 20 bytes to the receivers. Just as in the 1000 bytes scenario using ContikiMAC takes more necessary transmissions than using NullMAC. The combination of NullMAC and Directed Diffusion fails also in this



(a) NullMAC, 1000 bytes

(b) ContikiMAC, 1000 bytes

(c) NullMAC, 20 bytes

(d) ContikiMAC, 20 bytes

Figure 5.13: Evaluation: number of transmitted packets, scenario 1.

scenario, because Directed Diffusion was not able to stable stable routes.

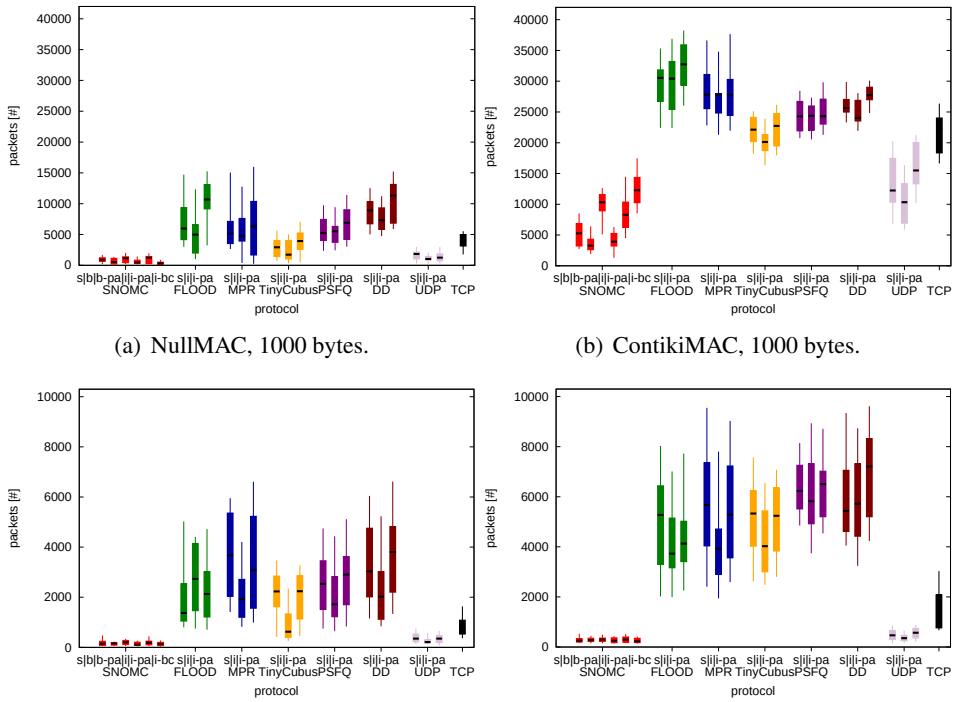
Therefore, we used scenario 2 with much less sensor nodes to evaluate the protocols. The results are shown in Figure 5.14.

SNOMC requires a somewhat lower number of transmissions than UDP-E2E. SNOMC requires additional packets for the joining phase. It has a slight advantage in number of hops due to the use of a distribution tree instead of using three parallel unicast connections as UDP-E2E uses.

The broadcast-based protocols use a high number of transmissions due to broadcasting and rebroadcasting of the packets, although only a small number of nine nodes (see Figure 5.11) are rebroadcasting the packets. Directed Diffusion requires many packets in the Interest phase, which uses broadcast transmissions to find the adequate paths. Also PSFQ uses broadcast transmissions in its pump operation.

Comparing the two MAC protocols we see that ContikiMAC requires much more transmissions than NullMAC, especially for the broadcast-based protocols. The reason is how ContikiMAC handles broadcast transmissions. As described in Section 2.6.1 ContikiMAC rebroadcast a packet for the full wake-up interval (128ms). We also see this effect comparing the SNOMC broadcast optimization for NullMAC and ContikiMAC.

Also the influence of the caching strategy can be seen in theses diagrams. Caching on the sender node requires end-to-end retransmissions of missed frag-



(a) NullMAC, 1000 bytes.

(b) ContikiMAC, 1000 bytes.

(c) NullMAC, 20 bytes.

(d) ContikiMAC, 20 bytes.

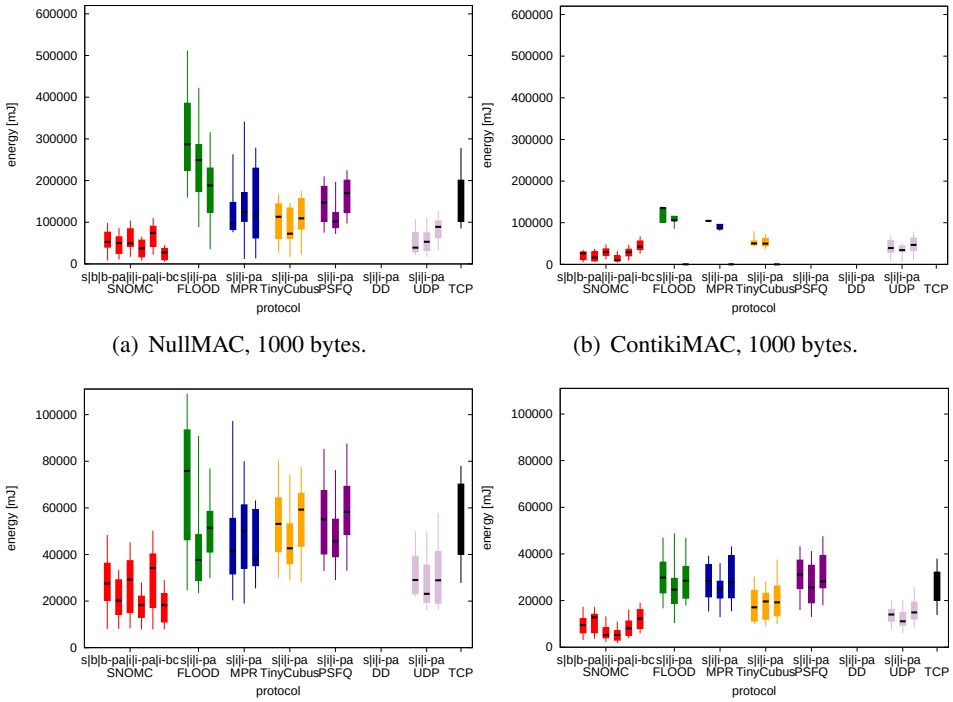
Figure 5.14: Evaluation: number of transmitted packets, scenario 2.

ments, which leads to a high number of transmissions. The `nack` message has to be transmitted from the receiver node to the sender node and the missed fragment has to retransmitted from the sender node back to the receiver node. If the missed fragment is cached on a branching or intermediate node, an end-to-end retransmission is not required. Thus, caching on intermediate nodes (or branching nodes) decreases the number of hops (and transmissions) for a packet request.

Figures 5.14(c) and 5.14(d) show the results of transmitting 20 bytes in scenario 2. The results are similar to the results of scenario 1 (see Figures 5.14(c) and 5.14(d)). SNOMC and UDP-E2E requires significantly less transmissions than Flooding, MRP and TinyCubus. Directed Diffusion and PSFQ perform as poor as the broadcast-based protocols because both have protocol phases, which depend also on broadcasts.

5.3.5 Energy Consumption

Energy consumption is measured for all nodes individually and is accumulated for the whole network. Thus, in scenario 1 energy consumption of 40 nodes is measured and accumulated. The consumed energy includes the energy for transmitting and receiving packets, the operation of the micro-controller on the sensor node, and reading/writing flash memory.



(a) NullMAC, 1000 bytes.

(b) ContikiMAC, 1000 bytes.

(c) NullMAC, 20 bytes.

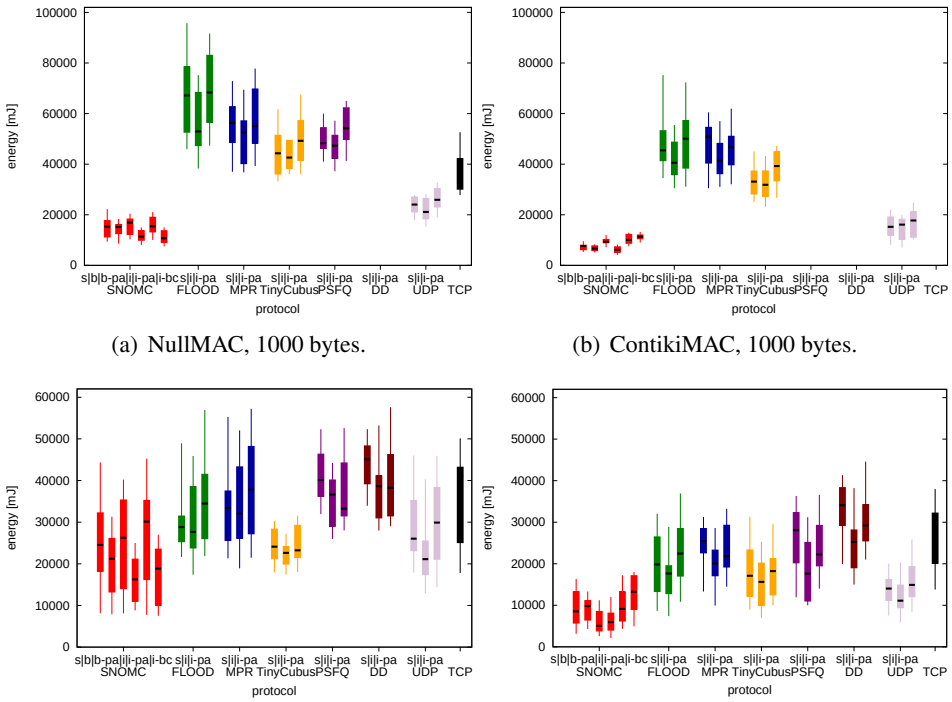
(d) ContikiMAC, 20 bytes.

Figure 5.15: Evaluation: energy consumption, scenario 1.

Figure 5.15(a) shows the results in energy consumption using NullMAC. Since, NullMAC is an always on protocol (without any energy saving mechanisms) the measured energy values behave as the transmission times - the faster the transmission of the 1000 bytes finishes, the less energy is needed for the transmission. Comparing the results using ContikiMAC (cf. Figure 5.15(b)), we again see that a number of protocol combinations failed such as Directed Diffusion and PSFQ.

Figures 5.15(c) and 5.15(d) show the results in energy-consumption transmitting 20 bytes in scenario 1. Naturally, less energy is consumed transmitting just 20 bytes than 1000 bytes. But the relation between the compared protocols is similar to the 1000 byte scenario. SNOMC requires less energy than the other protocols independent of the underlying MAC protocol. Using NullMAC requires more energy than using ContikiMAC, because NullMAC has no energy-saving mechanisms.

Due to the problems in scenario 1 described in Section 5.3.2, we re-run the experiments using scenario 2, with just 9 sensor nodes. The results are shown in Figure 5.16. Using ContikiMAC (cf. Figure 5.16(b)) leads to longer transmission time but also to lower energy consumption. ContikiMAC uses duty cycles as energy-saving mechanism so that the sensor node go to sleep when there is nothing to transmit or receive. The sensor nodes awake periodically and check if there is an incoming packet. Thus, considerable energy can be saved. The energy-consumption depends on the transmission time, the number of transmitted and re-



(a) NullMAC, 1000 bytes.

(b) ContikiMAC, 1000 bytes.

(c) NullMAC, 20 bytes.

(d) ContikiMAC, 20 bytes.

Figure 5.16: Evaluation: energy consumption, scenario 2.

ceived packets and the duty cycles of the sensor nodes.

Saving energy using ContikiMAC works fine when we have mainly unicast transmissions as in SNOMC, UDP, TCP, and partly Directed Diffusion and PSFQ. In these cases we save a lot of energy compared to using NullMAC (approximately by the factor of 2 to 3). For the broadcast-based protocols such as Flooding, MPR and TinyCubus we save less energy, explained by the way how ContikiMAC handles broadcast transmissions. It broadcasts packets for a time period of 128ms and during this time the node does not go to sleep. Therefore, broadcast-based protocols using ContikiMAC save less energy (approximately by the factor of 1.5 to 2) than the other data dissemination protocols (SNOMC, UDP-E2E, TCP).

The results using NullMAC in scenario 2 are shown in Figure 5.16(a). Since NullMAC does not save energy, the results for energy consumption correspond to the results for transmission time.

Figures 5.16(c) and 5.16(d) show the results for transmitting 20 bytes in scenario 2. Also here we can see the expected results. SNOMC and UDP-E2E require less energy than the other data dissemination protocols.

Summarizing the results of the experiments in the real-world testbed we can see that SNOMC outperforms the other data dissemination protocols in terms of transmission time, energy consumption, and number of transmitted packets in different scenarios. SNOMC outperforms unicast-based protocols such as UDP-E2E

because it requires a lower number of hops to deliver the data. In big scenarios with a large number of sensor nodes broadcast-based protocols such as Flooding, MPR, TinyCubus, and PSFQ perform very bad or even fail to transmit the data to the receiver nodes. The energy consumption strongly depends on the underlying MAC protocol. Using energy-saving ContikiMAC saves a lot of energy compared to NullMAC. SNOMC performs very good independent of the underlying MAC protocol.

5.4 Comparison of Simulated and Real-World Results

Our main observation is that simulation results differ from the results of the real-world experiments. The reasons for that difference are mainly in the implementation of the protocols in the OMNeT++ simulator.

One major aspect is the handling of the packet in the protocol stack. In OMNeT++ all processing time of a packet in the protocol stack is zero. In Contiki OS the transfer of a packets through the several protocol layers from the radio to the application layer takes a certain amount of time. Since Contiki OS uses protothreads, processes can be interrupted and thus the processing time of packets can differ a lot.

Another fact that causes difference in results is that broadcasting of ContikiMAC was implemented in OMNeT++ in a different way compared to the implementation in Contiki OS. In the OMNeT++ implementation it does not broadcast the packets for a certain time period as in the Contiki OS implementation. Instead, in the OMNeT++ implementation ContikiMAC broadcasts a packet just three times in a row as it is done in the OMNeT++ implementation of NullMAC. This fact also causes difference between simulations results and the results of the real-world experiments.

Although, in OMNeT++ a realistic radio model was implemented to gain realistic results, our experiments using testbed showed that in reality interferences have a much higher impact on transmissions. Additional external interferences are taken into account in the implementation of the radio model but no effect such as reflection or multi-path fading. Due to the in-house testbed is located in the institutes building this effects occur regularly and influence the results strongly.

There are some differences among the scenarios. The distance between the the sensor nodes in the simulation scenarios is much higher than in the in-house testbed scenarios. Therefore, on the in-house testbed, due to more overlapping interference ranges, transmitted packets experience higher collision probabilities. This is especially the case for broadcast-based protocols. To avoid these problems several timers such as the retransmission timer, random back off timer, etc, in the Contiki OS implementation were given much higher values compared to ones in the OMNeT++ implementation.

Beside the differences between the simulation results and the real-world results we have also commonalities. The relation in the results between the different

protocols in the simulation and in the real-world experiments are similar. We can see that in both evaluation environments broadcast-based protocols perform much worse than the other protocols. SNOMC performs always best independent of of the underlying MAC protocol.

The main advantage of the evaluation of SNOMC in the OMNeT++ simulator is that it allows an easy implementation of the protocol and thus a fast proof-of-concept. The main advantage of the evaluation of SNOMC in Wisebed is that it helps to optimize the protocol for real-world environments.

5.5 Conclusions

We propose the Sensor Node Overlay Multicast (SNOMC) protocol to support a reliable, time-efficient and energy-efficient dissemination of bulky data from one sender node to many receivers. To ensure end-to-end reliability we designed and implemented a NACK-based reliability mechanism. Further, to avoid costly end-to-end retransmissions we propose different caching strategies implemented in SNOMC.

In this chapter we evaluated SNOMC against a number of data dissemination protocols for wireless sensor networks, such as Flooding, MPR (Multipoint Relay), PSFQ (Pump Slowly, Fetch Quickly), TinyCubus, and Directed Diffusion as well as the unicast-based protocols UDP and TCP. To ensure a fair competition we implemented the same reliability mechanisms and caching strategies on top of the selected protocols. As underlying MAC protocols we chose BEAM, ContikiMAC and NullMAC for the simulations, and ContikiMAC and NullMAC for the real-world evaluation.

As proof-of-concept of SNOMC we made an evaluation in the OMNeT++ simulator. For evaluating the performance of SNOMC in real-world we run a number of experiments in our in-house Wisebed testbed. We choose three performance metrics: transmission time, number of transmitted packets and energy consumption.

We showed that the SNOMC protocol performs very well compared to the selected data dissemination protocols. Especially, broadcast-based data dissemination protocols, such as Flooding, MPR, and TinyCubus cause broadcast storms and thus perform very poor. Data dissemination protocols especially designed for wireless sensor networks, such as Directed Diffusion or PSFQ, have some protocol phases also relying on broadcast transmissions. Thus, also these protocols perform worse than SNOMC. The most serious competitor of SNOMC is UDP-E2E. In this case the advantage of SNOMC depends on the topology of the network and the distribution tree as we showed in Section 5.3.3. Especially in scenario 3 SNOMC shows its advantage compared to UDP-E2E, because the forks in distribution tree are close to the receivers.

Further, we showed the influence of the different caching strategies. In general, we can say that caching on intermediate nodes improves the performance of each

protocol, since expensive end-to-end retransmissions are avoided. We also showed that pro-active requesting of missed fragments does not improve the performance at all. Pro-active requesting causes additional packets, higher traffic and thus much more collisions, which cancels the advantage of the intended optimization.

Moreover, we showed that SNOMC performs well with different underlying MAC protocols that support different levels of reliability and energy-efficiency. It proves that SNOMC can be used in different environments and in different systems, independently of the used protocol stack.

We therefore conclude that SNOMC can offer a robust, high-performing solution for the efficient distribution of code updates and management information in a wireless sensor network. To further show the usability of SNOMC, we integrate it into the MARWIS architecture as described in Chapter 8 and evaluate the performance of SNOMC in MARWIS (presented in Section 8.4).

Part II

MARWIS: A Management Architecture for Wireless Sensor Networks

Chapter 6

Management Architecture and Protocol Design

In this chapter we present the design and architecture of MARWIS (Management Architecture for Wireless Sensor Networks). MARWIS [127, 123, 14] is a management architecture for heterogeneous wireless sensor networks that supports monitoring, configuration and code updating of the wireless sensor network in general and the sensor nodes in particular.

Section 6.1 describes the problem we address with MARWIS and motivates the need of such architecture. Section 6.2 describes a typical management scenario and corresponding management tasks such as monitoring, configuration and code updating. Section 6.3 goes into design details of the architecture and its components, namely, management station, mesh node with MARWIS Server, and different types of sensor nodes each running with the Sensor Node Agent. Subsequently, Section 6.4 describes the management protocols needed to support the three management tasks. Finally, Section 6.5 concludes the chapter.

6.1 Introduction

A heterogeneous wireless sensor network consists of several different types of sensor nodes. Various applications supporting different tasks, e.g., event detection, localization, tracking, and environmental monitoring, may run on these specialized sensor nodes. In addition, new applications have to be deployed as well as new configurations and bug fixes have to be applied during network lifetime. In a network with thousands of nodes, this becomes a very complex task and a general management architecture is required.

One of the challenges that the management architecture should address is how to achieve that monitoring, configuration, and code updating can be performed on heterogeneous sensor nodes for the whole duration of their lifetime? Another question is how to structure such a heterogeneous wireless sensor network to handle management tasks efficiently and automatically over the network?

We propose the usage of a wireless mesh network as a backbone to build a

heterogeneous wireless sensor networks. The proposed new management architecture called MARWIS supports common management tasks such as monitoring and configuring the wireless sensor network, and disseminating code updates of new program versions of the applications running on the sensor nodes.

Beside these management tasks there are few other general features, which the architecture should support:

1. reliable communication,
2. time-efficient communication,
3. energy-efficient operation,
4. support of heterogeneous sensor nodes.

Since code updating and network configuration are both critical tasks, end-to-end reliability is required. This means that the user should be informed if an update was successful or not. Although monitoring, (re)configuration and code updates are no high priority tasks, they should be executed in reasonable time. Energy-efficient operation of the management tasks plays a very important role, given the limited power capacity of the battery-operating sensor nodes. Furthermore, the architecture should be able to service heterogeneous sensor node platforms. The qualitative and quantitative evaluation of these features can be found in the Chapter 7 addressing the MARWIS implementation.

6.2 Management Scenario and Tasks

Many different applications may run in a wireless sensor network, e.g. as mentioned before, event detection, localization, tracking, and monitoring. Therefore, different types of sensor nodes, which can measure different sensor values and perform different tasks, are required. Existing sensor node platforms in general have different radio modules, which are not interoperable. Sensor nodes of the same type build a sensor sub-network, which is not able to communicate directly to another sensor sub-network. Several such sensor sub-networks build a heterogeneous wireless sensor network. To interconnect such a heterogeneous wireless sensor network mesh nodes are proposed as gateways between the sensor sub-networks. A sensor node plugged into a serial interface (e.g. USB) of a mesh node works as gateway. The wireless mesh nodes communicate among each other via IEEE 802.11.

A possible scenario of a heterogeneous wireless sensor network is shown in Fig. 6.1. Sensor nodes are depicted in green, red, turquoise and khaki. The mesh nodes are depicted by grey circles. Each mesh node operates a sensor sub-network, which consists of sensor nodes of the same type and hence communicate to each other. The sensor node plugged into a mesh node via USB, serial, or another interface operates as sensor node gateway to the specific sensor sub-network. A management station is a mesh node in the mesh network with additional functionality

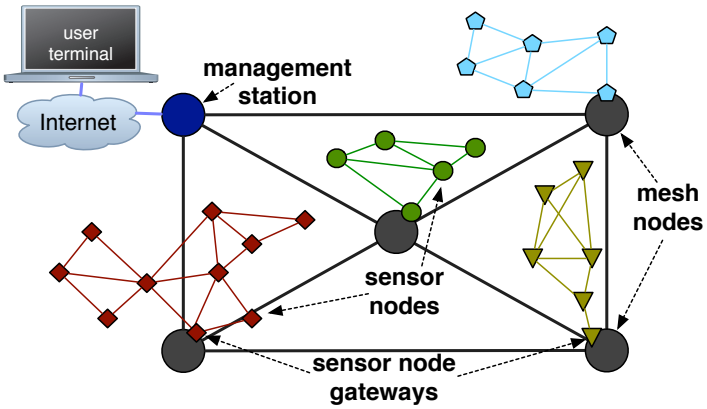


Figure 6.1: A possible scenario for heterogeneous wireless sensor networks with management devices.

such as a web server. It creates the connection between the user and the wireless mesh network including the wireless sensor networks.

The use of an architecture with a wireless mesh network as backbone has various advantages. The main benefit is the ability to communicate with different types of sensor nodes in several sensor sub-networks. Another advantage of using a wireless mesh network is that a new sensor node platform can be easily inserted into the heterogeneous wireless sensor network by plugging a sensor node gateway into a mesh node. Moreover, the use of a wireless mesh network has advantages by subdividing a large wireless sensor network into smaller sensor sub-networks. A wireless sensor network consisting of thousands of nodes and one base station creates many communication problems. Most of them are caused by the high number of hops in larger wireless sensor networks. Subdivision into smaller sensor sub-networks limits the sensitive sensor node links to three or four hops to the next sensor node gateway. Connections with lower number of hops result in a better communication performance with a clearly lower packet delay, jitter and packet loss. Sensor nodes in the vicinity of the sink also benefit from preserved energy and processing power, because they do not have to forward the whole wireless sensor network traffic.

In addition to the communication gateway functions, mesh nodes further perform management tasks for the heterogeneous wireless sensor networks. As result, the limited sensor nodes have less management functions to perform, which decreases memory and computation requirements towards them.

In addition to the mesh nodes, which provide the management functionality, there are one or more management stations. With their support a user can perform a large set of management tasks. In a heterogeneous wireless sensor network with a large number of heterogeneous sensor nodes a comprehensive management architecture is required. From the management point of view three tasks are required to operate a heterogeneous wireless sensor network.

1. monitoring the wireless sensor network and the sensor nodes,

2. (re)configuring the wireless sensor network and the sensor nodes,

3. updating and reprogramming the sensor nodes.

Monitoring includes visualization at the management station of all sensor nodes in the various sensor sub-networks. Furthermore, status information about the sensor nodes has to be continuously collected and displayed. This includes sensor node hardware features (micro-controller, memory, transceiver), sensor node software details (operating system versions, protocols, applications), dynamic properties (battery, free memory), and, if available, position information. *Configuration* includes configuring the sensor nodes, the running applications or the network. Examples are configuring the sensing intervals or configuring the communications protocols (duty cycles, timers, etc). *Updating and reprogramming* the sensor nodes is a another important task. Both the operating system and applications must be updated, fully or partially. In a large wireless sensor network manual execution of this task is not feasible. A mechanism to handle this automatically and dynamically over the network is required. Mechanisms to handle incomplete, inconsistent, and failed updates have to be provided as well.

6.3 Management Architecture

The proposed MARWIS architecture to manage heterogeneous wireless sensor networks reliably, time- and energy-efficiently contains the following structural elements: one or more management stations running the customized Linux distribution ADAM [113], several mesh nodes as management nodes running the MARWIS server, sensor node gateways plugged into a mesh node, and the different types of sensor nodes running the sensor node agent. These elements are shown in Figure 6.2.

In the next sections the detailed description of the three components can be found.

6.3.1 Management Station with Management System for Wireless Mesh Networks

The **management station** (shown in Figure 6.2(a)) is divided into two parts. First, it contains of a **user terminal** to access a web-based graphical user interface (GUI) to control the wireless sensor network. The **user interface** displays the wireless sensor network topology with the mesh nodes including the subordinate sensor nodes and information about the state of the sensor nodes(**1**).

Second, it contains a mesh node running **ADAM**, a management system for wireless mesh networks [113]. ADAM is a small Linux distribution including all required applications, especially a HTTPS server to handle the requests and transmit them to the sensor nodes. Further, the management station contains a web

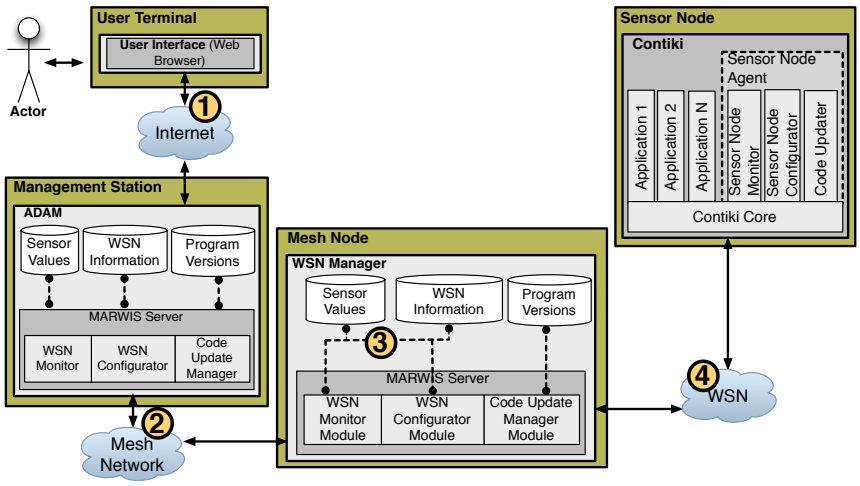


Figure 6.2: Architecture of the MARWIS components.

server and a **MARWIS server** with three program modules, such as a **WSN monitor**, a **WSN configurator**, and a **code update manager**. To store and backup information about the wireless sensor network and the sensor nodes centrally on the management station it also contains three databases for sensor values, for wireless sensor network information, and for the program versions of the applications running on the sensor nodes.

All communication in the network between the mesh nodes as well as between the sensor nodes is done via TCP/IP **(2)** and SNOMC [126, 125] (see Chapters 3 - 5).

6.3.2 Mesh Node with MARWIS Server

The **WSN manager** is located on every mesh node and shown in Figure 6.2(b). It also runs on ADAM and provides the management functionality for the different sensor sub-networks. It consists of three databases and the **MARWIS server** with three program modules, namely a WSN monitor, a WSN configurator, and a code update manager (as shown in Figure 6.2).

The **WSN information database** stores all information about the sensor nodes and the WSN, such as topology (neighbours, addresses), and states of the sensor nodes (battery, memory). The **program version database** stores all versions of all programs for all platforms, which can be installed on the sensor nodes. Finally, the **sensor value database** stores all data measured by the sensors. All databases are accessible by an API to get and store data **(3)**.

The **WSN monitor** 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 sensor nodes into the databases. The **WSN configurator** is responsible for the configuration tasks. It queries properties from the sensor nodes and stores them in the WSN information database. The **code update manager** stores newly received program images (and related information) in the program version database and notifies the management station about available programs. To execute the updating process, it transmits the image to the sensor node.

The communication between the mesh node and the sensor network uses µIP [34] and SNOMC **(4)** and the communication between the mesh nodes uses TCP/IP **(5)**.

6.3.3 Sensor Node with SN Agent

On the sensor node, the management tasks are handled by a **SN agent** (as shown in Fiqure 6.2(c)). 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 mesh node. The **SN configurator** executes the configuration requests and notifies the mesh node. The **code updater** is responsible for the code replacement on the sensor node. It receives the program image of the application and performs the update by loading the new module and replacing the old one. Finally, it informs the mesh node about the success of the update.

The communication between the mesh node and the sensor network uses µIP and SNOMC **(6)**.

6.4 WSN Management Protocols

We consider monitoring, reconfiguration, and code updating the most important tasks of our management architecture. This section provides the description of the management protocols operation and the relation to the architectural components. The management protocols are:

- WSN Monitoring Protocol
- WSN Configuration Protocol
- WSN Code Update Protocol

6.4.1 WSN Monitoring Protocol

Monitoring a wireless sensor network enfolds, first, static information about the sensor node, such as type of node, running operation system and applications, and secondly the state of the sensor node, such as battery, connectivity, memory. With MARWIS monitoring of a wireless sensor network can be done in two different ways. Since all information about the network is stored in the databases on the mesh nodes the user can explore this information requesting from the mesh node,

serving the corresponding subordinate sensor network. Alternatively, the user can request a selected sensor node directly.

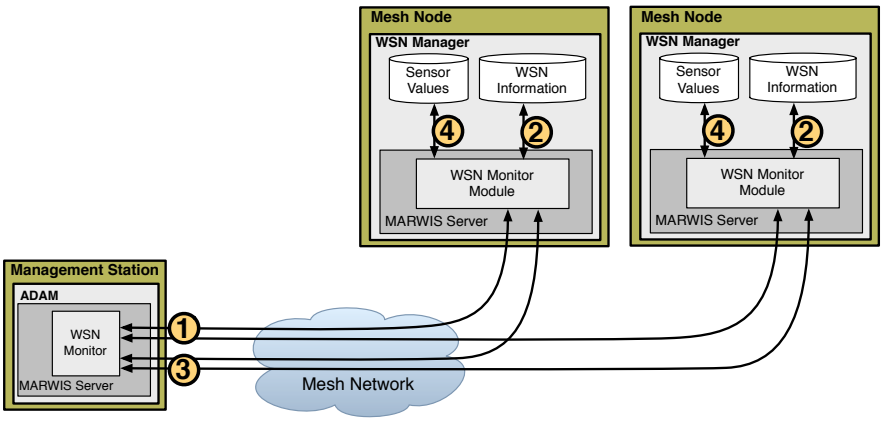


Figure 6.3: WSN monitor queries the mesh nodes

As shown in Figure 6.3), requesting information via the mesh nodes works as follows:

1. The management station queries all mesh nodes about their subordinate sensor nodes.

2. To answer the request, the WSN monitor module queries the database WSN information database and sends the requested data back.

3. Afterwards the management station queries current sensor data of the desired sensor node by sending the request to the according mesh nodes via HTTPS.

4. To answer the request, the WSN monitor module queries the sensor data database and sends the requested data back.

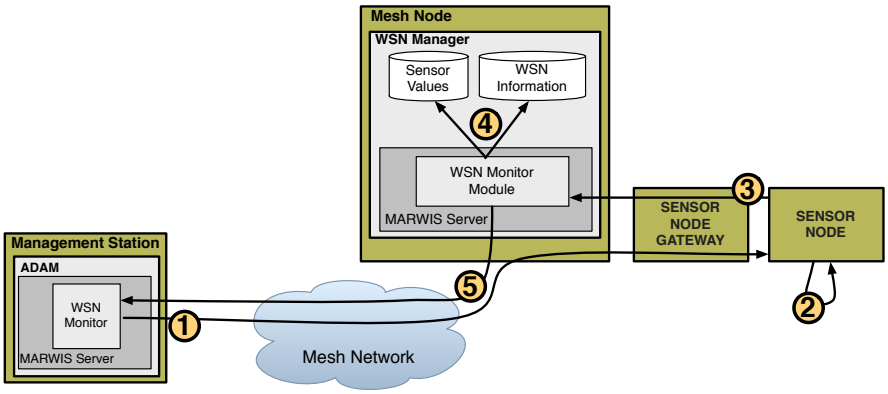


Figure 6.4: User requests sensor node information directly.

In the second case, the user wants to request information of a sensor node directly and not query the WSN information database on the mesh node. This is shown in Figure 6.4. The protocol works as follows:

1. The user requests either sensor node information (e.g. neighbourhood information) or sensor data (e.g. temperature, humidity) from a single sensor node or from a group of sensor nodes. The request created by the WSN monitor is forwarded to the affected sensor node gateway. The sensor node gateway sends this request via a unicast or multicast transport protocol, e.g., SNOMC, to the queried sensor nodes.

2. The sensor node generates the response message (measures data or reads the internal information).

3. The sensor node sends the requested information back via UDP or TCP to the mesh node, forwarded by the sensor node gateway.

4. The WSN manager module writes the new information into the according database (WSN information database and/or sensor value database).

5. The WSN manager module responds to the request and sends the information to the management station, where it is displayed to the user.

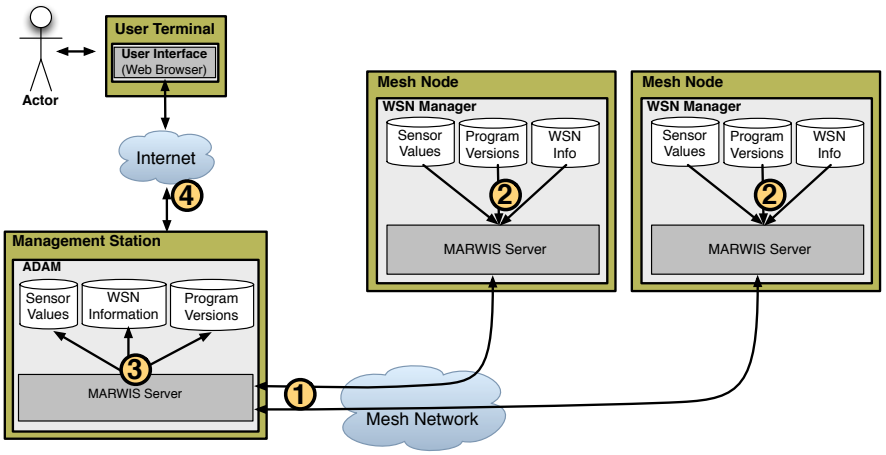


Figure 6.5: Management station requests database information from the mesh nodes.

Since the information about the wireless sensor network topology is stored in the databases on the mesh nodes it can be several minutes old and can be therefore obsolete. For example, there could be a newer and still unknown sensor node, subordinate to another mesh node. Therefore, the management station requests the information of all mesh nodes stored in their databases (sensor value database, WSN information database, and program version database) and merge it into their own databases accordingly. This is shown in in Figure 6.5. The protocol works as follows:

1. The management station asks via HTTPS all mesh nodes to transmit information about, e.g., topology, node state, etc from the databases for a certain period.

2. This information is taken from the databases and transmitted to the management station using TCP/IP.

3. The values from the different mesh nodes are merged, stored and backuped in the according databases at the management station.

4. The user gets notified about this process and the timeliness of the information.

6.4.2 WSN Configuration Protocol

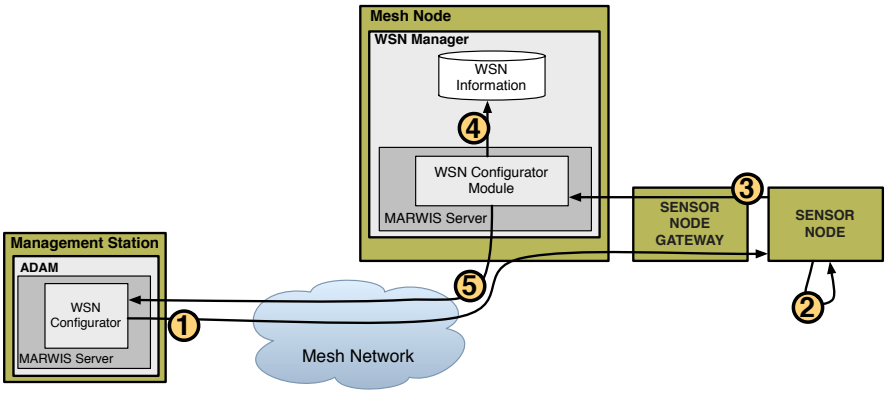


Figure 6.6: The WSN configuring protocol.

The task of the WSN configuration protocol is to perform a configuration at one or more sensor nodes. Possible configuration scenarios are: turn the sensors on/off, change sensing cycles (e.g. a measurement every second or minute), changing routing tables, configuring communication protocols, etc. The WSN configuration protocol is shown in Figure 6.6. The protocol works as follows:

1. The user selects a configuration attribute and one single sensor node (or a group of sensor nodes), which should be configured. The WSN configurator creates a request, which is forwarded to the involved sensor node gateway. The sensor node gateway sends this request via a unicast or multicast transport protocol to the queried sensor node(s).

2. On the sensor node(s) the Sensor Node Configurator performs the configuration task and generates a notification (with the new configuration of the property).

3. This notification is forwarded via the sensor node gateway to the mesh node using UDP or TCP.

4. The mesh node stores the new value of the configuration property in the WSN information database.

5. On the mesh node the WSN configurator module informs the user about the success or failure of the configuration request.

To verify the new configuration the management station queries periodically the WSN information database via HTTPS.

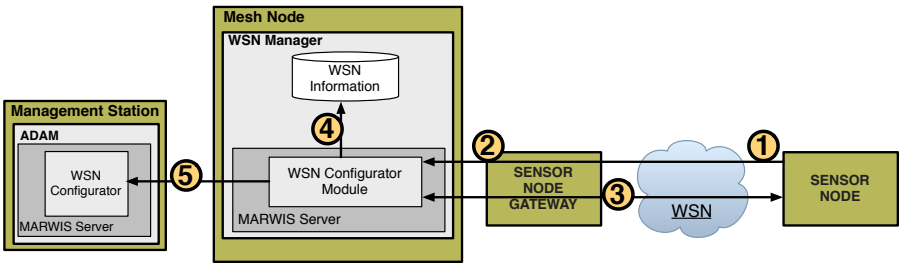


Figure 6.7: A new sensor node joins the sensor sub-network.

If a new sensor node joins a sensor sub-network it has to be registered into MARWIS to be displayed to the user. Figure 6.7 shows how a new sensor node joins the sensor sub-network. The protocol works as follows:

1. When a new sensor node joins the sensor network, it broadcasts a `Hello` message.

2. The `Hello` message is forwarded by the sensor nodes of the sub-network and the sensor node gateway to the WSN configurator module.

3. The WSN configurator module and the sensor node negotiate the necessary network configuration. Afterwards, the WSN configurator module requests all available information (e.g., chip, transceiver, battery, operating system, neighbours, etc.) from the sensor node.

4. The WSN configurator module registers the sensor node in the WSN information database.

5. The information about the joint sensor node is also propagated to the management station.

6.4.3 WSN Code Update Protocol

The WSN code updating protocol is responsible for three tasks: uploading the new image and distributing it within the mesh network, notifying the management station about the available programs and finally performing the update. The first task, the uploading process, is shown in Figure 6.8. The protocol operates as follows:

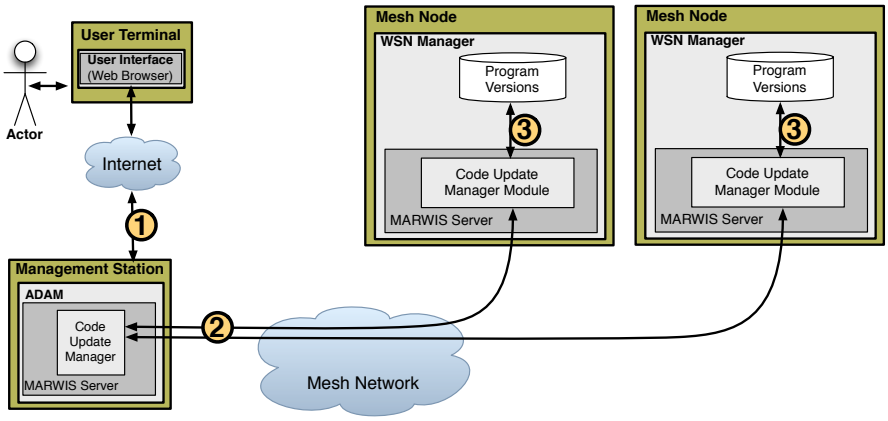


Figure 6.8: The image gets uploaded to all affected mesh nodes.

1. The user selects the new image and the according information (version, components, etc), which should be uploaded, using the user interface.

2. The image is transmitted to the affected mesh nodes managing the sensor nodes using SNOMC.

3. The code update manager module in the MARWIS Server stores the new image plus the associated information to the program version database.

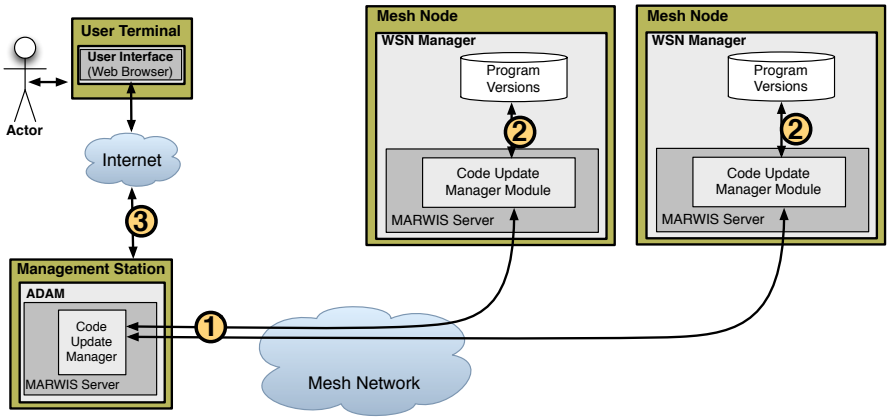


Figure 6.9: The user asks for all available images.

The second task is the notification of the management station about which programs are available in the program version database. This is shown in Figure 6.9. The protocol works as follows:

1. All available program versions are listed in the user interface. To update this list the code update manager queries via HTTPS the MARWIS server of all mesh nodes in the mesh network.

2. The code update manager module in the MARWIS server queries the program version database for the available programs/applications.

3. Afterwards the user gets notified about the available programs/applications.

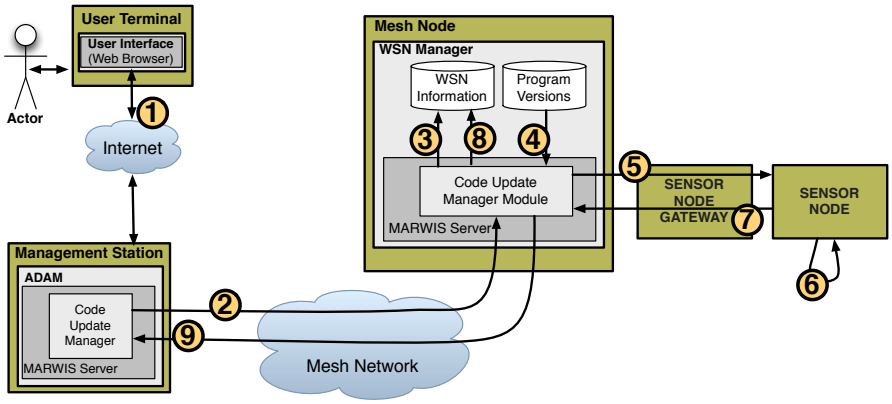


Figure 6.10: The user initiates the code update for the sensor node.

The main task of the protocol is updating the image on the sensor nodes itself. This is shown in Figure 6.10. The procedure works as follows:

1. The user selects the new program version and the sensor nodes, which should be updated, in the user interface.

2. The code update manager sends this request via HTTPS to the corresponding mesh nodes.

3. The code manager module on the mesh node checks, which program version is installed on the selected sensor nodes by quering the WSN information database.

4. The code update manager module takes the image of the new version of the selected program.

5. The new image is transmitted via the sensor node gateway to the targeted sensor nodes, using an adequate unicast or multicast transport protocol.

6. On the sensor node the update is installed by copying the image in the memory and starting it.

7. If the update is successful the sensor node verifies this by sending the new program version via the sensor node gateway to the mesh node.

8. The code update manager module stores this new information in the WSN information database.

9. The management station gets informed whether the update was successful.

6.5 Conclusions

This chapter introduced the MARWIS architecture, which is specifically designed for the management for heterogeneous wireless sensor networks in reliable, time- and energy-efficient manner. A distinguished feature of MARWIS is the use of a wireless mesh backbone. The advantages of this approach are enabling diverse communication platforms and offloading functionality from the sensor nodes to the mesh nodes and so decreasing their computational requirements.

In particular, the focus of the chapter was set on discussing design decisions as well as describing the architecture components and the underlying management protocols. In a nutshell the main components of the architecture are a management station and a management (mesh) node, which enable the interaction between end users (via a user interface) and sensor node(s). The smooth co-operation of these components relies on protocols for the monitoring, configuration and code updates of the sensor nodes.

The following Chapter 7 presents the MARWIS implementation, a MARWIS demonstrator, and related implementation issues.

Chapter 7

Implementation of MARWIS and Demonstrator

In this chapter we present the implementation of MARWIS (Management Architecture for Wireless Sensor Networks), which was described in the previous chapter. In Section 7.1 addresses the challenges of the implementation. Sections 7.2, 7.3, and 7.4 describe the implementation details of MARWIS, including the MARWIS Demonstrator described in Section 7.5. Finally, Section 7.6 concludes the chapter.

7.1 Introduction

After we described the design and architecture of MARWIS in the previous chapter 6 this chapter addresses two goals: the implementation and the quantitative and qualitative evaluation of MARWIS.

First, we describe in detail how the different MARWIS components were implemented. The implemented MARWIS components include (i) the MARWIS server running on the mesh nodes; (ii) a management station that is also running on a mesh node but includes beside the MARWIS Server also a web server running a based Graphical User Interface; and (iii) a sensor node agent, running on the sensor nodes with the Contiki OS.

Implementing MARWIS on a real system gave a demonstrator platform, which enables us to perform several tests. The purpose of these tests was to establish whether the proposed architecture is operational and to provide a qualitative and quantitative evaluation. In terms of evaluation we were interested to show that MARWIS enables reliable and time-efficient communication combined with energy-efficient operation in a network of heterogeneous wireless sensor node.

7.2 MARWIS Server Implementation

The MARWIS server located on the mesh nodes consists of three management modules (WSN Monitor Module, WSN Configurator Module, and Code Update

Module), one module for each management tasks; three databases (WSN information database, program version database, sensor value database), and a graphical user interface to depict all information about the WSN and perform the management tasks.

7.2.1 Management Modules

The modules handle the management tasks as well as the communication between the mesh nodes and the sensor nodes. The modules and the communication interface are implemented as a server program written in C using UDP/TCP sockets.

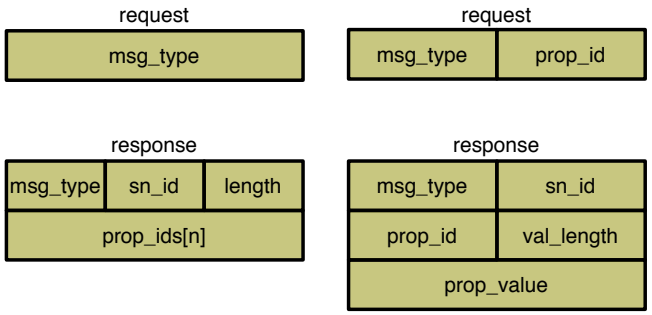
The **WSN Monitor Module** has several tasks. It stores data coming from the sensor nodes into the WSN information database and the sensor value database, respectively. Furthermore, it answers requests from the management station by retrieving data from the two databases. To retrieve data from the sensor nodes a message with the request is generated and transmitted to the according sensor node via the sensor node gateway (cf. Section 6.4.1). This functionality is implemented as follows:

- `getAllSensors(sn_id[])`: This function generates a message, which includes a request for all available sensors (prop_id) and sends it to the given sensor nodes (sn_id[]). Each sensor node determines all installed sensors and generates a message back to the mesh node.
- `getAllProps(sn_id[])`: This function generates a message, which includes a request for all available properties (prop_id) and sends it to the given sensor nodes (sn_id[]). Each sensor node determines the available properties and generates a message back to the mesh node.
- `getSensorValue(sn_id[], prop_id[])`: This function generates a message, which includes a request for a sensor value of a given sensor (prop_id) and sends it to the given sensor nodes (sn_id[]). Each sensor node determines the requested values and generates a message reporting these values to the mesh node.
- `getPropValue(sn_id[], prop_id[])`: This function generates a message, which includes a request for a property value of a given sensor (prop_id) and sends it to the given sensor nodes (sn_id[]). Each sensor node determines the value of the requested property and generates a message sending this value back to the mesh node.
- `getNeighbors(sn_id[])`: This function generates a message, which includes a request for the neighbours of the sensor node and sends it to the given sensor nodes (sn_id[]). Each sensor node determines its neighbours and generates a message sending this information back to the mesh node.
- `getHopDistance(sn_id[])`: This function generates a message, which includes a request for the hop distance from the sensor node to the gateway

and sends it to the given sensor nodes (sn_id[]). Each sensor node determines this information and generates a message back to the mesh node.

- `confProp(sn_id[], prop_id, conf_cmd)`: This function generates a message, which includes the configuration command (conf_cmd) to configure a given property (prop_id) and sends it to the given sensor nodes (sn_id[]).
- `uploadImg(sn_id[], prop_id, img, update_cmd)`: This function generates a message, which includes the updated image (img) and the update command (update_cmd) and sends it to the given sensor nodes (sn_id[]). To identify the application for later configuration it also gets a property id.

All requests are sent over TCP/IP and µIP respectively to the sensor nodes. As transport protocols UDP or SNOMC are used.



(a) getAllProperties (b) getPropValue

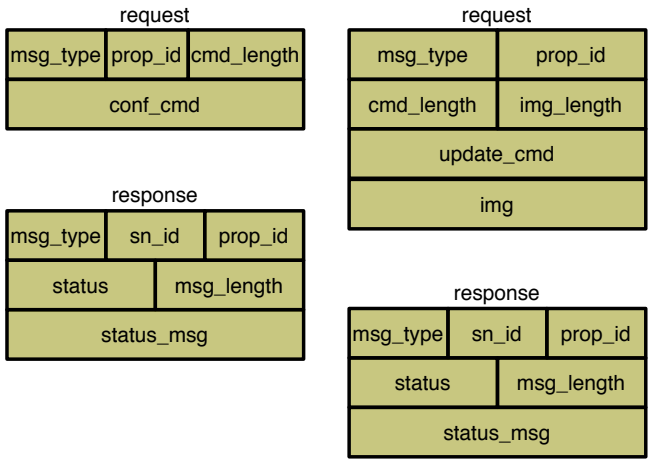
Figure 7.1: MARWIS server messages for retirieving information from the sensor nodes.

The request and response messages for the *getAllSensors()* and *getAllProps()* functions have the same format and are shown in Figure 7.1(a). The request message contains just one field named `msg_type`, sized one byte. The `msg_type` for requesting all sensors or all properties is *REQUEST_ALL_SENSORS* and *REQUEST_ALL_PROPS* respectively. The response message has four fields. In the `msg_type` field, again one byte long, the message type is indicated with *RESPONSE_ALL_SENSORS* or *RESPONSE_ALL_PROPS*. In the `sn_id` field the identifier of the responding sensor node is indicated. In our case the Contiki RIME address is used for that. Therefore, this field has a size of two bytes. The `length` field, itself one byte long, indicates the length of the `prop_ids` field and so the number of all responded properties. Each `prop_id` is one byte long.

For the functions *getSensorValue()*, *getPropValue()*, *getNeighbors()*, and *getHopDistance()* the same format of request and response message is used as shown in Figure 7.1(b). The request message contains again the one byte field `msg_type` with the value *REQUEST_PROP_VALUE* and a one byte field called `prop_id`, which indicates the requested property. The response message contains five fields. The `msg_type` field has the value *RESPONSE_PROP_VALUE*, the `sn_id` field

contains the RIME address of the sensor node (two bytes), and the `prop_id` field contains the identifier of the requested property (one byte). The value in the field `val_length` indicates the length of the requested property. The range can be from one byte (e.g. current battery value or sensor value) up to many bytes (e.g. routing table). In case the requested property does not exist, the length is zero and the `prop_value` field is empty.

The **WSN Configurator Module** is responsible for all configuration tasks. It connects to the WSN information database to read and write data. Furthermore, it creates packets for the sensor nodes to query sensor node properties and send commands.



(a) confProp (b) updateImage

Figure 7.2: MARWIS server messages for configuration and code update.

The configuration messages have different formats and are shown in Figure 7.2(a). The `msg_type` is *REQUEST_CONF_PROP* and the `prop_id` is the corresponding property, which should be configured. The field `cmd_length` indicates the length of the configuration command `conf_cmd`, which is a string. The response message contains the field `msg_type` (*RESPONSE_CONF_PROP*), `sn_id` (RIME address of the requested sensor node), and `prop_id` (identifier of the configured property). Moreover, it contains a one byte field `status`, which indicates the status of the configurations. Possible values are *SUCCESS*, *IN_PROCESS*, or *FAILED*. Moreover, a status message can be sent (field `status_msg`) as a string. The field `msg_length` indicates the length of the message string.

The **Code Update Manager Module** is responsible for storing newly uploaded code (and related information) in the program version database. Furthermore, the code manager module answers to requests from the management station about available programs by sending a complete list of all program versions stored in the program version database. It also executes updating of the sensor nodes by transmitting the image to the selected sensor nodes.

The message format for updating an application is shown in Figure 7.2(b). The request message has again the fields `msg_type` (*REQUEST_UPDATE_IMAGE*) and `prop_id`. It contains further two fields indicating the length of the update command string (`cmd_length`) and the length of the image (`img_length`), both one byte long. It contains a string-field with the update command (`update_cmd`) as well as a field, which includes the binary data for the image (`img`). The response message looks similar to the response message for the configuration command, with the only difference that the `msg_type` is *RESPONSE_UPDATE_IMAGE*.

7.2.2 Database Implementation

The three databases (WSN information database, program version database, and sensor value database) are managed with SQLite3 [110]. SQLite3 is a file-based database management system contained in a small C programming library (approximately 350KB).

sn_id	sn_name	sn_ip,	hw_id	os_id
INT	VARCHAR(20)	VARCHAR(20)	INT	INT
pos_x	pos_y	pos_z	gw	mn_id
INT	INT	INT	BOOL	INT

Table 7.1: Database table `sensornodes` for the sensor nodes.

The **WSN information database** stores data about the sensor nodes and the WSN. This database contains tables. The basic information about the sensor nodes are stored in the table `sensornodes`, which is depicted in Table 7.1. It contains the id, the name, and the IP address of the sensor node, the identifiers of the platform and the running operating system, the 3D position of the node, the information if the node is a sensor node gateway or not, and the identifier of the corresponding mesh node. The primary key is the column sn_id, indicated in bold.

mn_id	mn_type	mn_ip,	pos_x	pos_y	pos_z
INT	VARCHAR(20)	VARCHAR(20)	INT	INT	INT

Table 7.2: Database table `meshnodes` for the mesh nodes.

For each mesh node a corresponding table exists. This table is called `meshnodes` and is depicted in Table 7.2. It contains the id, IP address, type, and 3D position of the mesh node. The column mn_id is the primary key of this table.

mn_id	**mn_neighbor_id**	time
INT	**INT**	TIMESTAMP

Table 7.3: Database table `wmn` for the mesh network.

The links between the sensor nodes and mesh nodes are stored into two tables: `wsn` (shown in Table 7.4) and `wmn` (shown in Table 7.3), respectively. With this information the networks (WSN and WMN) can be shown. The tables contain

sn_id	**sn_neighbor_id**	etx	time
INT	**INT**	INT	TIMESTAMP

Table 7.4: Database table `wsn` for the sensor network.

the identifier of the sensor node (or mesh node) and the identifier of the neighbor node. Due to the dynamic characteristic of a link we also store the timestamp of the link. If a link disappears the according row in the table will be deleted. In case of the sensor network there is an additional column storing the link quality. The primary key of both tables is the pair sn_id, sn_neighbor_id or mn_id, mn_neighbor_id, repectively.

sn_id	**prop_id**	prop_value	time
INT	**INT**	VARCHAR100	TIMESTAMP

Table 7.5: Database table `sn_properties` storing the values of the sensor node properties.

The values of all properties of a sensor node are stored in a table called `sn_properties`. This table contains the identifier of the sensor node and the identifier of the property as well as the value and the time of the property. The table format is depicted in Table 7.5. Since all properties can have only one value at a time the primary key includes the columns sn_id and prop_id.

prop_id	prop_desc	prop_type
INT	TEXT	VARCHAR20

Table 7.6: Database table `properties` storing descriptive information about the properties.

An additional table called `properties` stores descriptive information about the properties of a sensor node. These properties can be static ones such as the micro-controller, the radio transceiver, flash memory, or dynamic ones such as the sensors, the LEDs and the battery. The table is shown in Table 7.6. It contains three columns: the identifier and description of the property, as well as the data type (which can be string, int or float). This is necessary because in the `sn_properties` table all property values are stored as a string and the data type information is used to convert the string into a correct data type. The primary key is the property id.

sn_id	**prop_id**	state	interval
INT	**INT**	INT	INT

Table 7.7: Database table `sensor_state` storing the state of the sensors.

A further table called `sensor_state` stores the current state of the sensors of a sensor node. This includes the switch state (on/off) of the sensor as well as the sensing interval. The identifier of the sensor node and the identifier of the property are the primary key.

snpf_id	snpf_desc	pic_file_name
INT	TEXT	VARCHAR(50)

Table 7.8: Database table `sn_platforms` for the sensor node platforms.

Several supporting tables store the information about the sensor node platforms, the mesh node platforms, and the operating system platforms. The table for the sensor node platform (in fact the sensor node type) is called `sn_platforms` and is depicted in Table 7.8. In contains the identifier and the description of the sensor node platform. To depict the sensor nodes on the Web-GUI we use small pictures of the nodes. Therefore, the table contains the file names of the picture file. The primary key of the table is the columns snpf_id.

mnpf_id	mnpf_desc	pic_file_name
INT	TEXT	VARCHAR(50)

Table 7.9: Database table `mn_platform` for the mesh node platforms.

ospf_id	ospf_desc
INT	TEXT

Table 7.10: Database table `os_platform` for the operating system platforms.

The tables for the mesh node platform (`mn_platform`, shown in Table 7.9) and the operating system platform (`os_platform`, shown in Table 7.10) are similar to the one for the sensor node platform. The columns mnpf_id and ospf_id are the primary keys of the both tables.

pic_file_name	pic_desc	pic_file_size	file
VARCHAR(50)	TEXT	INT	BLOB

Table 7.11: Database table `pics` storing the pictures used by the GUI.

Furthermore, there is a table `pics` storing the information of the pictures used by the GUI. It contains the file name of the picture, the description the file size and the binary data of the picture and is depicted in Table 7.11. The primary key of this table is the column pic_file_name.

To access the WSN information database the following functions are implemented:

- `getAllMeshNodes()` ← `mn_id[]`: This function returns a list of all available mesh nodes (mn_id). The table `meshnodes` is queried.
- `getAllSensorNodes()` ← `(sn_id, mn_id)[]`: This function returns a list of all available sensor nodes (sn_id) including the responsible mesh node (mn_id). In this case the table `sensornodes` is queried.
- `getAllOwnSensorNodes(mn_id)` ← `sn_id[]`: This function returns

a list of all accessible sensor nodes (sn_id[]) from a given mesh node (mn_id). Again the information is gathered from the table `sensornodes`.

- `getSensorNodeProperties(sn_id)` ← `prop_id[]`: This function returns a list of all available properties (prop_id[]) of a given sensor node (sn_id) by querying the table `sn_properties`.
- `getPropertyValue(sn_id, prop_id)` ← `value`: This function returns the value of a given property (prop_id) of a given sensor node (sn_id) gathering the information from the tables `sn_properties` and `sensor_state`.
- `insertPropertyValue(sn_id, prop_id)`: This function inserts (or updates) a value of a given property (prop_id) of a given sensor node (sn_id) into the tables `sn_properties` and `sensor_state`.
- `insertProperty(sn_id, prop_id, [value])`: This function inserts a new property (prop_id), [optional with the value of the property], of a given sensor node (sn_id) into the database. In this case the table `properties` is affected as well.
- `insertNewSensorNode(sn_id)`: this function inserts a new sensor node (sn_id) into the database. In this case the tables `sensornodes`, `wsn`, `sn_properties` and `sensor_state` have to be updated.

sn_id	**img_id**	img_state	proc_slot	update_time
INT	**INT**	INT	INT	TIMESTAMP

Table 7.12: Database table `sn_images` storing which images running on which sensor nodes.

The **program version database** stores all versions of all programs, which can be installed on the sensor nodes. This database contains two tables: `sn_images` (shown in Table 7.12) and `images` (shown in Table 7.13). The first table contains information about which image of the applications and the operating system is running on which sensor node. It contains the following columns: identifiers of the sensor node and image. Furthermore, it contains the state of the image (running, stopped) and the process slot on the sensor node in which the application runs. Finally, it contains the time of the update. The primary key is the identifiers of the sensor node and the image.

img_id	img_name	img_version	img_type	img_desc
INT	VARCHAR(100)	VARCHAR(10)	INT	TEXT
ospf_id	img_file_name	img_file_size	img_file_time	img_file
INT	VARCHAR(50)	INT	TIMESTAMP	BLOB

Table 7.13: Database table `images` storing the images.

The second table images contains the images of all applications for all platforms. It contains columns for the identifiers of the sensor node, the image and the supported operating system (see Table 7.10), name, version, type (application or whole operating system image), and description of the image. It also contains the file of the image, including the file name, file size and file timestamp. The image identifier is the primary key of this table. To access the databases the following functions are implemented:

- `getAllProgVersions() ← [(img_id, img_name, img_version, img_type, img_desc)]`: This function returns a list of all available programs (img_id) including their name, version, type and description. The table images is querried.
- `insertNewProgVersion(img_name, img_version, img_type, img_desc, ospf_id, img_file)`: This function inserts a new program into the database, including all related information such as name, version, type, description, and supported operating system. The table images is updated.
- `getImage(img_id) ← img_file`: This function is the counterpart of the previous function. It returns the image of a given program (img_id). In this case the table images is queried as well.
- `getProgsOfNode(sn_id) ← [(img_id, img_name, img_version, img_type, img_desc, img_state, proc_slot, update_time)]`: This function returns a list of all programs on a selected sensor node. This includes the running applications as well as the stopped applications, which are still stored in the memory of the sensor node. This query affects the tables sn_images and images and contains all related information about the image.

sn_id	**prop_id**	value	time
INT	**INT**	**INT**	**TIMESTAMP**

Table 7.14: Database table sn_values storing the measured sensor values.

The **sensor value database** stores all data measured by the sensors. It contains only one table, which is shown in Table 7.14. The table called sn_values contains the identifiers of the sensor nodes, the property and the measured value of the time of the measurement. The primary key consists of all four columns. The difference to the table sn_properties is that in this table all values of the past are stored and not only the most recent value. To access the database the following functions are implemented:

- `getSensorValues(sn_id, prop_id, time_from, time_to) ← value[]`: This function returns a list of all sensor values from a given

sensor (prop_id) on a given sensor node (sn_id) within a time interval (time_from, time_to).

- `getCurrSensorValue(sn_id, prop_id) ← (value, time)`: This function gives the most recent value of a given sensor (prop_id) on a given sensor node.
- `getAllSensorValues() ← [(sn_id, prop_id, value, time)]`: This function dumps the database and gives a list of all values (including the time stamp) of all sensors on all sensor nodes.
- `insertSensorValue(sn_id, prop_id, [(value, time)])`: This function inserts a value (or a list of values) from a given sensor (prop_id) on a given sensor node (sn_id) into the database.

The management station also contains the three databases to additionally store and backup all data from all mesh nodes' databases. Thus, the management station queries all databases from the mesh nodes to retrieve their data. For this, the following functions are implemented:

- `newDataAvailable(time, database) ← mn_id[]`: This function creates a request to all databases of the mesh nodes, if new data is available in a given database and for a given time. It returns a list of the mesh nodes, which have new data in the given database.
- `getData(time, mn_id, database) ← data`: This function creates a request to query the selected database for the given time at all indicated mesh nodes.

7.3 Management Station with Graphical User Interface

As front-end of MARWIS a web-based graphical user interface was implemented. With this GUI the user can perform major management tasks, such as monitoring, configuration, and code update.

A start-up web site, shown in Figure 7.3, visualizes the topology of the networks as a map and the information about the sensor nodes. In the example case in Figure 7.3 the map depicts the two upper floors in the building of the Institut für Informatik und angewandte Mathematik. The mesh nodes are depicted with grey circles and the sensor nodes from type tmote sky are depicted with green circles. The dotted lines represent the links between the nodes. The information about the mesh and sensor nodes include the IP address, information about the platform and the operating system, information about the hardware (micro-controller, radio, etc.), information about the sensors, the LEDs and the status of the battery (current voltage). All this data is shown on the right side of the map. By clicking on the hosting mesh node (e.g. *marwismn01*), the connected sensor sub-network is shown.

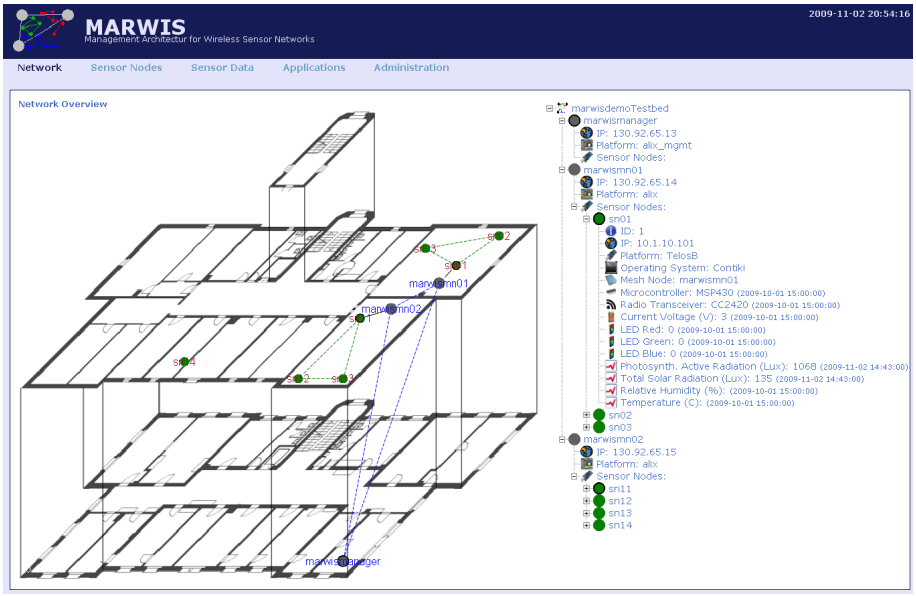


Figure 7.3: User interface: network overview

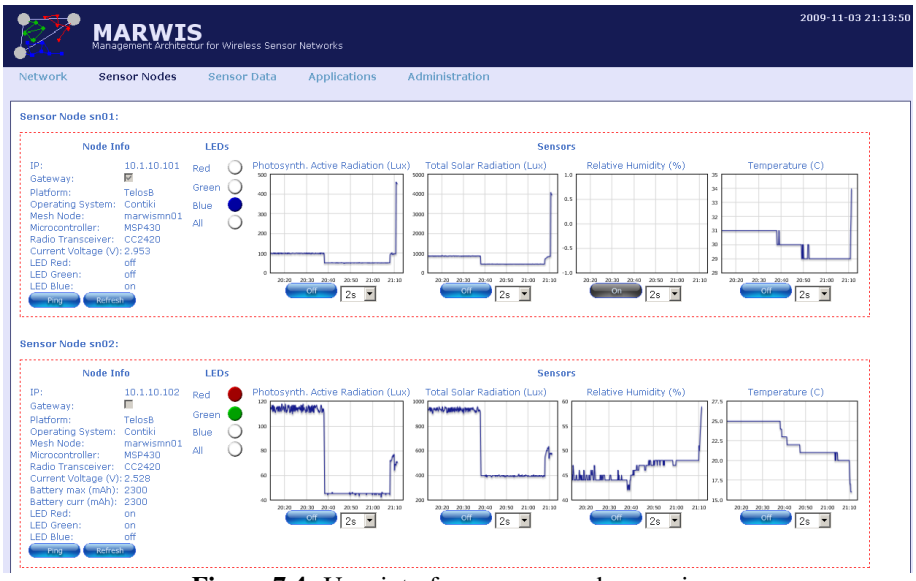


Figure 7.4: User interface: sensor node overview.

By clicking on a sensor node (e.g, *sn01*), the information about the sensor node and the operating system is depicted as shown in Figure 7.4. On this page all sensor nodes of the sensor sub-network are listed. For each sensor node basic information (*Node Info*) such as platform, operating system, hosting mesh node, hardware information, and current voltage (monitoring task) is presented. This information can be seen on the left side of Figure 7.4. The user can ping the sensor node to find out, if the node is still alive (monitoring task). Under *LEDs*

the user can find the state of the LEDs. By selecting them the user can switch the LEDs on or off (configuration task). The measured values of the available sensors are depicted in the diagrams. In the example case we have two light sensors (photosynthetic active radiation, total solar radiation), one for relative humidity and one for temperature. The user can switch on/off the sensors and decide the sensing interval (configuration task). All information is taken from the databases from the mesh nodes (see Figure 6.3 in Section 6.4.1). With *Refresh* the user can refresh the information by querying the sensor nodes directly (see Figure 6.4).

Another management task is the code update. The user uploads first the image to the database. The user has to give the application a name, a version, a description and afterwards select the code file (see Figure 6.8 in Section 6.4.3). After the image is uploaded to the mesh node, the user can select the image of the application and the sensor node, which should be updated and start the updating process (see Figure 6.9). It is possible to start, stop and delete an application on the sensor node.

The web-based graphical user interface runs on the management station, which has an Apache web server. It is written in php version 5.4.6 [84] using packages for Sqlite3. The graphics in the map have been created by Graphviz 2.12 [50] using the neighbourhood information from the database. The tree has been implemented using dhtmlxtree version 1.6 [5]. The diagrams are made with Flot version 0.7 [44].

7.4 Sensor Node Agent Implementation

In this section the implementation details of the SN Agent on the sensor node are discussed. This includes addressing of the sensor nodes and the three management tasks on the node (monitoring, configuration, and code updates).

As described in Section 2.2.2 we are using Contiki 2.5 as operating system. Contiki has some advantage to support most of the sensor node platforms we are using, such as Tmote SKY / TelosB, MicaZ and MSB. In case of the BTNode platform we implemented a port Contiki [103]. Contiki supports protothreads, which allows code updates of several applications running on the node. Furthermore, Contiki supports TCP/IP (µIP) and RIME that eases implementation of the management protocols (see Section 6.4).

7.4.1 Addressing

The first task is the **addressing** of the sensor nodes. Each node has an IP address. The first three bytes identify the sensor sub-network (*10.1.10*); the last byte identifies the sensor nodes itself (*101*). We are using the Contiki RIME address, which has two bytes and corresponds to the identifier of the sensor node. A node identifier of 72 corresponds to a RIME address of 72.0. To create a RIME address of 10.101 the node identifier has to be set to 2661. This example is shown in Figure 7.5. The routing tables on the hosting mesh nodes look accordingly.

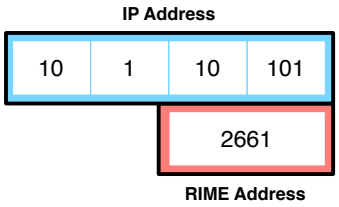


Figure 7.5: SN Agent: addressing.

7.4.2 Sensor Node Monitor

The **Sensor Node Monitor** is responsible for handling monitor requests. First, when a sensor node switches on, the values of all available properties are collected and stored in an array. The array consists of the property id, the information flag to indicate if the property is static or dynamic, the value of the property and a time stamp. Possible static properties are node platform, node id, node IP address, node role (gateway or not), micro-controller, radio transceiver, operating system, or memory space. Dynamic properties are for instance the value of the sensor (light, humidity, temperature), current battery value, LED status, or free memory space. When the Sensor Node Monitor receives a monitor request, it checks if the property is static or dynamic. In case of a static property it takes the value from the array. In case of a dynamic property it determines the value by, e.g., requesting the sensor directly. Afterwards it generates a packet to transmit the requested value back to the mesh node. The following functions implement the functionality:

- `InitSensorNodeMonitor()`: When a sensor node starts, this function determines all available (static) properties and their values. These values are stored in the array described above.

- `ReceiveMonRequest(prop_id[], sender_ip)`: When a request for one or more sensor node properties reaches the monitor, this function is called. The parameter is an array of the requested properties (prop_id[]) and the IP address of the sender. This function determines the requested values (by calling GetPropertyValue()) and sends it back to the sender (sender_ip) by calling SendSensorValue()).

- `GetPropertyValue(prop_id[])` $\leftarrow$ `value[]`: This function reads the values of all requested properties (prop_id[]). If a static property is requested its returned value is read from the array. In case of a dynamic property it calls the driver functions of the, e.g., sensors, to retrieve the dynamic values of the requested properties. In the latter case the value is also stored into the property array including the according time stamp.

- `SendSensorValue(prop_id[], value[], sender_ip)`: This function creates one or more packets with property values and transmit them to the sender.

7.4.3 Sensor Node Configurator

The **Sensor Node Configurator** executes the configuration commands on the sensor node. The configuration task is strongly depending on the application, which has to be configured. It can be permanent or non-permanent. In the permanent case the variable is written into the flash memory. Thus, this variable is still available after rebooting the sensor node. An application reads this variable and changes its behaviour during run-time accordingly. In the non-permanent case only a global variable located in the RAM is written. The application reads this global variable and changes its behaviour accordingly. After rebooting the value of the global variable has disappeared. The Sensor Node Configurator receives a message containing a configuration command. The configuration command contains the name and the value of the global variable, which has to be set and an information if the variable is located in the RAM or in the flash memory. In the latter case the address of the variable has to be indicated. After writing the global variable in the RAM or the variable in the flash memory the notification message is created and transmitted to the mesh node. The following functions implement the functionality of the Sensor Node Configurator:

- `ReceiveConfRequest(conf_cmd, sender_ip)`: This function is called when a configuration request has been received. It starts the configuration process by calling Configure(). The IP address of the sender (sender_ip) is recognized to send the notification message later.
- `Configure(conf_cmd)`: This function performs the configuration by executing the configuration command (conf_cmd). As described above executing the configuration command means to write a value to a global variable in the RAM or to a variable in the flash memory, respectively.
- `RestartSensorNode()`: If a configuration demands a restart, this function restarts the sensor node.
- `SendNotification(message, sender_ip)`: This function sends a notification about the success of the configuration.

7.4.4 Code Updater

The **Code Updater** is responsible for the code updating process. Contiki works with loadable modules to allow replacing applications. This allows to start/stop/load-/update application during run-time. The image of the application is divided into several packets and transmitted to the sensor node. The packets are received and the image is stored into the flash memory. Then, the referenced variables are checked and functions of the new application are linked and relocated by the Contiki dynamic Link Editor (CLE). The code updater starts the application using the Contiki function *_init()*. Finally, it sends a packet to notify the mesh node of the success of the update. The following functions implement this functionality:

- `ReceiveUpdRequest(img*, sender_ip)`: After receiving a packet with a part of the image, this function adds the image data on top of the previous ones in the flash memory. After the last packet with parts of the image has arrived, the image building process is started by calling the function BuildImage(). The IP address of the sender (sender_ip) is marked to send later the notification message.
- `BuildImage(img*, upd_type)`: This function creates the application using the Contiki dynamic Link Editor (CLE).
- `RestartSensorNode()`: This function restarts the sensor node.
- `RestartApplication(app_id)`: This function starts (or restarts) the selected application (app_id).
- `SendNotification(message, receiver_ip)`: This function sends notification about the success of the update. If th update was successful, the new version number and a timestamp of the update is sent to the mesh node. If not, the message contains an error message.

7.5 MARWIS Demonstrator

To show the ability of MARWIS to support heterogeneous WSNs we implemented a MARWIS demonstrator [124]. We built a small WSN including sensor nodes from several platforms (Tmote SKY / TelosB, MSB, MICAz, and BTnodes) and a WMN as backbone with ALIX 3d2 mesh nodes. This demonstrator contains the complete MARWIS implementation with the management station with the web-based GUI running on a mesh node, the MARWIS Server running on mesh nodes and SN Agent running under Contiki OS on the several sensor node platforms. Figure 7.6 shows the scenario.

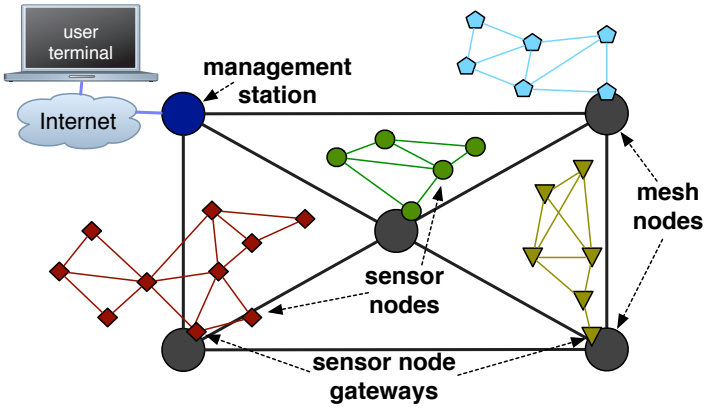


Figure 7.6: A possible scenario for heterogeneous WSNs with management devices.

We demonstrated the operation of the three management tasks (monitoring, configuration, and code update). For monitoring the current battery state and the

sensor values of the temperature, light and humidity sensors are displayed. Additionally, the sensor nodes are pinged to show if the sensor node is alive or not. To demonstrate the configuration task the sensing interval of the sensors is configured. To demonstrate the code update we implemented simple applications, which perform simple blinking sequences (e.g., blinking red, blinking loop, etc).

For the demonstration we setup a heterogeneous WSN with different types of sensor nodes, similar as shown in Figure 7.6). We used 5 Alix 3D mesh nodes, 3 TelosB sensor nodes, 3 MSB sensor nodes, 3 MicaZ sensor nodes, 3 BTnodes sensor nodes. One mesh node act as the management station, the other 4 mesh nodes handle the 4 sensor sub-networks containing the 4 different types of sensor nodes.

7.6 Conclusions

This chapter focussed on the details of the MARWIS implementation The MARWIS server with the management modules and the databases was implemented on the mesh nodes. On the management station a web-based graphical user interface was implemented to allow the user to perform the management tasks, such as monitoring, configuration, and code updates. On the sensor nodes SN Agents were implemented performing the management tasks on the sensor nodes.

Further, we implemented a MARWIS demonstrator to qualitatively evaluate MARWIS's support of heterogeneous types of sensor nodes and the function of the MARWIS management protocols.

The following Chapter 8 presents the integration of SNOMC into the MARWIS architecture and a quantitative evaluation of MARWIS.

Chapter 8

SNOMC Integration into MARWIS

This chapter presents the integration of the SNOMC protocol into MARWIS. SNOMC is the overlay multicast transport protocol for wireless sensor networks that supports reliable and time- and energy-efficient transport of bulky data presented in Chapter 3.

Section 8.1 describes the problem we address with the integration of SNOMC into MARWIS. While Section 8.2 presents the architectural aspects of the integration, Section 8.3 focuses on the implementation, including the implementation of SNOMC on wireless mesh nodes and adaptations of the web-based graphical user interface of MARWIS. Section 8.4 presents a detailed evaluation of the time- and energy-efficiency of SNOMC in combination with MARWIS. Finally, Section 8.5 concludes the chapter.

8.1 Introduction

The combination of SNOMC and MARWIS supports reliable, time-, and energy-efficient management of heterogeneous wireless sensor networks. While SNOMC provides the reliable, time-, and energy-efficient communication in heterogeneous wireless sensor networks, MARWIS provides the management functionality for monitoring, configuration and code updating. To achieve this, SNOMC has to run on each entity of the heterogeneous network, not only on sensor nodes but also on mesh nodes.

Figure 8.1 shows an example scenario using SNOMC in a heterogeneous MARWIS environment. In this scenario we have a wireless mesh network, which forms the backbone of the heterogeneous wireless sensor network. In the wireless mesh network we have a management station and a number of mesh nodes. Each of the mesh nodes handle a sensor sub-network.

The SNOMC distribution tree is marked by the orange arrows. Thicker arrows show the distribution tree within the wireless mesh network, blue arrows show the tree in the mesh network, orange arrows shows the tree in the sensor sub-networks.

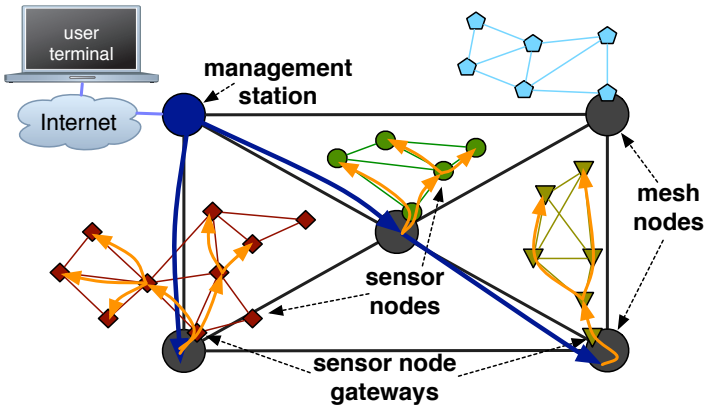


Figure 8.1: SNOMC in a heterogeneous MARWIS scenario.

8.2 Architecture

In order to integrate SNOMC into the MARWIS the architecture of MARWIS requires some changes. Another important aspect is the addressing of the mesh nodes and sensor nodes in the heterogeneous network.

Figure 8.2 shows the architectural changes on the Management Station and the mesh nodes while integrating SNOMC. Both on the Management Station and the mesh nodes a **MARWIS Communication Server** is running, which is responsible for the communication within the mesh network. In the MARWIS Communication Server the transport protocols SNOMC or UDP-E2E are integrated to support efficient and reliable data delivery in a MARWIS environment.

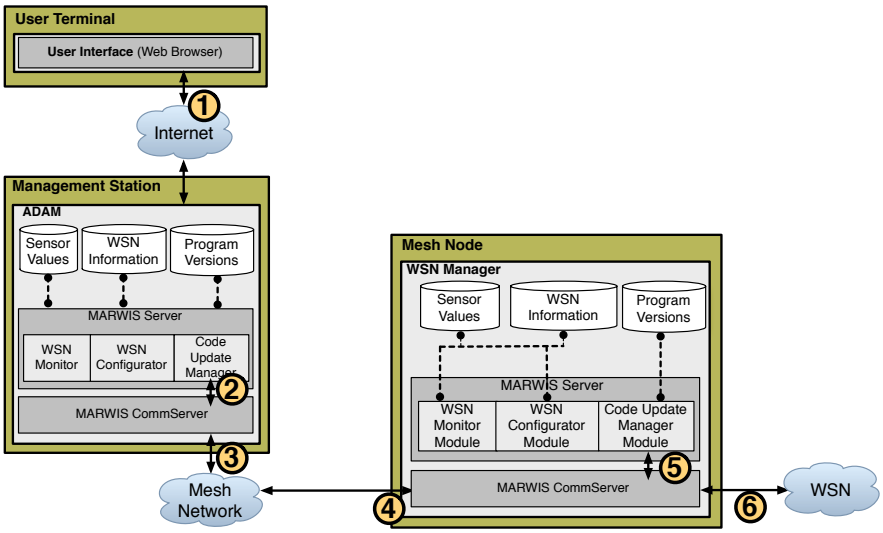


Figure 8.2: SNOMC integrated into the MARWIS architecture.

Figure 8.2 shows the code updating process of sensor nodes using SNOMC in MARWIS (including the wireless mesh network and the wireless sensor network) as shown in Figure 6.10 in Chapter 6.4.3.

1. The user selects the update, the receiver nodes, and the transport protocol for the data transmission. This information are sent to the Code Update Manager on the Management Station.

2. The Code Update Manager fragments the image and transmits it including the information about the receivers and the selected transport protocol to the MARWIS Communication Server. In case of using SNOMC the MARWIS Communication Server initiates the join procedure to build the distribution tree.

3. After a successful joining procedure of the sensor nodes, the transmission of the data fragments starts.

4. Beside on the management station, a MARWIS Communication Server is also running on the mesh nodes and listens to incoming packets.

5. The MARWIS Communication Server also caches the fragments on the mesh nodes. Caching is the responsibility of the Code Update Manager.

6. Afterwards the fragments are transmitted to the wireless sensor network via the sensor gateway node attached to the mesh node.

In a heterogeneous scenario it can also happen that the code update requires different images for different types of sensor nodes. In this case the distribution tree is built by the mesh nodes. The image for each sensor sub-network node is taken from the database of the mesh node, which manages the sub-network.

In case of transmitting commands for monitoring and configuration the distribution tree is always built by the management station. The same message is transmitted to all receivers, because even if sensor nodes are of a different type they receive the same configuration message.

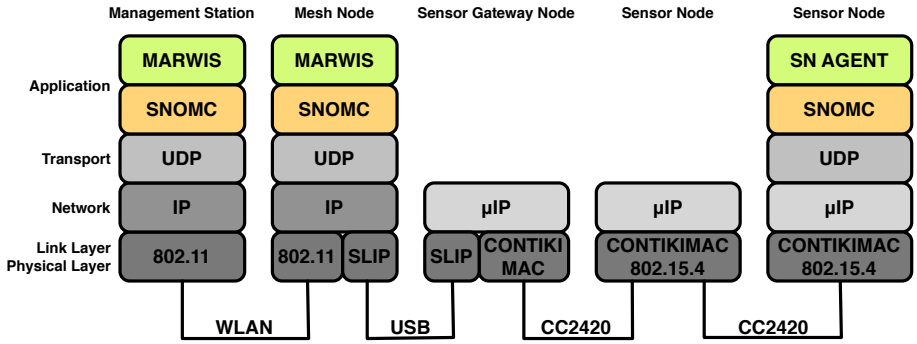


Figure 8.3: Protocol stack containing SNOMC in the heterogeneous MARWIS architecture.

The SNOMC integration into MARWIS follows the same design principles as SNOMC for wireless sensor networks. It includes the same protocol phases such as joining and data transmission, and also includes the same reliability mechanisms as described in Chapter 3. Moreover, SNOMC uses the same messages *join*, *join_ack*, *data*, *data_ack*, and *nack* as on the sensor nodes with the Contiki OS. Therefore, SNOMC supports heterogeneity by running on different types of networks, such as 802.11 or 802.15.4. The protocol stack in the different entities of the MARWIS architecture, such as management station, mesh node, sensor gateway node, and sensor nodes is shown in Figure 8.3. A packet is transmitted from the management station to one or more sensor nodes (only one sensor node as receiver is depicted in the Figure). The packet is created by the MARWIS and distributed by the SNOMC protocol. The packet is passed through the protocol stack at the management station. Management station and mesh nodes are connected using IEEE 802.11. On the mesh node the packet is directed to the sensor node gateway. Sensor node gateway and mesh node are connected with a serial line, using SLIP as protocol. From the sensor node gateway the packet is transmitted using a radio transceiver, which is IEEE 802.15. 4 compliant and ContikiMAC as MAC protocol. On top is µIP on the network layer. On the receiver node the packet is passed through the protocol stack, including µIP, UDP, SNOMC. Afterwards, the packet is delivered to the SN Agent, which is the MARWIS implementation on the sensor node.

SNOMC on the sensor nodes and SNOMC integrated into MARWIS differs not in the design principles but in the implementation, which is described in Section 8.3.

Addressing of the nodes in the MARWIS architecture plays an important role. The addressing scheme is shown in an example in Figure 8.4. All nodes have private IP addresses. The management station and the mesh nodes are in the same sub-network as in each sensor sub-network. The SLIP interface (*tuneslip*) of the mesh nodes is in the same network as the sensor sub-network and has a private IP address, e.g., *10.1.10.100*.

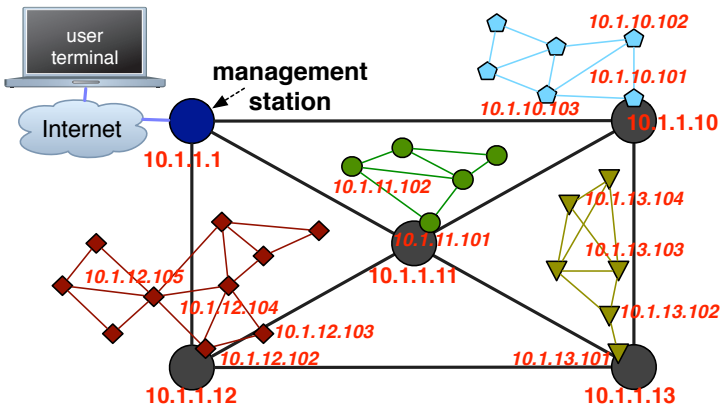


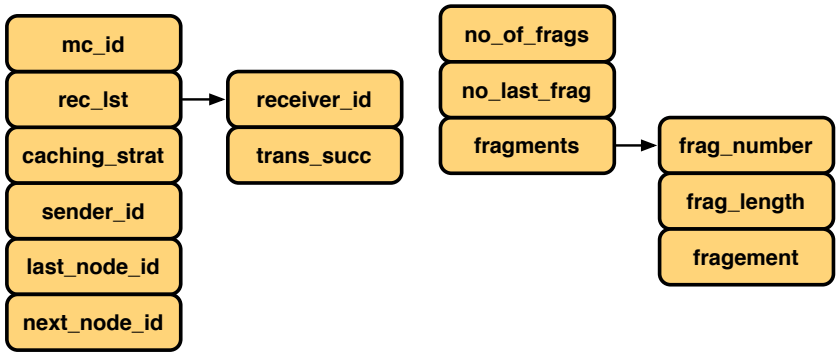
Figure 8.4: Addressing scheme of the nodes in the MARWIS architecture.

8.3 Implementation

The implementation of SNOMC on mesh nodes is the major aspect for the integration of SNOMC and MARWIS. Furthermore, some changes at the MARWIS' web-based graphical user interface were required so that the user can select the protocol to transmit the data to the selected sensor nodes.

8.3.1 Implementation of SNOMC on Wireless Mesh Nodes

SNOMC and the MARWIS Communication Server are implemented in C/C++ using UDP/TCP sockets. In contrast to the implementation in Contiki OS, we are using standard C templates (STL) such as linked lists to store the required information on each node. The SNOMC control structure is depicted in Figure 8.5(a). It contains the multicast identifier (*mc_id*) from type *int*. Then, it contains a list of receivers from type *std::list*. The entries of the list contain a structure containing the IP address of the receiver from type *struct sockaddr_in* and a *bool* which indicates if this receiver successfully received all fragments. Further, it includes the IP addresses of the sender, the last node, and the next node (both forwarder or branching node) from type *struct sockaddr_in*. Only the sender-driven mode of SNOMC is implemented.



(a) control structure (b) fragment list

Figure 8.5: SNOMC: structures.

The second structure describes the cached fragments and is shown in Figure 8.5(b). It describes the total number of fragments and the sequence number of the last fragment (both from type *int*). Further, it describes the cached fragments, which are stored in an own structure called *struct fragments*. This structure describes the sequence number and the length of the stored fragment (both from type *int*) and the data of the fragment from type *string*. The fragments are stored in the order according to their sequence number.

Table 8.1: Possible protocol combinations.

protocol	caching strategy	number
UDP-E2E	sender node	1
UDP-E2E	intermediate nodes	2
UDP-E2E	intermediate nodes, pro-active,	3
SNOMC	sender node	11
SNOMC	branching nodes	12
SNOMC	branching nodes, pro-active	13
SNOMC	intermediate nodes	14
SNOMC	intermediate nodes, pro-active	15
TCP	sender node	21

8.3.2 Adaptation of the MARWIS Graphical User Interface

Changes of the web-based graphical user interface were also necessary. The user can not only select the image and the sensor nodes, which should get the code update, but also the transport protocol. Two different transport protocols are selectable: SNOMC and UDP-E2E. For all protocols each of the tree caching strategies as *caching on sender node*, *caching on branching nodes*, and *caching on each intermediate node* can be chosen. Each combination of transport protocol and caching strategy is encoded by an integer which is transmitted to the MARWIS Communication Server. Possible combinations are shown in Table 8.1.

The MARWIS Communication Server provides functions for the communication. The function `snomc_join(rec_lst, caching_strat)` initiates the joining phase. The parameters are the list of receivers and the selected caching strategy. The functions returns a value indicating if the joining was successfully or not. The function `snomc_send_image(img *)` initiates the sending of the image. The according function for transmitting configuration or monitoring commands is called `snomc_send_cmd(cmd)`.

8.4 Evaluation

The section presents the quantitative evaluation of MARWIS with the integrated SNOMC protocol. The following features were evaluated: **(1)** reliable and time-efficient communication, and **(2)** energy-efficient operation.

8.4.1 Evaluation Scenario

After the integration of MARWIS and SNOMC we made several experiments in our Wisebed testbed. The evaluation was performed in a small network including a management station, two mesh nodes, and two sensor sub-networks with five sen-

sor nodes each, as shown in Figure 8.6. Each of the two mesh nodes are connected to sensor node gateways, which are connected to parts of the Wisebed testbed.

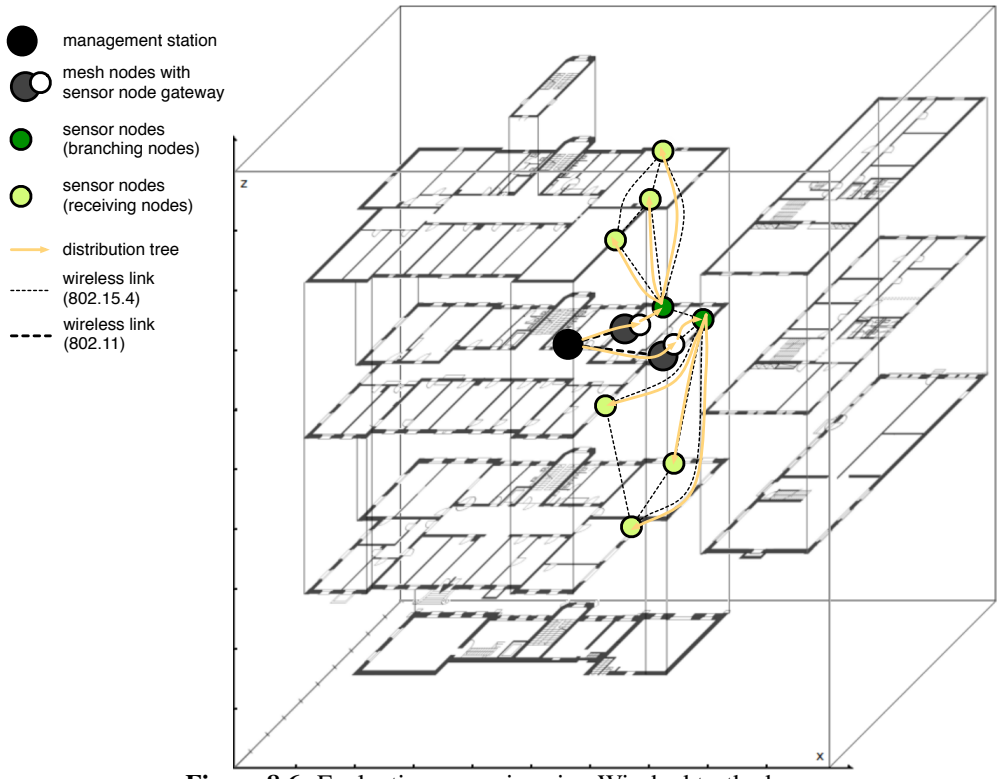


Figure 8.6: Evaluation scenario using Wisebed testbed.

We defined two evaluation scenarios. In the first one we distribute to the sensor nodes an image of an application that controls LEDs at sensor nodes. This application has a size of 1392 bytes, which corresponds to 20 data messages.

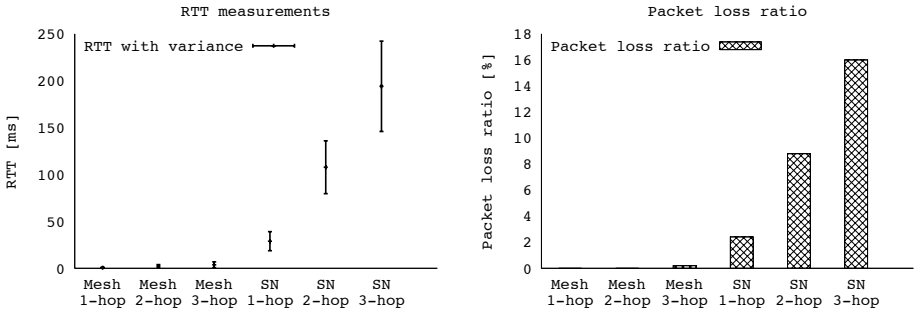
In the second scenario, we transmit a configuration command of 20 bytes to configure the sensing interval of a sensor on the sensor node. This corresponds to one data message.

We measure the time to transmit the code update (1392 bytes) and the configuration command (20 bytes) using either SNOMC or UDP-E2E. As MAC protocols we are using ContikiMAC and NullMAC, both integrated in Contiki OS. As sensor nodes Tmote SKY nodes are used. Further, we measure the energy consumption of the sensor nodes during the transmission time. The energy consumption of the mesh nodes including the sensor node gateways is left out. Overall we made 20 experimentation runs to evaluate both metrics.

8.4.2 Time-Efficient Communication

Preliminary Experiments without SNOMC

In a first experiment we evaluated the use of a wireless mesh network as a backbone to divide a large wireless sensor network into smaller sensor sub-networks [127]. For investigation of the additional round-trip time (RTT) and packet loss for increased hop count in wireless sensor and mesh networks, we made experiments with ICMP echo request (ping). The round-trip time was evaluated with one, two and three hop sensor node and mesh node links. The wireless sensor network tests were made with Tmote SKY nodes running a standard Contiki installation. As mesh nodes ALIX 3d2 mesh node with 128 MB RAM and two 60 mW IEEE 802.11g interfaces have been used. The sensor nodes and mesh nodes are located in different rooms. They are placed in such distance from each other that a node can only communicate with its one hop neighbours. Figure 8.7 shows the results of the experiments.



(a) RTT.

(b) Packet loss ratio.

Figure 8.7: Evaluation of reliable and time-efficient communication.

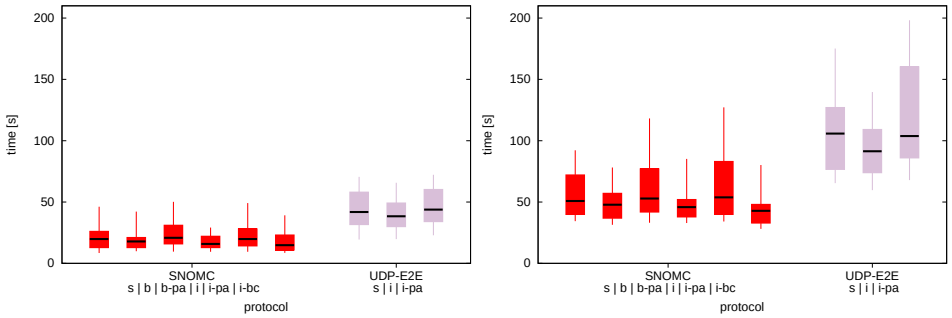
Figure 8.7(a) shows the different round-trip times in wireless mesh networks and wireless sensor networks for 500 measurements (pings). The results are self-explanatory: an additional sensor node hop causes over 50 times longer round-trip time than a additional mesh node hop.

Figure 8.7(b) shows the packet loss over 500 packets. As it can be seen using a wireless mesh network results in almost no packet loss. In a wireless sensor network we have a much higher packet loss, from approximately 2% (one hop) up to approximately 16% (three hops).

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. The further away a sensor node is located from the base station, the higher is the packet loss and the round-trip time. Retransmissions for lost packets can additionally increase the round-trip time and jitter. A large wireless sensor network with large number of hops (e.g., more than 30) will be unserviceable.

Experiments with SNOMC using in-house Testbed

After the integration of SNOMC and MARWIS we executed several experiments using the Wisebed testbed. Figure 8.8 shows the transmission time for a code update of 1392 bytes using SNOMC or UDP-E2E with the different caching strategies in combination with NullMAC or ContikiMAC.



(a) NullMAC. (b) ContikiMAC.

Figure 8.8: Evaluation: transmission time, code update (1392 bytes).

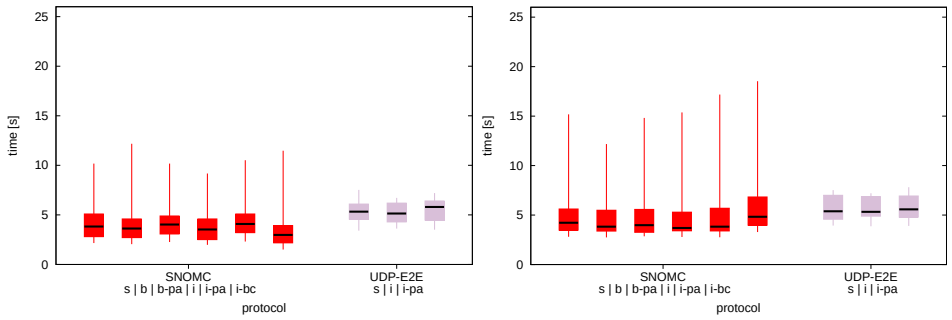
As we can see SNOMC performs, in the different combinations, better than UDP-E2E by approximately a factor of two. Due to Contiki OS limitations it is only possible to have four parallel UDP flows. Therefore, to address six receivers we have to transmit the data in two steps with three parallel UDP flows in each step. That is one reason why UDP-E2E has a higher transmission time.

In general, the caching strategy did not show to have a big effect. In case of caching only on sender node, the fragments are only cached on the management station. In case of caching on branching nodes, the fragments are cached on each of the two branching nodes (green circles in Figure 8.6). In case of caching on every intermediate node the fragments are cached on every mesh node and every sensor node, including the sensor node gateways. In the last case, a maximum of four hops are necessary to request a lost fragment. Thus, the caching strategy has not a big influence.

In case of pro-active requests of missed fragments we can reduce the number of hops of a request from the receivers. However, a pro-active request causes higher traffic and thus a higher collision probability. Thus, as a conclusion the negative effect of higher traffic neutralizes the positive effect of reduced number of hops for requesting missing fragments.

Comparing NullMAC and ContikiMAC, we see that NullMAC is faster by a factor of three. Further, we can see that the broadcast on the last hop to the receivers has a positive influence in case of using NullMAC. Using ContikiMAC, the broadcast optimization turns into a disadvantage due to the way how ContikiMAC handles broadcast transmissions. More specifically, ContikiMAC broadcasts a packet continuously for a time period of 128ms until it can be sure that each neighbour was awake during this time period and able to receive the broadcasted

packet. In contrast, using NullMAC every node is always awake and thus NullMAC just needs to broadcast the packet ones.



(a) NullMAC. (b) ContikiMAC.

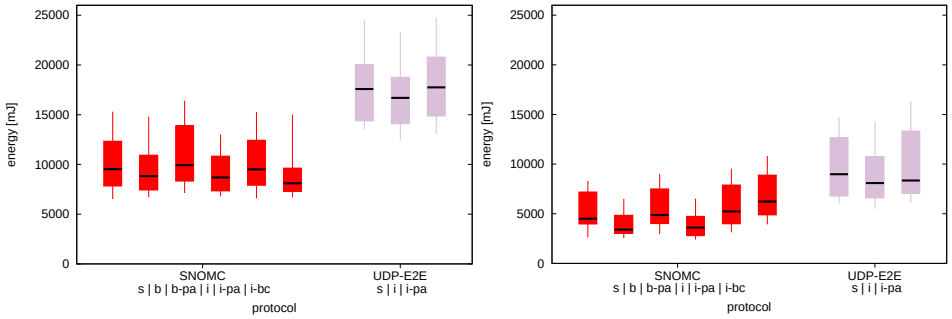
Figure 8.9: Evaluation: transmission time, configuration command (20 bytes).

Figure 8.9 shows the transmission time for transmitting one configuration message to the six receiver nodes. The statistical spread is much lower than in the previous scenario. We have just one data packet to transmit and this mainly works without any problems, because we do not have many packets in the network that could interfere each other. The caching strategy has almost no influence for the same reason. The outliers occur if after a successful join the data packet gets lost. In this case the receivers have a timer of five seconds to wait for the first packet. After this timer expires the receiver requests for the first data fragment. The difference in transmission time between SNOMC and UDP-E2E is caused by the same reason as described above (no more than four parallel UDP flows possible in Contiki OS). UDP-E2E has no join procedure whereas SNOMC has. Thus, if the only data fragment gets lost, the receivers do not know that there was a transmission. Hence, there is no such timer, which makes a request after five seconds as it is in SNOMC. And thus, UDP-E2E has no outliers as SNOMC has.

8.4.3 Energy-Efficient Operation

In this section we discuss the results of energy consumption. It is measured during the transmission time of a 1392 bytes code update and a 20 bytes configuration command. For the measurement we are using the Contiki in-built module *Powertrace* [36].

Figure 8.10(a) shows the results for the code update using SNOMC and UDP-E2E in combination with NullMAC. Since NullMAC is an always-on MAC protocol (cf. Section 2.2.2) the results for the energy consumption are linear to the transmission time. The longer a transmission takes the higher is the consumed energy. Thus, the energy-consumption with UDP-E2E is higher than with SNOMC. The broadcast optimization of SNOMC has the lowest energy consumption. The use of pro-active requests for lost fragments does not only cause a higher transmission time, but also a higher energy consumption, as result of higher number of

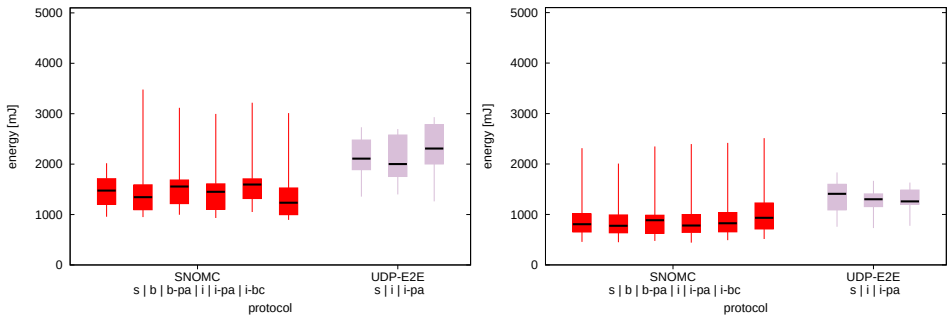


(a) NullMAC. (b) ContikiMAC.

Figure 8.10: Evaluation: energy consumption, code update (1392 bytes).

packets and higher traffic.

Figure 8.10(b) shows the results for the code update using SNOMC and UDP-E2E in combination with ContikiMAC. Compared to NullMAC the use of ContikiMAC decreases the energy consumption approximately by the factor of two. ContikiMAC is an energy-saving protocol using duty cycles. But, with bursty traffic as we have it by transmitting 1392 bytes, ContikiMAC reduces the sleeping cycles and awakes more often.



(a) NullMAC. (b) ContikiMAC.

Figure 8.11: Evaluation: energy consumption, configuration command (20 bytes).

Figure 8.11(a) shows the results for transmitting a configuration command to the sensor nodes using NullMAC. Also in this case the energy consumption corresponds to the transmission time. There are very few outliers, because just one data packet is transmitted, which leads to very little interference problems in most of the experiments. UDP-E2E has no big outliers. The best effort has the SNOMC broadcast optimization.

Figure 8.11(b) shows the energy consumption using SNOMC and UDP-E2E in combination with ContikiMAC. Compared to NullMAC we reduce the energy consumption by the factor of approximately two. SNOMC has again bigger outliers, while UDP-E2E has only very little outliers. The reasons are described above.

8.5 Conclusions

This chapter focussed on the integration of SNOMC into the MARWIS architecture. For this we adapted the MARWIS architecture. We added a MARWIS Communication Server, which handles the communication in the wireless mesh network. Furthermore, changes are done on the MARWIS graphical user interface to allow the user to select the used transport protocol. Thus, SNOMC can be used in an heterogeneous wireless sensor network which is supported by a wireless mesh backbone.

We evaluated the co-operation of SNOMC and MARWIS by running experiments using the in-house testbed. In these experiments we transmitted a code update (1392 bytes) and a configuration command (20 bytes) to six sensor nodes. The results of the experiment showed that the combination of SNOMC and MARWIS supports reliably, time- and energy-efficient operation of a heterogeneous wireless sensor network.

Chapter 9

Conclusions and Outlook

While wireless sensor networks have experienced an increasing degree of (academic) research interests in the last decade, nowadays research in wireless sensor networks increasingly finds its way into a growing number of industrial, environmental and business applications.

Wireless sensor networks deployed for real-world applications are composed of different types of sensor nodes fulfilling various tasks, depending on the application purpose. The operation of such wireless sensor networks needs to be cost-efficient, energy-efficient and to ensure functional reliability.

These requirements, despite the wide-spread adoption of wireless sensor networks, motivate the still big potential for further research in the area of wireless sensor networks and are the driver behind the work performed in this thesis.

9.1 Addressed Challenges

A heterogeneous wireless sensor network consists of several different types of sensor nodes. Various applications supporting different tasks, e.g., event detection, localization, tracking, and environmental monitoring, may run on these specialized sensor nodes. In addition, new applications have to be deployed and new configurations and bug fixes have to be applied during the network lifetime.

In a network with thousands of nodes, this becomes a very complex task and a general management architecture is required. One the one hand, the management architecture should address **monitoring**, **(re)configuration**, and **code updating** of wireless sensor nodes. Although all three tasks are not high priority tasks, they should be executed in reasonable time and most importantly conforming to certain quality of execution. In particular, code updating and node configuration are both critical tasks, the proper execution of which is required for the healthy operation of the wireless sensor network.

On the other hand, the operation of the management architecture should ensure that several essential functional requirements are met. Referring to the importance of the management tasks, **end-to-end reliability** and **time-efficiency** become key necessities in the realisation of code updates and node configuration.

In addition, the limited energy capacity of the battery-operating sensor nodes sets **energy-efficient operation** as another important requirement towards the management tasks. Future wireless sensor networks will combine a multitude of various sensor platforms and, therefore, require the ability to handle **network heterogeneity**. Each of these challenges alone have been studied a lot in the literature. It is their combined support, however, that is still not fully achieved.

In order to fill the gap in research this thesis proposes a management architecture able to provide monitoring, (re)configuration, and code updating in a reliable, energy- and time-efficient manner for heterogeneous wireless sensor networks. For the realisation of the communication within the architecture a reliable, time-, and energy-efficient multicast protocol was developed.

9.2 Main Contributions and Summary

The contributions presented in this thesis are the result of our work to offer a solution to the challenges addressed in the previous section. In Part I we presented the design, implementation, and evaluation of the SNOMC (Sensor Node Overlay Multicast) protocol (Chapters 3 - 5). In Part II MARWIS the Management Architecture for Wireless Sensor Networks (Chapters 6 and 7) and the integration of SNOMC and MARWIS (Chapter 8) are presented.

SNOMC (Sensor Node Overlay Multicast)

The Sensor Node Overlay Multicast (SNOMC) protocol was specifically designed for data dissemination using overlay multicast in wireless sensor networks. It supports reliable, time-efficient and energy-efficient dissemination of bulky data from one sender node to many receiver nodes. SNOMC was especially designed to use it in the MARWIS architecture. It distributes management data, such as code updates or configuration commands to selected sensor nodes in an efficient and reliable way.

A distribution tree is used to transmit the data efficiently from the sender node to the receiver nodes using overlay connections. The distribution tree is formed during a *joining phase* and is composed of one *sender node*, several *forwarding nodes* and *branching nodes*, as well as one or more *receiver nodes*. The distribution tree is formed during a *joining phase* and is composed of one *sender node*, several *forwarding nodes* and *branching nodes*, as well as one or more *receiver nodes*. To optimise performance and avoid unnecessary unicast transmissions we introduced broadcast transmission at the branching nodes in case of two ore more following nodes.

The support of end-to-end reliability is realised by deploying a NACK-based reliability mechanism, combined with a data acknowledgement after successful reception of all data fragments by the receiver nodes. Caching was adopted to decrease the number of required end-to-end retransmissions. More specifically three

caching strategies were used, namely, caching on each intermediate node, caching on branching nodes, or caching only on the sender node. Moreover, an option was included to pro-actively request missing fragments. In this case each intermediate sensor node, which caches fragments can pro-actively request a missed fragment.

In order to evaluate the performance of SNOMC, on the one hand, we compared it to other popular data dissemination protocols for wireless sensor networks. These were Flooding, MPR (Multipoint Relay), PSFQ (Pump Slowly, Fetch Quickly), TinyCubus, and Directed Diffusion as well as the unicast-based protocols UDP and TCP. As underlying MAC protocols we chose BEAM, ContikiMAC and NullMAC for the simulations, and ContikiMAC and NullMAC for the real-world evaluation.

Three performance metrics were chosen for evaluation, namely, transmission time, number of transmitted packets and energy consumption. Our findings were presented in Chapter 5. We showed that the SNOMC protocol performs better compared to the selected data dissemination protocols, both transmitting 20 bytes (refers to a configuration command) and 1000 bytes (refers to a code update). In particular, data dissemination protocols relying on broadcast, such as Flooding, MPR, and TinyCubus cause broadcast storms and perform poorly. Performance improves for data dissemination protocols that are especially designed for wireless sensor networks, such as Directed Diffusion or PSFQ, but, as they also have some protocol phases relying on broadcast transmissions, SNOMC still outperforms them. The most serious competitor of SNOMC is UDP-E2E, in which case the advantage of SNOMC depends on the network topology and the distribution tree as we showed in Section 5.3.3.

The performance with different caching strategies was investigated as well. In general, our observations show that caching on intermediate nodes improves the performance independently of the tested protocol. That is due to the avoidance of expensive end-to-end retransmissions. We further demonstrated that pro-active requesting of missed fragments does not improve the performance at all. It only causes additional packets to be sent, increasing the probability of collisions and cancelling the advantage of SNOMC's optimization.

Finally, we showed that SNOMC can deliver good performance in various environments and with different underlying MAC protocols, which support different levels of reliability and energy-efficiency. In conclusion, SNOMC can offer a robust, high-performing solution for the efficient distribution of code updates and management information in a wireless sensor network.

MARWIS (Management Architecture for Wireless Sensor Networks)

The MARWIS architecture is the answer we offer to a comprehensive management of heterogeneous wireless sensor networks. The architecture specifically takes into account the operation of such networks in reliable, time- and energy-efficient manner and its design was described in Chapter 6. A distinguished feature of MARWIS is the use of a wireless mesh backbone, which is the main enabler of the commu-

nication and interaction among the various sensor network types. Moreover, a mesh-based backbone allows for the offloading of functionality from the sensor nodes to the mesh nodes and by doing so decreasing computational requirements towards the sensor nodes.

Chapter 6 introduced the design decisions taken in the development of MARWIS, the architecture components and underlying management protocols. In terms of components, MARWIS consists of sensor node(s) and a user interface as well as management station(s) and management (mesh) node(s), which bridge the interaction between the first two groups. In terms of protocols, solutions for monitoring, configuration and code updates of the sensor nodes were offered to ensure the smooth co-operation of the MARWIS components.

Following the design, Chapter 7 focused on the implementation of the MARWIS components. On the one hand, a web-based graphical user interface was implemented on the management station (a mesh node) to allow user interaction with the sensor nodes. Moreover, the MARWIS Server implementation included the corresponding management modules and databases (also on mesh nodes). On the other hand, a SN Agent was implemented on the sensor nodes to enable the management tasks.

To perform the management task in a reliably, time- and energy-efficient way, we integrated SNOMC into MARWIS. The changes required for the integration of MARWIS and SNOMC were described in Chapter 8. Using SNOMC in MARWIS allows us transmit code updates and configuration command efficiently to the selected sensor nodes, independent from the underlying MAC protocol The evaluation of the integrated system itself was discussed in detail in the same Chapter 8. We showed that SNOMC can be used in a heterogeneous wireless sensor network, where the networks backbone is formed by wireless mesh nodes, and, more importantly, that this is done in a reliable, time- and energy-efficient manner. This was evaluated for large as well as small (single packet) payloads. Therefore, we claim that the combination of MARWIS and SNOMC offers an optimal solution for the management of sensor network comprising multiple sensor type domains.

9.3 Outlook

We showed that our overlay multicast protocol SNOMC provides reliable, time- and energy-efficient one-to-many communication in wireless sensor networks and in combination with the newly proposed management architecture MARWIS, management tasks can be provided efficiently in heterogeneous wireless sensor networks. We can imagine several possible improvements of each contribution especially related to real-world deployments.

1. While the main communication pattern in wireless sensor networks is many-to-one, management tasks follow more the one-to-many communication pattern. In a real-world wireless sensor network both communication patterns

can appear. SNOMC can be enhanced to offer many-to-one data dissemination. The distribution tree can be reused inversely to collect the data from the sensor nodes, with reliability and energy-efficiency as key aspects. The operational mode of the enhanced SNOMC can be then driven by the sensor nodes, which want to transmit data or driven by the sink node which wants to gather the data from several sensor nodes.

2. The performance of data dissemination in wireless sensor networks affects all layers of the protocol stack. Harmonization between the protocols across different layers is an important target for optimizations. The SNOMC protocol is located at the application layer and uses UDP as data transport protocol. Although our evaluations showed that SNOMC plus UDP perform well in combination with various MAC protocols, cross-layer information passed from or to the MAC protocol would improve the performance of SNOMC.

3. An Internet Engineering Task Force (IETF) working group has standardized the transmission of IPv6 packets over IEEE 802.15.4 low power wireless personal area network (LoWPAN) as the 6LoWPAN protocol [77]. Since IPv6 multicast has been left out in 6LoWPAN, the Trickle Multicast Internet draft [53] tries to fill this gap. A still unsolved problem remains in reliable data transport.

4. In the area of management we envision improvements of code updating. So far MARWIS only supports updating of applications but not of the complete image of the operating system. An image of the whole operating system is much bigger (up to 30KB) than the image of an application. To find techniques to transmit, store, replace, and start the complete operating system to a sensor node is a new field of research.

5. In order to reduce the number of necessary transmissions for code updating the use of compression techniques or the use of diff-like or incremental updates is a next step to improve MARWIS performance. This should benefit transmission time and energy consumption. Evaluations have to show if uncompressing or reconstructing the image on the sensor node would not cost more energy than the saved energy for transmission.

Bibliography

[1] "A4-Mesh: Authentication, Authorization, Accounting, and Auditing in Wireless Mesh Networks." [Online]. Available: https://a4-mesh.unibe.ch/

[2] "ADAM: Administration and Deployment of Adhoc Mesh networks." [Online]. Available: http://cds.unibe.ch/research/software.html

[3] "Castalia - a simulator for Wireless Sensor Networks." [Online]. Available: http://castalia.npc.nicta.com.au

[4] "CTI-Mesh: Wireless Mesh Networks for Interconnection of Remote Sites to Fixed Broadband Networks." [Online]. Available: http://cds.unibe.ch/research/cti-mesh.html

[5] "dhtmlxtree: JavaScript Tree Menu." [Online]. Available: http://dhtmlx.com/

[6] "ELF: Executable and Linkable Format." [Online]. Available: http://www.linux-kernel.de/appendix/ap05.pdf

[7] "GloMoSim: A Scalable Simulation Environment for Wireless and Wired Network Systems." [Online]. Available: http://pcl.cs.ucla.edu/projects/domains/glomosim.html

[8] "LBA: Location Based Analyser." [Online]. Available: http://cds.unibe.ch/research/lba/index.html

[9] "OMNeT++: Discrete Event Simulation System." [Online]. Available: http://www.omnetpp.org

[10] "Scalable Network Technologies, Qualnet." [Online]. Available: http://www.scalable-networks.com.

[11] A. Varga, "INET Framework, an open-source communication networks simulation package for the OMNeT++ simulation environment." [Online]. Available: http://inet.omnetpp.org/

[12] M. Anwander, G. Wagenknecht, and T. Braun, "Management of wireless sensor networks using tcp/ip," in *Proceedings of the International Workshop*

on Sensor Network Engineering (IWSNE'08), Santorini Island, Greece, June 2008, pp. II.1–II.8.

[13] M. Anwander, G. Wagenknecht, T. Braun, and K. Dolfus, "Beam: A burst-aware energy-efficient adaptive mac protocol for wireless sensor networks." in *Proceedings of the International Conference on Networked Sensing Systems (INSS'10)*, Kassel, Germany, June 2010, pp. 195–202.

[14] M. Anwander, G. Wagenknecht, T. Staub, and T. Braun, "Management of Heterogenous Wireless Sensor Networks," in *Proceedings of the 6. GI/ITG KuVS Fachgespraech Drahtlose Sensornetze (FGSN'07)*, Aachen, Germany, July 2007.

[15] Atmel: ATmega128L micro-controller. [Online]. Available: http://www.atmel.com/devices/atmega128.aspx

[16] M. Baar, E. Koeppe, A. Liers, and J. Schiller, "The ScatterWeb MSB-430 Platform for Wireless Sensor Networks." SICS Contiki Workshop, Kista, Sweden, March 2007.

[17] S. Bhattacharyya, "RFC 3569 An Overview of Source- Specific Multicast," Internet Engineering Task Force, July 2003.

[18] R. Braden, "RFC 1122 Requirements for Internet Hosts - Communication Layers," Internet Engineering Task Force, October 1989. [Online]. Available: http://tools.ietf.org/html/rfc1122

[19] BTnode rev3.24 Datasheet. [Online]. Available: http://www.btnode.ethz.ch/pub/files/btnode_rev3.24_productbrief.pdf

[20] BTnodes - A Distributed Environment for Prototyping Ad Hoc Networks. [Online]. Available: http://www.btnode.ethz.ch/

[21] M. Buettner, V. Gary, E. Anderson, and R. Han, "X-MAC: A Short Preamble MAC Protocol for Duty-cycled Wireless Sensor Networks." ACM Conference on Embedded Networked Sensor Systems (SenSys), Boulder, USA, November 2006, pp. 307–320.

[22] V. Cerf, Y. Dalal, and C. Sunshine, "Specification of Internet Transmission Control Program," RFC 675, Internet Engineering Task Force, December 1974. [Online]. Available: http://www.ietf.org/rfc/rfc675.txt

[23] P. Chaporkar and S. Sarkar, "Wireless multicast: Theory and apporaches," *IEEE Transactions on Information Theory*, vol. 51, no. 6, pp. 1954–1972, Jun 2005.

[24] C.-K. Chiang and C.-T. King, "Source routing for overlay multicast in wireless ad hoc and sensor networks," in *Proceedings of the International Conference on Parallel Processing Workshops (ICPPW '07)*, Washington, DC, USA, Sep 2007, p. 75.

[25] C. D. Cordeiro, D. F. H. Sadok, J. Kelner, and P. da F. Pinto, "Establishing a trade-off between unicast and multicast retransmission modes for reliable multicast protocol," in *Proceedings of the International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication (MASCOTS'00)*, San Francisco, CA, USA, Aug/Sep 2000, pp. 85–91.

[26] G. Coulson, B. Porter, I. Chatzigiannakis, C. Koninis, S. Fischer, D. Pfisterer, D. Bimschas, T. Braun, P. Hurni, M. Anwander, G. Wagenknecht, S. Fekete, A. Kröller, and T. Baumgartner, "Flexible experimentation in wireless sensor networks," *Communications of the ACM*, vol. 55, no. 1, January 2012.

[27] Crossbow: MICA2 Datasheet. [Online]. Available: http://bullseye.xbow.com:81/Products/Product_pdf_files/Wireless_pdf/MICA2_Datasheet.pdf

[28] Crossbow: MICAz Datasheet. [Online]. Available: http://bullseye.xbow.com:81/Products/Product_pdf_files/Wireless_pdf/MICAz_Datasheet.pdf

[29] Crossbow: TelosB Datasheet. [Online]. Available: http://www.willow.co.uk/TelosB_Datasheet.pdf

[30] Crossbow: Tmote Sky Datasheet. [Online]. Available: http://www.snm.ethz.ch/snmwiki/pub/uploads/Projects/tmote_sky_datasheet.pdf

[31] S. Das, H. Pucha, and Y. Hu, "Distributed hashing for scalable multicast in wireless ad hoc networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 3, pp. 347–362, March 2008.

[32] Deliverable D4.1: First Set of well-designed Simulations, "Experiments and possible Benchmarks. Technical Report," June 2008. [Online]. Available: http://www.wisebed.eu

[33] A. Dunkels, L. Mottola, N. Tsiftes, F. Osterlind, J. Eriksson, and N. Finne, "The Announcement Layer: Beacon Coordination for the Sensornet Stack." European Conference on Wireless Sensor Networks (EWSN'11), Bonn, Germany, February 2011, pp. 211–226.

[34] A. Dunkels, "Full TCP/IP for 8-Bit Architectures," in *Proceedings of the International Conference on Mobile Systems, Applications and Services (MobiSys'03)*, San Francisco, CA, USA, May 2003, pp. 85–98.

[35] ——, "RIME - A Lightweight Layered Communication Stack for Sensor Networks," in *Proceedings of the European Conference on Wireless Sensor Networks (EWSN)*, ser. Poster/Demo session, 2007.

[36] A. Dunkels, J. Eriksson, N. Finne, and N. Tsiftes, "Powertrace: Network-level Power Profiling for Low-power Wireless Networks," Swedish Institute of Computer Science, Tech. Rep. T2011:05, Mar. 2011.

[37] A. Dunkels, N. Finne, J. Eriksson, and T. Voigt, "Run-Time Dynamic Linking for Reprogramming Wireless Sensor Networks," in *Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys'06)*, Boulder, CO, USA, Oct/Nov 2006, pp. 15–28.

[38] A. Dunkels, B. Grönvall, and T. Voigt, "Contiki - A Lightweight and Flexible Operating System for Tiny Networked Sensors," in *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks*, ser. LCN '04, vol. 0. Washington, DC, USA: IEEE Computer Society, 2004, pp. 455–462.

[39] A. Dunkels, O. Schmidt, T. Voigt, and M. Ali, "Protothreads: Simplifying Event-driven Programming of Memory-constrained Embedded Systems," in *Proceedings of the 4th international conference on Embedded networked sensor systems*, ser. SenSys '06. New York, NY, USA: ACM, 2006, pp. 29–42.

[40] A. Dunkels, F. Österlind, and Z. He, "An adaptive communication architecture for wireless sensor networks," in *Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys'07)*, Sydney, Australia, Nov 2007, pp. 335–349.

[41] A. Dunkels, F. Österlind, N. Tsiftes, and Z. He, "Software-based on-line energy estimation for sensor nodes," in *Proceedings of the IEEE Workshop on Embedded Networked Sensors (EmNetS'07)*, Cork, Ireland, Jul 2007, pp. 28–32.

[42] E. Ertin, A. Arora, R. Ramnath, and M. Nesterenko, "Kansei: A Testbed For Sensing At Scale." ACM/IEEE International Conference on Information Processing In Sensor Networks (IPSN), Nashville, Tennessee, USA, April 2006, pp. 399–406.

[43] C.-H. Feng and W. B. Heinzelman, "Rbmulticast: Receiver based multicast for wireless sensor networks," in *Proceedings of the IEEE Wireless Communication & Networking Conference (WCNC'09)*, Budapest, Hungary, Apr 2009, pp. 2672–2677.

[44] Flot: Attractive JavaScript Plotting for jQuery. [Online]. Available: http://www.flotcharts.org/

[45] S. Floyd, V. Jacobson, C.-G. Liu, S. McCanne, and L. Zhang, "A reliable multicast framework for light-weight sessions and application level framing," *Transactions on Networking*, vol. 5, no. 6, pp. 784–803, Dec 1997.

[46] R. Flury and R. Wattenhofer, "Routing, anycast, and multicast for mesh and sensor networks," in *Proceedings of the IEEE Computer and Communications Societies (INFOCOM'07)*, Anchorage, AK, USA, May 2007, pp. 946–954.

[47] Freescale Corporation: MMA7260Q XYZ Three-Axis Low g Acceleration Sensor. [Online]. Available: http://www.freescale.com

[48] FTDI: USB Device Solutions. [Online]. Available: http://www.ftdichip.com/

[49] L. Girod, M. Lukac, A. Parker, T. Stathopoulos, J. Tseng, H. Wang, D. Estrin, R. Guy, and E. Kohler, "A reliable multicast mechanism for sensor network applications," Center for Embedded Network Sensing, University of California, Los Angeles, Los Angeles, CA, USA, Technical Report 48, Apr 2005.

[50] Graphviz: A Graph Visualization Software. [Online]. Available: http://www.graphviz.org

[51] V. Handziski, A. Koepke, A. Willig, and A. Wolisz, "TWIST: A Scalable and Reconfigurable Testbed for Wireless Indoor Experiments with Sensor Network." ACM/SIGMOBILE International Workshop on Multi-hop Ad Hoc Networks (REALMAN), Florence, Italy, May 2006.

[52] M. Hosseini, D. T. Ahmed, S. Shirmohammadi, and N. D. Georganas, "A survey of application-layer multicast protocols," *Communications Surveys & Tutorials*, vol. 9, no. 3, pp. 58–74, 3rd Quarter 2007.

[53] J. Hui and P. Thubert, "Multicast forwarding using trickle," Internet Engineering Task Force, April 2011. [Online]. Available: http://tools.ietf.org/html/draft-ietf-roll-trickle-mcast-00

[54] J. W. Hui and D. Culler, "The dynamic behavior of a data dissemination protocol for network programming at scale," in *Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys'04)*, Baltimore, MD, USA, Nov 2004, pp. 81–94.

[55] P. Hurni, M. Anwander, G. Wagenknecht, T. Staub, and T. Braun, "TARWIS - A Testbed Management Architecture for Wireless Sensor Network Testbeds." IEEE/IFIP Network Operations and Management Symposium (NOMS), Maui, Hawaii, USA, April 2012, pp. 611–614.

[56] ——, "TARWIS - A Testbed Management Architecture for Wireless Sensor Network Testbeds." International Conference on Network and Service Management (CNSM), Short Paper Session, Paris, France, October 2011, pp. 1–5.

[57] P. Hurni, T. Staub, G. Wagenknecht, M. Anwander, and T. Braun, "A Secure Remote Authentication, Operation and Management Infrastructure for Distributed Wireless Sensor Network Testbeds." First Workshop on Global Sensor Networks (GSN'09), co-located with KiVS'09, Kassel, Germany, March 2009.

[58] P. Hurni, G. Wagenknecht, M. Anwander, T. Staub, and T. Braun, "A Testbed Management Architecture for Wireless Sensor Network Testbeds (TARWIS)." European Conference on Wireless Sensor Networks (EWSN), Coimbra, Portugal, February 2010, pp. 33–35.

[59] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva, "Directed diffusion: A scalable and robust communication paradigm for sensor networks," in *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'00)*, Boston, Massachussetts, Aug 2000, pp. 56–67.

[60] ——, "Directed diffusion for wireless sensor networking," *IEEE/ACM Transactions on Networking*, vol. 11, no. 1, pp. 2–16, Feb 2002.

[61] L. Ji and M. S. Corson, "Differential destination multicast - a manet multicast routing protocol for small groups," in *Proceedings of the IEEE Computer and Communications Societies (INFOCOM'01)*, Lusheng Ji, M. Scott Corson, Apr 2001, pp. 1192–1201.

[62] D. Koutsonikolas, S. Das, Y. C. Hu, and I. Stojmenovic, "Hierarchical geographic multicast routing for wireless sensor networks," in *Proceedings of the International Conference on Sensor Technologies and Applications (SENSORCOMM'07)*, Valencia, Spain, Oct 2007, pp. 347–354.

[63] T. Kunz and E. Cheng, "Multicasting in ad-hoc networks: Comparing maodv and odmrp," in *Proceedings of the Workshop on Ad hoc Communications (WADHC'01)*, Bonn, Germany, Sep 2001.

[64] J. Lee, E. Lee, S. Park, S. Oh, and S.-H. Kim, "Consecutive geographic multicasting protocol in large-scale wireless sensor networks," in *Proceedings of the IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC'10)*, Istanbul, Turkey, Sep 2010, pp. 2192–2197.

[65] B. N. Levine and J. Garcia-Luna-Aceves, "A comparison of reliable multicast protocols," *Multimedia Syst.*, vol. 6, no. 5, pp. 334–348, Sept. 1998.

[66] P. Levis, S. Madden, J. Polastre, R. Szewczyk, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler, "TinyOS: An Operating System for Sensor Networks." Ambient Intelligence Part II, Springer Verlag, USA, 2004, pp. 115–148.

[67] P. Levis and D. Culler, "Mate: A tiny virtual machine for sensor networks," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-X)*, San Jose, CA, USA, Oct 2002, pp. 85–95.

[68] P. Levis, N. Lee, M. Welsh, and D. Culler, "Tossim: Accurate and scalable simulation of entire tinyos applications," in *Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys'03)*, Los Angeles, CA, USA, Nov 2003, pp. 126 –137.

[69] P. Levis, N. Patel, S. Shenker, and D. Culler, "Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks," in *Proceedings of the Conference on Symposium on Networked Systems Design and Implementation (NSDI'04)*, San Francisco, CA, USA, Mar 2004, pp. 15–28.

[70] T. Liu and M. Martonosi, "Impala: A middleware system for managing autonomic, parallel sensor systems," in *Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'03)*, San Diego, CA, USA, Jun 2003, pp. 107–118.

[71] S. Makharia, D. Raychaudhuri, M. Wu, H. Liu, and D. Li, "Experimental study on wireless multicast scalability using merged hybrid arq with staggered adaptive fec," in *Proceedings of the IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM'08)*, Newport Beach, CA, USA, Jun 2008.

[72] M. Marin-Perianu and P. Havinga, "Rmd: Reliable multicast data dissemination within groups of collaborating objects," in *Proceedings of the IEEE International Conference on Local Computer Networks (LCN'06)*, Tampa, FL, USA, Nov 2006, pp. 656–663.

[73] P. J. Marron, A. Lachenmann, D. Minder, J. Hähner, R. Sauter, and K. Rothermel, "Tinycubus: A flexible and adaptive framework for sensor networks," in *Proceedings of the European Conference on Wireless Sensor Networks (EWSN'05)*, Istanbul, Turkey, Jan 2005, pp. 278–289.

[74] P. J. Marron, D. Minder, A. Lachenmann, and K. Rothermel, "Tinycubus: An adaptive cross-layer framework for sensor networks," *it - Information Technology*, vol. 47, no. 2, pp. 87–97, Feb 2005.

[75] M. Mauve, H. Füssler, J. Widmer, and T. Lang, "Position-based multicast routing for mobile ad-hoc networks," Department of Computer Science, University of Mannheim, Mannheim, Germany, Technical Report TR-03-004, Mar 2003.

[76] Micron: Serial Flash Memory M25P80: 8 Mbit, low voltage, serial Flash memory with 75 MHz SPI bus interface. [Online]. Available: http://www.micron.com/~/media/Documents/Products/Data%20Sheet/NOR%20Flash/Serial%20NOR/M25P/M25P80.pdf

[77] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, "RFC 4944 Transmission of IPv6 packets over IEEE 802.15.4 networks," Internet Engineering Task Force, September 2007. [Online]. Available: http://tools.ietf.org/html/rfc4944

[78] "The Network Simulator NS-2." [Online]. Available: http://www.isi.edu/nsnam/ns/

[79] A. Okura, T. Ihara, and A. Miura, "Bam: Branch aggregation multicast for wireless sensor networks," in *Proceeding of the IEEE International Conference on Mobile Adhoc and Sensor Systems Conference (MASS'05)*, Washington, DC, USA, Nov 2005, p. 10.

[80] M. Pandey and D. Zappala, "Hop-by-hop multicast transport for mobile ad hoc wireless networks," in *Proceedings of the IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS'08)*, Atlanta, GA, USA, Sep/Oct 2008, pp. 450–455.

[81] PC Engines GmbH: ALIX system boards. [Online]. Available: http://www.pcengines.ch/alix3d2.htm

[82] C. Perkins, E. Belding-Royer, and S. Das, "RFC 3561 Ad hoc On-Demand Distance Vector (AODV) Routing," Internet Engineering Task Force, July 2003. [Online]. Available: http://tools.ietf.org/html/rfc3561

[83] H. N. Pham, D. Pediaditakis, and A. Boulis, "From simulation to real deployments in wsn and back," in *World of Wireless, Mobile and Multimedia Networks, 2007. WoWMoM 2007. IEEE International Symposium on a*, june 2007, pp. 1 –6.

[84] PHP: Hypertext Preprocessor. [Online]. Available: http://www.php.net/

[85] D. Plummer, "Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware," RFC 826 (Standard), Internet Engineering Task Force, Nov. 1982. [Online]. Available: http://www.ietf.org/rfc/rfc826.txt

[86] J. Polastre, J. Hill, and D. Culler, "Versatile Low Power Media Access for Wireless Sensor Networks." ACM Conference on Embedded Networked Sensor Systems (SenSys), Baltimore, USA, November 2004, pp. 95–107.

[87] J. Postel, "User Datagram Protocol," RFC 768 (Standard), Internet Engineering Task Force, August 1980. [Online]. Available: http://www.ietf.org/rfc/rfc768.txt

[88] ——, "Internet Control Message Protocol," RFC 792 (Standard), Internet Engineering Task Force, Sept. 1981, updated by RFCs 950, 4884. [Online]. Available: http://www.ietf.org/rfc/rfc792.txt

[89] ——, "RFC 791 Internet Protocol - DARPA Inernet Programm, Protocol Specification," Internet Engineering Task Force, September 1981.

[90] A. Quayyum, L. Viennot, and A.Laouiti, "Multipoint relaying: An Efficient Technique for Flooding in Mobile Wireless Networks," INRIA, Sophia Antipolis, France, Tech. Rep., 2000.

[91] D. Raychaudhuri, I. Seskar, M. Ott, S. Ganu, K. Ramachandran, H. Kremo, R. Siracusa, H. Liu, and M. Singh, "Overview of the ORBIT Radio Grid Testbed for Evaluation of Next-Generation Wireless Network Protocols." Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC'05), New Orleans, USA, March 2005.

[92] O. Rensfelt, F. Hermans, L. Larzon, and P. Gunningberg, "Sensei-UU: a relocatable Sensor Network Testbed." ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization (WiNTECH), Chicago, USA, September 2010, pp. 63–70.

[93] L. Rizzo and L. Vicisano, "Rmdp: an fec-based reliable multicast protocol for wireless environments," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 2, no. 2, pp. 23–31, Apr 1998.

[94] J. Romkey, "Nonstandard for transmission of IP datagrams over serial lines: SLIP," RFC 1055 (Standard), Internet Engineering Task Force, June 1988. [Online]. Available: http://www.ietf.org/rfc/rfc1055.txt

[95] B. rong Chen, K.-K. Muniswamy-Reddy, and M. Welsh, "Ad-hoc multicast routing on resource-limited sensor nodes," in *Proceedings of the International Workshop on Multi-hop Ad-Hoc Networks (REALMAN'06)*, Florence, Italy, May 2006, pp. 87–94.

[96] E. M. Royer and C. E. Perkins, "Multicast operation of the ad-hoc on-demand distance vector routing protocol," in *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, ser. MobiCom '99. New York, NY, USA: ACM, 1999, pp. 207–218.

[97] L. B. Ruiz, J. M. Nogueira, and A. A. F. Loureiro, "Manna: A management architecture for wireless sensor networks," *IEEE Communications Magazine*, vol. 41, no. 2, pp. 116–125, Feb 2003.

[98] A. Ruzzelli, M. O'Grady, G. O'Hare, and R. Tynan, "Merlin: A synergetic integration of mac and routing protocol for distributed sensor networks,"

in *Proceedings of the IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks (SECON'06)*, Reston, VA, USA, Sep 2006, pp. 266–275.

[99] J. A. Sanchez, P. M. Ruiz, and I. Stojmenovic, "Gmr: Geographic multicast routing for wireless sensor networks," in *Proceedings of the IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks (SECON'06)*, Reston, VA, USA, Sep 2006, pp. 20–29.

[100] ——, "Energy efficient geographic multicast routing for sensor and actuator networks," *Computer Communications*, vol. 30, no. 13, pp. 2519–2531, Sep 2007.

[101] ScatterWeb: Modular Sensor Board 430. [Online]. Available: http://www.mi.fu-berlin.de/inf/groups/ag-tech/projects/Z_Finished_Projects/ScatterWeb/modules/mod_MSB-430.html

[102] ScatterWeb^2 Operating System - Freie Universität Berlin & ScatterWeb GmbH. [Online]. Available: http://www.mi.fu-berlin.de/inf/groups/ag-tech/projects/Z_Finished_Projects/ScatterWeb/index.html

[103] C. Schluep, "Porting contiki to the btnode sensor node platform," Bachelor's Thesis, University of Magdeburg, 2009.

[104] Sensirion Sensor Company: SHT11 Digital Humidity and Temperature Sensor. [Online]. Available: http://www.sensirion.com/en/products/humidity-temperature/humidity-sensor-sht11/

[105] Seventh Framework Programme FP7 - Information and Communication Technologies, "Wireless Sensor Networks Testbed Project (WISEBED)," FP7 Project 2008-2011. [Online]. Available: http://www.wisebed.eu

[106] A. Sheth, B. Shucker, and R. Han, "Vlm2: A very lightweight mobile multicast system for wireless sensor networks," in *Proceedings of IEEE Wireless Communications and Networking Conference (WCNC'03)*, New Orleans, LA, USA, Mar 2003, pp. 1936–1941.

[107] W. Si and C. Li, "Rmac: A reliable multicast mac protocol for wireless ad hoc networks," in *Proceedings of th International Conference on Parallel Processing (ICPP'04)*, Montreal, Canada, Aug 2004, pp. 494–501.

[108] J. S. Silva, T. Camilo, P. Pinto, R. Ruivo, A. Rodrigues, F. Gaudêncio, and F. Boavida, "Multicast and ip multicast support in wireless sensor networks," *Journal of Networks*, vol. 2, no. 3, pp. 19–26, Mar 2008.

[109] J. S. Silva, T. Camilo, A. Rodrigues, M. Silva, F. Gaudencio, and F. Boavida, "Multicast in wireless sensor networks - the next step," in *Proceedings of the International Symposium on Wireless Pervasive Computing (ISWPC'07)*, San Juan, Puerto Rico, Feb 2007, pp. 185–190.

[110] SQLite: A Lightweight File-Based Database. [Online]. Available: http://www.sqlite.org

[111] V. Srinivas and L. Ruan, "An efficient reliable multicast protocol for 802.11-based wireless lans," in *Proceedings of the IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM'09)*, Kos, Greece, Jun 2009.

[112] T. Staub, "Development, testing, deployment and operation of wireless mesh networks," dissertation, Universität Bern, 2011.

[113] T. Staub, S. Morgenthaler, D. Balsiger, P. K. Goode, and T. Braun, "ADAM: Administration and Deployment of Adhoc Mesh Networks," in *Proceedings of the IEEE Workshop on Hot Topics in Mesh Networking (HotMESH'11)*, Lucca, Italy, June 2011, pp. 1–6.

[114] P. Suarez, C. Renmarker, T. Voigt, and A. Dunkels, "Increasing ZigBee network lifetime with X-MAC." ACM Workshop on Real-World Wireless Sensor Network (REALWSN), Glasgow, Scotland, November 2008, pp. 13–18.

[115] M.-T. Sun, L. Huang, A. Arora, and T.-H. Lai, "Reliable mac layer multicast in ieee 802.11 wireless networks," in *Proceedings of the International Conference on Parallel Processing*, aug. 2002, pp. 527–536.

[116] K. Sundaresan, V. Anantharaman, H.-Y. Hsieh, and A. Sivakumar, "Atp: a reliable transport protocol for ad hoc networks," *IEEE Transactions on Mobile Computing*, vol. 4, no. 6, pp. 588–603, nov.-dec. 2005.

[117] Texas Instruments: 16-Bit Ultra-Low Power MSP430 Microcontrollers. [Online]. Available: http://www.ti.com/lsds/ti/microcontroller/16-bit_msp430/overview.page

[118] Texas Instruments CC1020: Single-Chip FSK/OOK CMOS RF Transceiver. [Online]. Available: http://www.ti.com/product/cc1020

[119] Texas Instruments CC1020: Single Chip Ultra Low Power RF Transceiver for 315/433/868/915 MHz SRD Band . [Online]. Available: http://www.ti.com/product/cc1000

[120] Texas Instruments CC2420: Single-Chip 2.4 GHz IEEE 802.15.4 / ZigBee-ready Radio RF Transceiver. [Online]. Available: http://www.ti.com/product/cc2420

[121] Tutornet University of Southern California, "A Tiered Wireless Sensor Network Testbed." [Online]. Available: http://enl.usc.edu/projects/tutornet

[122] G. Wagenknecht, M. Anwander, and T. Braun, "Hop-to-Hop Reliability in IP-based Wireless Sensor Networks - a Cross-Layer Approach," in *Proceedings of the International Conference on Wired/Wireless Internet Communications (WWIC'09)*, Enschede, The Netherlands, May 2009.

[123] ——, "Marwis: A management platform for heterogeneous wireless sensor networks," *Ercim News*, vol. 5031/2008, no. 6, January 2009.

[124] ——, "Demo: MARWIS - a Management Architecture for Heterogeneous Wireless Sensor Networks," in *Proceedings of the IEEE 36th Conference on Local Computer Networks (LCN'11), Demo Session*, Bonn, Germany, October 2011. [Online]. Available: http://www.ieeelcn.org/prior/LCN36/lcn36demos.html

[125] ——, "Performance Evaluation of Reliable Overlay Multicast in Wireless Sensor Networks," in *Proceedings of the International Conference on Wired/Wireless Internet Communications (WWIC'12)*, Santorini, Greece, June 2012, pp. 75–78.

[126] ——, "SNOMC: An Overlay Multicast Protocol for Wireless Sensor Networks," in *Proceedings of the Annual Conference on Wireless On-demand Network Systems and Services (WONS'12)*, Courmayeur, Italy, January 2012, pp. 75–78.

[127] G. Wagenknecht, M. Anwander, T. Braun, T. Staub, J. Matheka, and S. Morgenthaler, "MARWIS: A Management Architecture for Heterogeneous Wireless Sensor Networks," in *Proceedings of the International Conference on Wired/Wireless Internet Communications (WWIC'08)*, Tampere, Finland, May 2008, pp. 177–188.

[128] G. Wagenknecht, M. Anwander, M. Brogle, and T. Braun, "Reliable Multicast in Wireless Sensor Networks," in *Proceedings of the 7. GI/ITG KuVS Fachgespraech Drahtlose Sensornetze (FGSN'08)*, Berlin, Germany, September 2008.

[129] C.-Y. Wan, A. T. Campbell, and L. Krishnamurthy, "Psfq: A reliable transport protocol for wireless sensor networks," in *Proceedings of the ACM international Workshop on Wireless Sensor Networks and Applications (WSNA'02)*, Atlanta, GA, USA, Sep 2002, pp. 1–11.

[130] G. Werner-Allen, P. Swieskowski, and M. Welsh, "MoteLab: a Wireless Sensor Network Testbed." ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN), Los Angeles, USA, April 2005, pp. 483–488.

[131] J. Wu, P. Havinga, S. Dulman, and T. Nieberg, "Eyes Source Routing Protocol for Wireless Sensor Networks." European Conference on Wireless Sensor Networks (EWSN'04), Berlin, Germany, January 2004.

[132] W. Xiangli, L. Layuan, and W. Wenbo, "An energy-efficiency multicast routing algorithm in wireless sensor networks," in *Proceeding of the ISECS International Colloquium on Computing, Communication, Control, and Management (CCCM'08)*, Guangzhou City, China, Aug 2008, pp. 572–576.

[133] W. Ye, J. Heidemann, and D. Estrin, "An Energy Efficient MAC Protocol for Wireless Sensor Networks." IEEE International Conference on Computer Communications (INFOCOM), New York, USA, June 2002, pp. 1567–1576.

[134] W. Zhang, X. Jia, C. Huang, and Y. Yang, "Energy-aware location-aided multicast routing in sensor networks," in *Proceedings of International Conference on Wireless Communications, Networking and Mobile Computing*, vol. 2, Sep 2005, pp. 901–904.

List of Publications

Refereed Papers (Journals, Conferences, Workshops)

- Gerald Wagenknecht, Markus Anwander, Torsten Braun: Performance Evaluation of Reliable Overlay Multicast in Wireless Sensor Networks, *10th International Conference on Wired/Wireless Internet Communications (WWIC'12)*, Santorini, Greece, June 6 - 8, 2012
- Philipp Hurni, Markus Anwander, Gerald Wagenknecht, Thomas Staub, Torsten Braun: TARWIS - A testbed management architecture for wireless sensor network testbeds, *IEEE/IFIP Network Operations and Management Symposium (NOMS'12)*, Maui, Hawaii, USA, April 2012
- Gerald Wagenknecht, Markus Anwander, Torsten Braun: SNOMC: An Overlay Multicast Protocol for Wireless Sensor Networks, *9th Annual Conference on Wireless On-demand Network Systems and Services (WONS'12)*, Courmayeur, Italy, January 2012
- Geoff Coulson, Barry Porter, Ioannis Chatzigiannakis, Christos Koninis, Stefan Fischer, Dennis Pfisterer, Daniel Bimschas, Torsten Braun, Philipp Hurni, Markus Anwander, Gerald Wagenknecht, Sandor Fekete, Alexander Kröller, Tobias Baumgartner: Flexible Experimentation in Wireless Sensor Networks, *Communications of the ACM*, Vol. 55, Nr. 1, January 2012
- Gerald Wagenknecht and Markus Anwander and Torsten Braun: Demo: MARWIS - a Management Architecture for Heterogeneous Wireless Sensor Networks, *IEEE Conference on Local Computer Networks (LCN'11), Demo Session*, Bonn, Germany, October 2011
- P. Hurni, M. Anwander, G. Wagenknecht, T. Staub, T. Braun: TARWIS - A Testbed Management Architecture for Wireless Sensor Network Testbeds, *International Conference on Network and Service Management (CNSM'11)*, Paris, France, October 2011
- Markus Anwander, Gerald Wagenknecht, Torsten Braun, Kirsten Dolfus: BEAM: A Burst-Aware Energy-Efficient Adaptive MAC Protocol for Wireless Sensor Networks, *Seventh International Conference on Networked Sensing Systems*, Kassel (INSS'10), Germany, June 2010,
- Philipp Hurni, Gerald Wagenknecht, Markus Anwander, Torsten Braun: A Testbed Management Architecture for Wireless Sensor Network Testbeds

(TARWIS), *7th European Conference on Wireless Sensor Networks (EWSN'10)*, Coimbra, Portugal, February 2010

- Gerald Wagenknecht, Markus Anwander, Torsten Braun: Hop-to-Hop Reliability in IP-based Wireless Sensor Networks - a Cross-Layer Approach, *International Conference on Wired/Wireless Internet Communications 2009 (WWIC'09)*, Enschede, The Netherlands, May 2009
- P. Hurni, T. Staub, G. Wagenknecht, M. Anwander, T. Braun: A Secure Remote Authentication, Operation and Management Infrastructure for Distributed Wireless Sensor Network Testbeds, *First Workshop on Global Sensor Networks (GSN'09)*, co-located with KiVS, Kassel, Germany, March 2009
- Gerald Wagenknecht, Markus Anwander, Torsten Braun: MARWIS: A Management Platform for Heterogeneous Wireless Sensor Networks, *Ercim News*, Vol. 5031/2008, Nr. 76, January 2009,
- Gerald Wagenknecht, Markus Anwander, Marc Brogle, Torsten Braun: Reliable Multicast in Wireless Sensor Networks, *7. GI/ITG KuVS Fachgespräch Drahtlose Sensornetze (FGSN'08)*, Berlin, Germany, September 2008
- Markus Anwander, Gerald Wagenknecht, Torsten Braun: Management of Wireless Sensor Networks using TCP/IP, *International Workshop on Sensor Network Engineering (IWSNE'08) at the 4th IEEE/ACM International Conference on Distributed Computing in Sensor Systems*, Santorini Island, Greece, June 2008
- Gerald Wagenknecht, Markus Anwander, Torsten Braun, Thomas Staub, James Matheka, Simon Morgenthaler: MARWIS: A Management Architecture for Heterogeneous Wireless Sensor Networks, *6th International Conference on Wired/Wireless Internet Communications (WWIC'08)*, Tampere, Finland, May 2008,
- Markus Anwander, Gerald Wagenknecht, Thomas Staub, Torsten Braun: Management of Heterogenous Wireless Sensor Networks, *6. GI/ITG KuVS Fachgespräch Drahtlose Sensornetze (FGSN'07)*, Aachen, Germany, July 2007

Unrefereed Papers (Technical Reports, Project Deliverables)

- Geoff Coulson, Gerald Wagenknecht, Markus Anwander, et al.: Report on the Integration of the Software Infrastructure, *WISEBED Deliverable D2.3*, June 2010
- M. Brogle, S. Serbu, D. Milic, M. Anwander, P. Hurni, C. Spielvogel, C. Fautsch, D. Harmanci, L. Charles, H. Sturzrehm, G.Wagenknecht, T. Braun, T. Staub, C. Latze, R. Standtke: *BeNeFri Universities Summer School on Dependable Systems*, Schloss Münchenwiler, Switzerland, September 2009, IAM-09-006

- M. Brogle, D. Milic, M. Anwander, G. Wagenknecht, M. Waelchli, T. Braun, R. Kummer, M. Wulff, R. Standtke, H. Sturzrehm, E. Riviere, P. Felber, S. Krenn, C. Ehret, C. Latze, P. Hurni, and T. Staub: *BeNeFri Universities Summer School on Dependable Systems*, Quarten, Switzerland, November 2008, IAM-08-003
- Torsten Braun, Ulrich Ultes-Nitsche, Marc Brogle, Dragan Milic, Patrick Lauer, Thomas Staub, Gerald Wagenknecht, Markus Anwander, Markus Waelchli, Markus Wulff, Carolin Latze, Michael Hayoz, Christoph Ehret, Thierry Nicola: *RVS Retreat 2007* at Quarten, December 2007, IAM-07-004
- Markus Anwander, Gerald Wagenknecht, Torsten Braun: Sensor Node Platform and Middleware for Management of Wireless Sensor Networks, June 2007, IAM-07-003
- Markus Anwander, Gerald Wagenknecht, Torsten Braun: Energy-efficient Management of Heterogeneous Wireless Sensor Networks, April 2007, IAM-07-002

Curriculum Vitae

Personal Details

Name	Gerald Wagenknecht
Date of Birth	January 2, 1978
Address	Waldstätterstrasse 12
	CH-3014 Bern, Switzerland
Hometown	Görlitz, Germany
Nationality	German

Education

2007-2013	PhD Student & Researcher in Computer Science, Computer Networks and Distributed Systems Group, University of Bern, Switzerland
2005	Master of Science in Computer Science (Diplom-Informatiker), Technical University of Cottbus, Germany
1997-2005	Study of Computer Science at the Technical University of Cottbus, minor fields in Economics and Quality Management
1992-1997	Mathematisch-Naturwissenschaftliches Gymnasium Görlitz, Germany, Emphasis on Physics and History

Erklärung

gemäss Art. 28 Abs. 2 RSL 05

Name/Vorname: Wagenknecht, Gerald

Matrikelnummer: 07-119-969

Studiengang: Informatik

Bachelor ☐ Master ☐ Dissertation ☒

Titel der Arbeit: Energy-efficient Management of Heterogeneous Wireless Sensor Networks

LeiterIn der Arbeit: Prof. Dr. Torsten Braun

Ich erkläre hiermit, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen benutzt habe. Alle Stellen, die wörtlich oder sinngemäss aus Quellen entnommen wurden, habe ich als solche gekennzeichnet. Mir ist bekannt, dass andernfalls der Senat gemäss Artikel 36 Absatz 1 Buchstabe o des Gesetztes vom 5. September 1996 über die Universität zum Entzug des auf Grund dieser Arbeit verliehenen Titels berechtigt ist.

Bern, 26.03.2013

Ort/Datum

Unterschrift