# Secure Multi-Purpose Wireless Sensor Networks

**Sichere, drahtlose Mehrzweck-Sensornetzwerke**

Vom Fachbereich Informatik der Technischen Universität Darmstadt genehmigte Dissertation zur Erlangung des akademischen Grades Doktor-Ingenieur (Dr.-Ing.)

von Dipl.-Inform. Daniel Jacobi aus Wiesbaden

Tag der Einreichung: 03.02.2016, Tag der Prüfung: 22.03.2016
Darmstadt 2016 — D 17

1. Gutachten: Professor Alejandro Buchmann, Ph.D.
2. Gutachten: Professor Dr. Marc Fischlin

TECHNISCHE UNIVERSITÄT DARMSTADT

DVS

Für
Adrian, Maja & Sonja

# Abstract

Wireless sensor networks (WSNs) were made possible around the late 1990s by industry scale availability of small and energy efficient microcontrollers and radio interfaces. Application areas for WSNs range from agriculture to health care and emergency response scenarios. Depending on the scenario a sensor network can span from some rooms to an area of several square miles in size and so the number of sensor nodes can vary from a fistful of nodes to hundreds or thousands. Sensor nodes are composed from a set of building blocks: processing, communication, sensing/actuating and a power supply. The power supply is usually a battery pack. Especially these limited energy resources make it tremendously important to save resources to achieve a long lifetime.

Most WSNs are planned and developed to satisfy only one application, and they are controlled by a single user. But, with the Internet of Things approaching, more and more sensor networks will be used for multiple tasks simultaneously and are reaching larger sizes. As sensor networks grow it becomes mandatory to localize traffic, both for energy conservation as well as security. Additionally, the broadcast medium of the wireless channel of WSNs allows an adversary all sorts of attacks, like eavesdropping, replaying messages, and denial of service attacks. In large or unattended networks it is even possible to physically attack the hardware of a sensor node to gain access to its firmware and possibly cryptographic keys.

In this work we propose the Scopes Framework and the security enhancement SecScopes. The Scopes Framework introduces dynamic partitioning of a WSN with support for multiple in-network tasks. SecScopes enables secure access control, key exchange and communication.

The partitioning is done by a scoping mechanism which allows the dynamic definition of subsets of sensor nodes. The Scopes Framework supports in-network tasks by managing network connections for each task, and allowing the selection of efficient routing algorithms. To allow access control on a partition of the network we introduce attribute-based encryption in sensor networks. Secure key exchange is also based on this encryption scheme. To secure communication more efficient symmetric cryptography is employed.

With the Scopes Framework we provide a modular and flexible architecture that can be adjusted to the needs of different scenarios. We present a detailed evaluation of the performance of the framework and compare and discuss the results for the different stages of the framework. The results of the evaluation show the general feasibility of the approach, in spite of the adverse resource constraints.

# Zusammenfassung

Drahtlose Sensornetzwerke (WSNs) sind Ende der 1990er Jahre durch die industrielle Verfügbarkeit von kleinen, energie-effizienten Mikrocontrollern und Funkschnittstellen möglich geworden. Einsatzgebiete für WSNs reichen von der Landwirtschaft bis in den medizinischen oder Notfall-Bereich. Abhängig vom Einsatzgebiet kann ein Sensornetzwerk über wenige Räume bis hin zu Gebieten von mehreren Quadratkilometern reichen und so kann auch die Anzahl von Sensorknoten von einer Handvoll bis zu mehreren hundert oder sogar tausend reichen. Sensorknoten setzen sich aus folgenden Blöcken zusammen: Ablaufsteuerung, Kommunikation, Sensorik/Aktorik und Stromversorgung. Die Stromversorgung ist in der Regel Batterie-basiert. Diese geringen und limitierten Energiereserven machen es sehr wichtig Ressourcen zu sparen, um eine lange Laufzeit zu erreichen.

Meistens werden WSNs für eine spezielle Anwendung entwickelt und sie werden von nur einem Benutzer kontrolliert. Aber, durch das Internet der Dinge, werden immer mehr Sensornetzwerke für mehrere, gleichzeitige Aufgaben genutzt werden und sie werden größer. Mit dem Wachsen von Sensornetzwerken wird es unumgänglich Netzwerkpakete möglichs lokal zu halten, sowohl um Ressourcen zu schonen, als auch zur Erhöhung der Sicherheit im Netzwerk. Zusätzlich erlaubt das Broadcast-Medium der Funkschnittstelle eines WSNs einem Angreifer viele Angriffsmöglichkeiten, z.B. Mithören von Verbindungen, Nachrichten wiedereinspielen oder Denial of Service-Attacken. In großen oder abgelegenen Sensornetzwerken kann es zudem zu physischen Attacken auf die Sensorknoten kommen, um Zugriff auf die Firmware oder sogar kryptographische Schlüssel zu bekommen.

In dieser Arbeit stellen wir das Scopes Framework und die Sicherheitserweiterung SecScopes vor. Das Scopes Framework erlaubt die dynamische Partitionierung eines WSNs mit Unterstützung für mehrere Tasks im Netzwerk. SecScopes erlaubt Zugriffskontrolle auf die einzelnen Partitionen des Netzwerks, sicheren Schlüsselaustausch und sichere Kommunikation.

Die Partitionierung des Netzwerks wird durch einen Scoping-Mechanismus erreicht, der eine dynamische Definition von Subsets von Sensorknoten erlaubt. Das Scopes Framework unterstützt Tasks innerhalb des Sensornetzwerks durch die Bereitstellung von Netzwerkverbindungen pro Task und der Möglichkeit ein effizientes Routing auszuwählen. Die Zugriffskontrolle auf eine Partition des Netzwerks wird durch attribut-basierte Verschlüsselung erreicht. Der sichere Schlüsselaustausch basiert ebenfalls darauf. Zur sicheren Kommunikation wird effizientere symmetrische Verschlüsselung eingesetzt.

Mit dem Scopes Framework stellen wir ein modulares und flexibles System zur Verfügung, das an verschiedenen Einsatzszenarien angepasst werden kann. Wir zeigen eben-

falls eine detaillierte Evaluation der Leistungsfähigkeit des Frameworks und vergleichen und diskutieren die Ergebnisse der verschiedenen Ausbaustufen. Das Ergebnis der Evaluation zeigt die Machbarkeit des Ansatzes, trotz der stark eingeschränkten Ressourcen.

# Preface

## Acknowledgments

As I started with this PhD project, I was not aware of what was coming. There were great people, great events and conferences, a stay in Virginia, and very interesting things to work on. But also frustration over long hidden bugs or interferences on demos. And, many hours of writing and evaluating what the sensor network spit out. In the end, it was big fun and hard work. Here I want to express my gratitude to all the people that made this happen.

First and foremost my advisor, Alejandro Buchmann, for making all this possible. For the freedom to pursue our research ideas, even if they do not fit in the end. The constant support and valuable advice, even in the last years where I was not working at DVS. And his last push to get the work done. Also Marc Fischlin, who guided me through the shallow waters of the math behind cryptography.

All the people at DVS who made working here an awesome experience. Especially Pablo Guerrero for the mental and practical support, and his friendship through all these years. Ilia Petrov for guidance and good questions. My officemates Patric Kabus, Arthur Herzog and Christian Seeger for the insightful discussions. My Colleagues at the GKmM and CASED programs for the deep and interesting insights in very different fields of research. And Stefan Weber for introducing me to the world of attribute-based encryption.

My current employer Zühlke Engineering where I found an environment of constant support and many great people to learn from. Here Gernot Trautman supported my endeavor of completing this work by providing time at different stages.

My family and friends who made me who I am and supported me during this time. Especially, Sonja, my love, who got the whole story started and supported me emotionally and by keeping my back free. Also Maja and Adrian as the two little sunshines of my life. Last, but not least, my parents, Sylvia and Edgar, and my sister Catherine for the support in so many ways over all the time.

## Publications and Projects

This thesis is partly based on previous publications. Chapter 2 is based on [55], chapter 3 on [54] and chapter 4 on [54, 56, 64]. A number of publications are not incorporated into this work [6, 40, 49, 57, 66].

The research performed for this dissertation has been funded by the following projects. The DFG PhD Research Training Program GRK Nr. 1362, *Cooperative, Adaptive and Responsive Monitoring in Mixed Mode Environments (GKmM)* and the hessian LOEWE *Center for Advanced Security Research Darmstadt (CASED)*.

# Contents

# List of Tables

# List of Figures

# Listings

# List of Abbreviations and Symbols

| | |
|---|---|
| $\mathbb{A}$ | Access policy of CP-ABE algorithm |
| API | Application Programmer Interface |
| CT | Cipher text; encrypted plain text or data |
| FLD | Flooding; Routing algorithm used as baseline in Scopes Framework evaluation. |
| GBR | Gradient-based routing, used in SOS version of the Scopes Framework. |
| LCM | Life-Cycle Manager task |
| MCU | Microcontroller unit |
| MK | Master key of CP-ABE algorithm |
| OTA | Over-the-air, generally used in terms of firmware updates. |
| PK | Public key of CP-ABE algorithm |
| RSSI | Received Signal Strength Indication |
| SCR | Scope Creation Request |
| SEL | see →SelfUR |
| SelfUR | Selective flooding, unicast return. Routing algorithm used in Contiki version of the Scopes Framework. |
| SK | Secret key of CP-ABE algorithm |
| SR | Scope Refresh |
| TTL | Time-to-live |
| VM | Virtual machine |
| WSN | Wireless Sensor Network |

# 1 Introduction

## Contents



The Internet has changed the way of life for most people in the last decade. Information is exchanged via email or instant messaging and it evolved from a tool for experts to a connector of people all over the world. And it is still evolving. One of these evolutions is the integration of more and more devices into the Internet. Today most of the new TV-sets are Internet capable and the *Internet of Things* tries to incorporate most electronic devices in the near future. One of the enabling technologies to access and use even the smallest device are Wireless Sensor Networks (WSNs).

With advances in miniaturization of electronic components and enhanced performance over the last decade, smart electronic devices are penetrating more and more areas of daily life. Application areas for WSNs range from agriculture or logistics to health care and emergency response scenarios. These applications can be split into several (sub-)tasks. Depending on the scenario a sensor network can span from some rooms to an area of several square miles in size and so the number of sensor nodes can vary from a fistful of nodes to hundreds or thousands. Sensor nodes are composed from a set of building blocks: processing, communication capabilities, sensing/actuating and power supply. The large quantities of nodes participating in sensor networks has pushed prices down. These nodes consist of a slow CPU combined with a limited amount of memory. This allows pre-processing or decision taking on the local node before reporting data via a communication link or triggering some kind of actuation. The transceiver is most of the time a low-bandwidth radio link, but also other techniques like infrared are possible. Sensing/actuation depends on the actual application, many sensors or actuators are possible here. The power supply is usually a battery pack. The low cost requirements result in highly constrained resources on a sensor node. This requires suitable and efficient algorithms, optimized for the limited CPU and memory space. Especially the limited energy resources make it tremendously important to save resources to achieve a long lifetime. Transmitting messages requires a significant amount of energy and, therefore, as few messages as possible are sent. Messages should be kept as

short as possible. The range of the transceivers used is typically limited and messages are sent in a multi-hop fashion, i.e. a message is received by a node and then forwarded to its destination, hop by hop.

## 1.1 Motivation

Today's WSNs are planned and developed to satisfy only one application, and they are usually controlled by a single user. In the future more and more sensor networks will be used for multiple tasks simultaneously and are reaching sizes where structuring is necessary for efficient use. This also leads to more heterogeneous networks, where several users have to cooperate in a large network. Even smaller networks benefit from structuring, as just parts of the network can be included in an application and the rest can stay in a power saving mode. With the broad deployment of sensor networks in industry, the need for security has increased and became an important issue. While approaches have been proposed to both problems in isolation, an integrated mechanism for secure structured sensor networks is still missing.

Having multiple users cooperating in a WSN introduces the need for frameworks that support multiple networking trees, meshes or just end-to-end connections. This is only marginally available in current approaches. Also many developments in the area of WSNs are focused on a specific scenario. This makes sense to achieve high efficiency during operation but makes deployment and reuse for other purposes more difficult. Looking back on hardware and performance advances over the last decade of WSNs, it is definitely feasible to push for a more generic design, introducing a higher level of reusability, and sacrificing some resources.

Securing a WSN is a tough challenge. On the one hand the limited resources of the sensor nodes require very efficient security algorithms. On the other hand sensor nodes are mostly deployed in insecure or hostile environments. Additionally the multi-hop communication allows an adversary all sorts of attacks, like eavesdropping, replaying messages, different man-in-the-middle attacks, like injecting, dropping or altering messages, and denial of service attacks by blocking radio channels in specific areas or the whole network. In large or unattended networks it is even possible to physically attack the hardware of a sensor node to gain access to its firmware and cryptographic keys.

The following two examples emphasize the need for structuring a WSN and introducing security in a corresponding framework. The first is an emergency response scenario where reusability and partitioning is a major demand. The second scenario is from the domain of logistics. Here support for multiple users and security are the major issues.

### 1.1.1 Scenario: Emergency Response

After an incident at a chemical plant the equipment is partly destroyed, other parts of the plant are still functional. The statically deployed infrastructure to monitor safety

and control the different systems is also damaged. This infrastructure was deployed as a WSN, so that after a potential incident it can stay functional in areas that were not affected. At the incident site single sensors can still work and monitor potential threats.

Getting the remaining infrastructure back in a working state is a high priority task, so it can be used by emergency response teams, such as, firefighters, robots or quadrocopters, to handle the incident. To acquire an overview of the situation in a short time, and identify areas with, e.g. leaked chemicals that are unsafe for human personnel, the remaining sensors have to be reconnected to the WSN. For this purpose the robots and quadrocopters are able to deploy additional sensor nodes to enhance the radio coverage of the WSN, or allow additional monitoring of critical areas through sensors at specific locations.

With the network reconnected, structuring helps to keep the unaffected areas of the plant in a normal state, and the damaged part is not affected by data exchanged there. Also the network at the incident site can get divided into high and low priority areas where different emergency response teams can work independently from each other. In that situation, new sensing or control tasks are only deployed in areas where they are needed.

### 1.1.2 Scenario: Container Harbor

Back in 2004 Culler et al. described a container harbor scenario [24] that still is challenging today. In a large international container harbor, like Hamburg or Rotterdam, you have different areas where containers can be loaded or unloaded, on ships or trucks. There are storage areas and an office complex, where companies, customs, etc. are located. All these parties want to know where their containers are, what the status of their containers is or what is exactly inside a specific container. In this scenario a sensor node is attached to every container. The nodes can identify whom they belong to, the content of the container and monitor and control its state, e.g. temperature in a cooling container.

To query a set of containers, a company can create a partition based on conditions that the containers must fulfill. Conditions could be defined on type of container (e.g. *all cooling containers*), owner or other grouping criteria. Later on the company can maintain the firmware for monitoring the container, query the logged entries or configure the cooling management.

Using a partition has the advantage of limiting the network traffic only to the areas of the network where the members are located, which favors network security. Data security is an important issue in this scenario. Unauthorized access to a competitor's sensor nodes should be prevented and only properly authorized and authenticated containers should join a partition.

## 1.2 Shortcomings of Wireless Sensor Networks

Starting with a one task network is just natural, but with further progress more tasks will be transferred to commonly used networks. At this point today's networks will fail because they were developed with a single application in mind and lack the ability to service multiple applications and are not reusable for emerging developments. Current operating systems for WSNs support parallel execution of different tasks, but support from the overlaying frameworks is very limited or non-existent.

As sensor networks grow it becomes more important to localize traffic, both for energy conservation as well as security. The lack of structure in today's networks makes it difficult to accommodate multiple users in a sensor network. In a structured WSN each application can get its own overlay network.

The different overlays in a structured WSN introduce a first level of security, as shown for publish / subscribe systems in [33]. They limit the visibility of a task and its data to a subset of the WSN. Only sensor nodes participating in the same overlay or are used to route messages for it can be aware of the overlay's presence. This drastically reduces the number of potential adversary nodes in a large WSN. However, malicious sensor nodes can pretend to be part of the group. To prevent this, access control mechanisms should be introduced to the group creation process. This feature is not available in current WSN frameworks.

## 1.3 Secure Multi-Purpose Wireless Sensor Networks

Statically structuring a sensor network is not enough. It is important to be able to define groups in a way that suits the needs of an application without excessive resource consumption. In the Scopes Framework this is achieved by a binary representation of a boolean expression to declaratively define a scope to structure the network and refresh messages for a scope with configurable refresh intervals.

Besides structuring of the WSN, the support for different scenarios and multiple concurrent tasks is a highly desirable property. Supporting the basic functionality of executing multiple concurrent applications is handled by the operating system. The Framework extends this support to manage network connections for each application or task, and allows the selection of different routing algorithms. Being able to choose the optimal routing for a scenario is important for efficiency. Therefore, the Scopes Framework is modular. Routing algorithms, the language or functions to determine scope memberships, and of course the tasks are adjustable to the specific needs of a scenario.

Security is one of the top priorities in commercial deployments of WSNs. Therefore, the Scopes Framework introduces secure group joining and communication procedures. This ensures the confidentiality of exchanged data and prevents malicious nodes from unauthorized joining an arbitrary group.

This thesis presents the Scopes Framework and the extensions that enable Secure Multi-Purpose Wireless Sensor Networks. The framework presented in this thesis makes the following contributions:

Scoping. This is the major building block of the framework and enables the dynamic definition of subsets of sensor nodes in the network. Therefore, a declarative language was defined to easily create and delete groups. Scoping also provides the ability to hierarchically structure groups and subgroups, so called scopes and subscopes, of nodes. It also enables further security measures, as messages are kept in the specific area of a scope and the creation mechanism offers a proper place for an access control mechanism.

Multitasking and Reusability. Multitasking is a major aspects for a multi purpose WSN and it is also an integral part of the Scopes Framework. It allows the management of tasks on a sensor node and enables also multiple users in one WSN. To use the Scopes Framework in many different scenarios and to be able to introduce new applications in an existing WSN, design modularity and reusability received high priority.

Access Control. To control who can access a scope is a basic security requirement. Using an Attribute-Based Encryption (ABE) scheme, access control can be integrated in the framework in a transparent way, restricting the access to a scope only to authorized devices. This is provided by our proposed security architecture that provides security in the process of creating, maintaining and deleting scopes.

Secure Key Exchange and Communication. Secure communication is, besides access control, the second basic security requirement. To be able to deploy a WSN in a business environment, as discussed in the scenario in section 1.1.2, confidentiality of the transmitted data is vital. In our proposed architecture we use efficient symmetric cryptography to secure the communication inside a scope. Therefore, a secure way to exchange symmetric keys is mandatory. This is an integral part of the ABE scheme also used for access control.

Implementation. The reference implementation of the *Scopes Framework* has undergone extensive development and testing and is a robust system that can be reused in future projects.

## 1.4 Organization of this Thesis

The remainder of this dissertation is organized as follows:

Chapter 2. In the *Scopes Framework* chapter we elaborate on the details of our framework design and explain the different modules and phases of the architecture and its runtime behavior.

Chapter 3. The *Securing the Scopes Framework* chapter deals with security problems in WSNs and solutions introduced in Scopes. It also discusses the influence of security on the architecture of the Scopes Framework introduced in chapter 2.

Chapter 4. After completing the architectural description, we will evaluate the Scopes Framework. This includes the different versions of the Scopes Framework, the security algorithms and libraries used.

Chapter 5. In this chapter we examine the work related to this thesis from different areas. This section is divided in two major parts. The first reviews the work related to scoping while the second part covers related work in the security domain.

Chapter 6. Last we present conclusions and point out further developments.

# 2 The Scopes Framework

## Contents



The basic concepts of scopes was originally conceived to provide structure in loosely coupled publish/subscribe broker networks [32, 33] in the Rebeca Publish/Subscribe system, see also section 5.1.2. These were later proposed for sensor networks by J. Steffan in [123, 124] and the first implementation was done in [53] for a simulator to prove feasibility.

Before we start with details of the Scopes Framework we will take a short step back and show some background on wireless sensor networks and define our understanding of common terms.

## 2.1 Prerequisites

Wireless sensor networks were created around the late 1990s. They were made possible by industry scale availability of small and energy efficient microcontrollers (MCUs)

and especially radio interfaces, the two most important building blocks of sensor networks. Over the last decade several hardware platforms were built. The early sensor nodes showed the feasibility around the year 2000. The first widely used platform that also was commercially available was the Mica2 [23], from 2002 and after that around 2004 Telos, TelosB/Tmote and micaZ [85] were developed. Developments from the last years went in different directions, like the Imote2, the SunSPOTS [94] or Econotags [107]. They are not built around the paradigm of energy efficiency like previous sensor nodes, but with much more performance, e.g. the microcontrollers used here have common XScale or ARM cores. Some sensor network platforms went back to the roots and tried to extend the still very popular Tmote platform with more powerful microcontrollers from the same MCU families as before, like the Zolteria Z1 [154]. The current versions of microcontrollers have more than doubled the RAM and flash memory, and added new hardware building blocks like AES encryption support with roughly the same power consumption. Other Platforms try to find an equilibrium between the powerful Imote2 class and the energy efficient Tmote class of sensor nodes; examples are the WASPmotes [120]. Besides those, many custom built sensor network platforms were used in different scenarios.

Wireless Sensor Networks were at first deployed in situations where they have to monitor data over a large area, e.g. environmental. There are numerous examples for this type of usage, like the Great Duck Island experiment [77], in California's coastal Redwood forests [127] or Glascweb to monitor glaciers in Norway [81].

The scenarios evolved and the next generation of sensor networks was able to not just sense, but also to actuate based on the sensed data. Sometimes with a human-in-the-loop, sometimes autonomously. Application areas are, for example, building automation or inventory control, like the chemical drum storage management [63].

The radio interface is quite stable since the introduction of the IEEE 802.15.4 [84] compliant radio, working on the same frequencies as wireless LAN or Bluetooth. But there are also versions for different frequencies. More powerful nodes may use Bluetooth or even wireless LAN as connection to the outside world, but for the majority of power efficient platforms the IEEE 802.15.4 interface is standard.

To ease the development for the different hardware platforms, small operating systems with basic functionality were created. One of the first and still popular ones is TinyOS [50]. It uses its own C dialect called nesC which had some reliability issues as we started with this work. Therefore, we used the SOS operating system [44] that was also popular at that time and was one of the first operating systems that supported over-the-air deployment. We moved the Scopes Framework to Contiki [27] after having stability issues with SOS. Contiki started as research project, and has a similar large and active community as TinyOS.

### 2.1.1 Sensors, Nodes, Platforms

This section summarizes common terms in wireless sensor networks and presents a consistent definition of them.

**Definition 2.1.** A *Sensor Node* (short *node*) is a small independent piece of hardware. In its simplest form it has an MCU, a radio device and some kind of sensor system. Additionally, sensor nodes can be equipped with flash storage or actuation devices, a simple example is a LED. They are mostly battery powered, but can also be attached to energy harvesting equipment [116] or a constant power supply. Sensor nodes can carry multiple sensors and can also use different wireless connections. Sensor nodes can be mobile or stationary.

**Definition 2.2.** A *Base Station* can be compared to a gateway in a common computer network, as it is used to communicate with the outside world of the sensor network, like the Internet or another network. Mostly there is only one base station in a sensor network, but it is also possible to use multiple stations. A base station can be a common wireless sensor node, but most of the time it is a more powerful node, or even a PC with constant power supply.

**Definition 2.3.** A *Wireless Sensor Network* (*sensor network*, WSN) is a wirelessly connected network of sensor nodes. If the sensor network is only built out of one hardware platform (maybe except the network's base station) it is called a *homogeneous* Wireless Sensor Network, if different platforms are used it is called a *heterogeneous* Wireless Sensor Network. A sensor network can contain static and mobile nodes and it is assumed that new nodes join or existing nodes leave the network at all times.

A *Wireless Sensor Actor Network* (WSAN) is a wireless sensor network, where sensor nodes not only can sense, but also actuate. The terms wireless sensor network and wireless sensor actor network, are sometimes used as synonyms, sometimes not. Here they are used as synonyms, as the Scopes Framework is explicitly designed to support not only sensing nodes, but also nodes are capable of actuatuion.

**Definition 2.4.** A *Wireless Sensor Network Application* (WSN Application) describes the overall scenario and specific goals that the application is meant to achieve. To reach the goals an application may be divided into smaller, separate *Tasks*, with each task providing a specific functionality to the overall application.

**Definition 2.5.** *Resources* in wireless sensor networks are limited goods, that may or may not be replenished and most of the time are meant to be scarce. The most prominent resources are energy, computation capacity and (wireless) network bandwidth. Energy, most of the time supplied by batteries, is the best example for a resource that is constantly drained and will not be replenished by itself. Network bandwidth and computational capacity, are goods that can run dry, during heavy duty periods, but are available again afterwards. Other resources are, e.g. storage space, RAM and code memory.

## 2.2 Architecture

The *Scopes Framework* is a modular framework that enables multitasking by bringing a logical structure to the network. With the Framework, nodes are organized into groups, which we call a scope. It is the central abstraction provided by the framework to applications. A scope can be defined by means of a logical expression, which must be satisfied by a node to become its member. Once a scope is created, it continues to exist until it is explicitly removed, even as nodes fail or if they temporarily leave and rejoin. The framework takes care of reliably maintaining the scope membership.

Scopes provide a bidirectional communication channel between the member nodes and their sink node. The framework can operate with multiple routing algorithms. Any routing algorithm allowing bidirectional traffic and multiple concurrent routes can be used. Each top-level scope can be dynamically configured to use any routing algorithm; subscopes – scopes that are embedded in another scope – inherit it implicitly. Separating the definition of a scope from the underlying routing algorithm potentially enables scopes to span across different node platforms, and allows other optimizations through the choice of diverse routing algorithms.

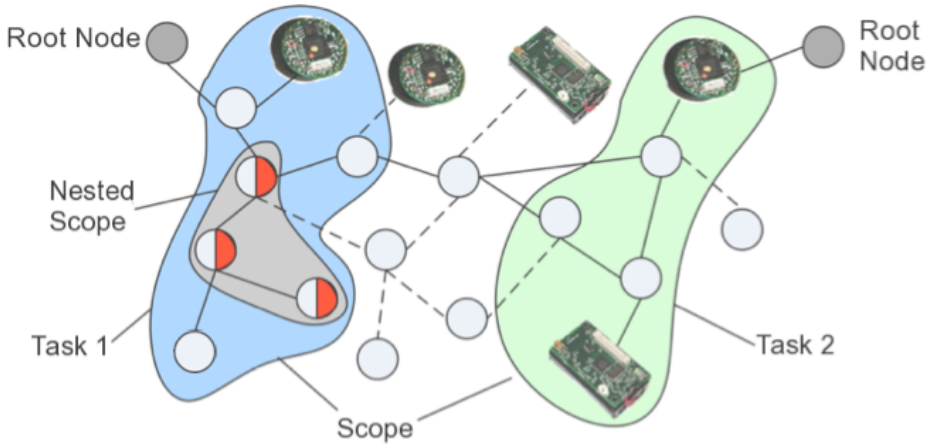



**Figure 2.1:** A Wireless Sensor Network with active Scopes.

### 2.2.1 Terms and Definitions

Next we will define some often used terms to ensure the proper definition and understanding of them.

#### 2.2.1.1 Scopes

**Definition 2.6.** A *Scope Definition* is a set of properties that a sensor node has to match to be a member of a scope. This definition is mostly a boolean expression built from key-value pairs and a set of operators. But the definition is not limited to boolean

expressions. By replacing the standard scope membership module by a custom one, other kinds of definitions are possible, as long as they are representable by a byte-string. A scope definition is transformed into a transport representation, a byte-string, to be distributed in a WSN.

**Definition 2.7.** A *Scope* is a subset of nodes of a wireless sensor network and is defined via the scope definition. A node can participate in a scope, iff it matches the scope's definition. These nodes are called *scope members* and can be potentially spread around the network. Scopes can be nested, this means that scopes can be created on top of existing scopes, these are called *nested scope*.

**Definition 2.8.** As scopes can build hierarchies, a scope's parent scope is called its *Super Scope* and there can only be one of it. Child scopes are called *Subscopes* and can be more than one.

**Definition 2.9.** The *World Scope* is an implicitly defined scope. Every node in the sensor network is member of the World Scope and does not have to evaluate a scope definition. The World Scope is only used to create other scopes inside. Scopes on top of the World Scope have a special meaning. They are called *top-level scopes* and they are the first scopes in the hierarchy that are used by applications.

#### 2.2.1.2 Node Roles

**Definition 2.10.** The scope's *Root Node* is the sink of a scope. This sink receives all messages from member nodes and sends messages to the members of a scope. It is responsible of managing the scope. This means that it is responsible to keep the scope alive and delete it, if requested.

**Definition 2.11.** A scope *Member Node* is a node that matches the definition of a scope and so participates in the scope and can send/receive messages. It is also used for routing purposes, as stated in the next definition. There can be several of them.

**Definition 2.12.** A scope *Intermediate Node* is a node that is on the routing path between a member node and the root node, but not a member of the scope. It is used to forward messages from the root node to member nodes or vice versa for the multi-hop communication of a WSN. This requires that all information needed for routing decisions have to be available in the header of the routing algorithm, as the intermediate node may not know how to parse to data area of the message.

### 2.2.2 Scoping

A *scope* is a group of sensor nodes, selected by specified properties. A Scope can be used several times for different queries and tasks. It is created by the so called *Scope Root Node* (see definition 2.10). The Scope Root Node is responsible for distributing

the *Scope Definition* (see definition 2.6) to all potential members of the Scope and it has to take care of the maintenance. The lifetime of a scope is limited by two factors. On the one hand it is the lifetime of the root node itself, when the root node leaves the sensor network for any reason, the node who manages the scope is removed and in consequence the scope is removed after a period of time. On the other hand the root node can send a message that deletes the Scope. This operation is only allowed to the root node.

Creating hierarchies is one of the major benefits of the scoping concept and an integral part of the Scopes Framework presented here. Two notions of scopes exist in the Framework: top-level scopes and nested scopes (see definitions 2.9 and 2.7). Top-level scopes are created inside (or on top) of the so called *World Scope* (see definition 2.9), which includes all sensor nodes in the wireless sensor network. All other scopes created inside a top-level or nested scope are nested scopes. Scopes that are created inside the same *Super Scope* (see definition 2.8) can overlap each other, depending on their Scope Definition. This means nodes can be member of multiple of these scopes at the same time, iff they are member of the common super scope. A new nested scope can only be created over nodes that are member of the same super scope, so a scope cannot overlap the boundaries of its super scope. Scopes with different super scopes are independent of each other and rely on the limits of their super scopes, as just mentioned.

In the Scopes Framework the visibility of messages sent through a scope is limited to the nodes participating in the scope. These are all *Member Nodes* (see definition 2.11) and all *Intermediate Nodes* (see definition 2.12). The visibility of a scope itself, which means knowledge about its existence, is limited to the nodes of its super scope. The benefit of these restrictions is that messages are kept in areas where they are needed and other areas of the network are unaffected by the traffic. This restriction is also the first step towards security, as it avoids spreading data of a scope all over the network, but keeping it in the regions the scope is created and used.

### 2.2.3 Multi-Tasking

The Scopes Framework allows the execution of multiple tasks at the same time. To allow different tasks the Scopes Framework manages the registration of a task and the scopes it has created. Only registered tasks are able to use the framework and can receive messages. And only the registered task that created a scope is able to delete it. In a sense this is a notion of ownership of a scope by its creating task. The multi-tasking itself is provided by the underlying operating system.

### 2.2.4 Reusability and Reconfigurability

One major aspect in creating the Scopes Framework was to be able to reuse it in as many scenarios as possible. We reviewed several scenarios and deployments where WSNs are used, like the container harbor scenario or others like habitat monitoring and

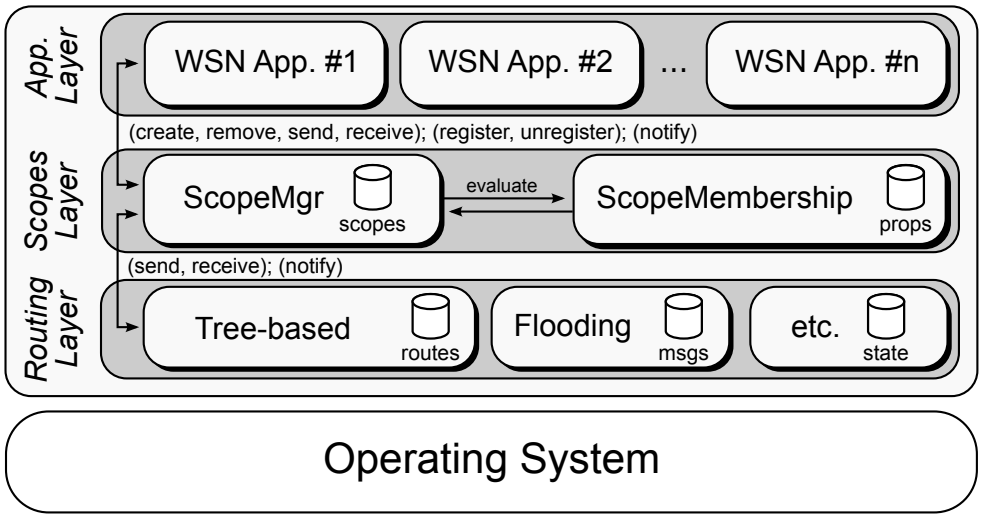



**Figure 2.2:** Layered Architecture of Scopes Framework.

environment monitoring [77, 82, 126, 127, 140]. This led to the understanding, that scenarios are different in the way of their specific tasks, their demand on properties to create scopes and the use of different messaging patterns. To cope with all these demands the Scopes Framework was created in a highly modular way, so it is easy to replace or add modules for routing, property definition or tasks. Especially to support different messaging patterns efficiently it was mandatory to enable support for different routing algorithms, so that the appropriate one can be chosen for a task. Another important objective was that after some time a WSN may be reconfigured to support new tasks that will be added. So a possible over-the-air (OTA) exchange of modules is important and is supported by our design. The transfer itself is handled by the *life-cycle management task* (LCM task) that can be used by the root node of a scope to distribute new modules and trigger their registration. The LCM task can also be used to delete or update existing tasks. An efficient mechanism to distribute modules was proposed in [39].

## 2.3 Layers of the Scopes Framework

The architecture of Scopes is divided into four layers, where three are shown in figure 2.2. The uppermost layer is responsible for providing a multitasking environment and management of the task lifecycle, e.g. the memory management to provide the ability to instantiate and run multiple tasks on a sensor node. These layers are the Scope layer and the Routing layer. The Scope layer is responsible for managing the scope membership status and additional information of the current node. The Routing layer manages network connections and message transfers. The lowest layer is the hardware abstraction layer provided by the wireless sensor network operating system not shown in figure 2.2. The operating system allows access to sensors, the network interface and other hardware via a defined API.

### 2.3.1 Application Layer

The tasks of a WSN application that have to be executed by the local node are located on the application layer. These tasks can be preinstalled or deployed at runtime. Tasks have three interfaces available to interact with the Scopes Framework: task, notification and property repository API.

The *task API* provides the ability to register the task with Scopes, manage and use a scope. The registration is mandatory to make the task known to the Scopes Framework. After that it can be used to interact with a scope and may create new scopes.

The *notification API* notifies the task regarding changes in the node's state of the scope. This means the Scope Framework signals the task if it joins or leaves a scope, or if it receives new scopes. With this information a task can decide which scope to use for communication or it can trigger other operations.

The *property repository API* enables a task to provide properties to the Scopes Framework via the property repository. Providing new properties to the framework has the advantage, that the task can influence scope joins or leaves. It is also possible to read properties, for example, latest sensor data from the framework's repository, and retrieve data for the tasks own usage. The naming of the properties provided here is used to create scope definitions, referring to data of the repository. This can be important in some scenarios, for example, if a node in the harbor scenario detected a very high temperature, it may create a scope over its neighboring nodes to detect if this data was caused by, e.g., a fire.

### 2.3.2 Scope Layer

This layer hosts the *scope manager module* and the *scope membership module*. These two maintain all information regarding scopes and decide what operations are to be executed. The scope manager module is the center of all Scope Framework activities. It stores all information needed for managing the scope's life-cycle and usage. It also tracks the state of the local node in specific scopes. These scopes may be, for example, the scopes a node is member of, scopes that it is only a potential member of [1], and scopes it is aware of and may act as an intermediate node.

The scope manager module has no knowledge about the content of a scope definition. For the scope manager module to be able to decide to join a scope or not, the definition is passed on to the scope membership module. This module can be the standard one provided with the framework, or it can be a custom one. To easily exchange this module there is an API, that can be implemented. Now, to get the definition for the scope, the specification array is parsed and evaluated by the scope membership module and after processing of the data a true or false is sent back, depending on whether the properties could be satisfied or not.

[1] The state depends on the nodes dynamic properties, for details see section 2.3.2.1

Besides the above mentioned interface to the scope membership module, the scope manager module provides APIs to the upper and lower layers. The APIs to the application layer are the task API and the notification API. With the task API tasks can register themselves with the scopes module, so they can interact with it, create or delete scopes, and send and receive data via a scope. The notification API is used to signal changes in known scopes or the status of scopes.

To the routing layer the scopes module also provides the notification API. It signals the same change and status information as to the task modules. Additionally, there are message exchange functions, so the scopes module can send and receive messages to and from other nodes in the network. This API still uses the notion of scope id's to identify the target of a message. This represents some additional effort to adapt a routing algorithm to the Scopes Framework, but it enables Scopes to have exchangeable routing modules.

#### 2.3.2.1 Scope-Membership

A scope is a group of sensor nodes complying with a specified scope definition. The most important part of a specification language for the Scopes Framework is the ability to sufficiently describe these definitions, which mostly rely on node-local properties. The properties used in scope definitions are highly scenario dependent; that is why the scope membership functions are separated from the rest of the scope management. This way the scope membership functions can be easily replaced or extended, depending on the needs of the scenario. For common operations the Scopes Framework provides a default membership module. For more scenario specific operations it may be extended or replaced. The standard operations provided only use node-local information for the sake of efficiency. In a custom module information from the network or combined properties can be used additionally for membership evaluation.

Properties can have different characteristics, static or dynamic. Dynamic properties are very interesting, as the change of the property may lead to a reevaluation of a scope definition. For example, it is possible that the change of the property *"Temperature"* causes the definition for a scope to be no longer fulfilled. In this case the sensor node drops its membership. Dynamic properties are implemented to trigger a reevaluation if a change occurs. This also means that dynamic properties are much more expensive than static ones, as they impose much more computation overhead. Additionally, a node can be in a third state evaluating a scope with dynamic properties in its definition. Besides member and not member it may be a potential member of this scope. This is the case when the static properties of the scope definition match the node's properties and the dynamic properties are currently not met. In contrast to the dynamic properties static properties, such as node id or owner, are simple. These properties are set at one point in time, e.g. before the deployment or before a new scenario is enabled in the network. They are assumed to be stable and no reevaluation is needed.

### 2.3.3 Routing Layer

On the routing layer messages are exchanged between different nodes in the network on one side and between the network and the local scope and task modules on the other side. Not all messages received by a node have to be forwarded to the other layers. For example, if a node is not a member of a scope and the scope has no dynamic properties this scope is not relevant for the local scope manager module.However, as the node can be on the path between two members of the scope, it is important that the routing forwards the messages. To decide how to handle an incoming message and if the message has to be delivered locally, routing algorithms use their routing information. Thid depends on the specific algorithm, and the information about memberships of the local node from the notification API.

The Scopes Framework enables the use of many different routing algorithms and can adapt to different traffic patterns with different routing modules used. The decision which routing module to use is taken by the task that generates the traffic, this is not done automatically. In this sense the developer of a task with special needs for its data traffic, has to select or provide a matching routing algorithm.

As mentioned in the scopes layer section, the routing layer has two interfaces available. The notification API can be used in routing modules to link routing information to a specific scope id, so that messages to or from a scope can be forwarded or delivered locally. Also the notifications make it possible to clean up memory. For example, after leaving or deleting a scope the corresponding routing information may be obsolete and memory can be freed. The message exchange API is used to deliver messages from the outside to the local scope manager module and for sending messages to the network.

## 2.4 Scope Specification

Specifying a scope definition is highly dependent on the scenario. Properties used in the two scenarios proposed in the introduction are also quite different. While in the container harbor scenario properties are related to content or owner; in the emergency response scenario it is more about locations and endangerments. But there are also other properties possible, like restricted areas where it is allowed to enter or not. Most of these properties are representable by a key-value pair and can be combined to a boolean expression to define a scope.

To be able to also support definitions that use more complex expressions or scenario specific operations, we leave the decision about the scope definition to the scope membership module shown above. External applications can resort to the default language described below to create scopes or to scenario specific extensions. Such expressions are parsed and flattened into a pre-order network format specifically designed for sensor networks. This format is descriptive enough to accommodate the necessary expressions, yet compact enough to typically fit in one network message. In-network applications can

also construct scopes by resorting to this predefined byte-string format. Any arbitrary node can be used to create a scope, however in practice we observe mostly two cases for scope creation. Scopes created from gateway nodes have global and permanent character. Regular nodes in most cases create localized neighborhood scopes for tasks such as event detection.

### 2.4.1 Scope Definition Language

The declarative language shown in figure 2.3 defines the available range of scope definitions and includes expressions to create and delete scopes. The basic constructs are shown here, for the version with extended operations see [41]. Scopes can be defined based on static properties, e.g. the owner of a container, or on dynamic properties, e.g. the content of a container. We exemplify the most important constructs within the container harbor scenario, the used scope definitions are shown in figure 2.4. Imagine we want to define a scope `ferdexMotion` over containers from company Ferdex that have been dropped or otherwise shaken above a defined threshold, here 3g. An application running on this scope can, for instance, report the id and position of the member nodes. `ferdexMotion` is satisfiable by nodes with a 3D-acceleration sensor, whose values are greater than the given threshold, and the user-defined variable `company` equals `ferdex`. Properties included in an expression may have a more static (e.g., `company`) or dynamic (e.g., `accel_{x,y,z}`) nature. Dynamic properties are very powerful, but their use is expensive since every change results in a re-evaluation of the scope membership expression.

Finally, scopes can be nested and therefore form a hierarchy. Nested scopes specialize a scope definition by implicitly restricting the membership condition of their parent-scope. For example, if a scope `dhll` is defined over all nodes belonging to company DHLL, the scope `dhllCooling` could specialize it by selecting those nodes attached to a cooling container. The latter can then be used to monitor the temperatures and regulate them correspondingly. Conceptually speaking, nesting scopes contributes clearer definitions and a better organization. Technically, they reduce the communication overhead thus improving the performance and the energy efficiency.

## 2.5 Scopes Network Behavior

Now that we know the details of the Scopes Framework's layers and how to specify a scope definition, we will have a closer look at the network-wide behavior of a scope. This behavior is the same for all scopes and independent of each other.

The lifecycle of a scope has three different phases. First, the *creation phase*, where the scope is established by distributing its definition and the joining of nodes to it. The second phase is the *maintenance phase* or usage phase. Here tasks can use the scope to send messages to member nodes or member nodes back to the root node. In this phase the framework maintains a scope, so that it is kept alive and joining or leaving nodes are

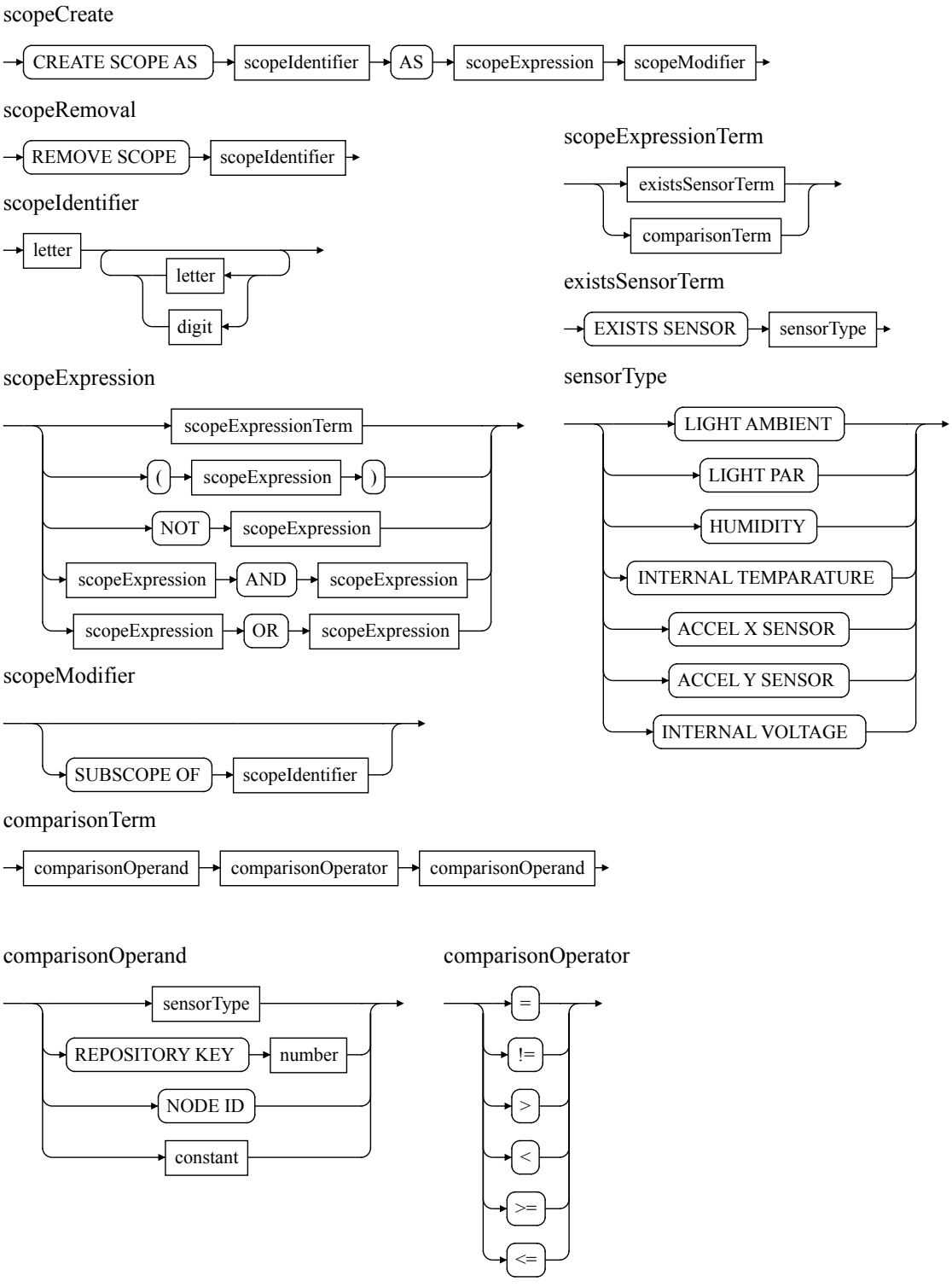



**Figure 2.3:** Scopes Definition Language Grammar

```
CREATE SCOPE ferdexMotion AS (
  company = 'Ferdex' AND
  EXISTS SENSOR Acceleration3D AND
   (accel_x > 3.0 OR
    accel_y > 3.0 OR
    accel_z > 3.0) );
```

```
CREATE SCOPE dhll (
  company = 'DHLL' ) ;


CREATE SCOPE dhllCooling (
    EXISTS SENSOR Temperature AND
    containerType = cooling
) AS SUBSCOPE OF dhll;
```

**Figure 2.4:** Scope declarative definitions

handled properly. At the end is the *deletion phase*. This phase can be triggered in two ways: the command to delete a scope is explicitly called from a task, or if the periodic keep-alive refresh is stopped, it is triggered after a timeout. In this phase the scope is deleted and all resources that were kept for management are freed.

### 2.5.1 Creation Phase

The scope creation phase starts with sending the first scope creation request and ends with all nodes in the network that match the scope definition becomming members of the scope.

When a scope is created from a task, figure 2.5, first a *scope creation request (SCR)* is sent to all possible members. Which nodes are possible members depends on the parent scope of the new scope. Is it created on top of the World Scope the SCR is delivered to all nodes in the sensor network, as it can not be known in advance which nodes will participate in the scope. Is the new scope created as part of another scope the SCR is disseminated through the super scope to all members of this scope.

If an SCR is received by a node the scope of the SCR is added to the scope table of that node, if the scope definition contains dynamic properties that have to be monitored, or if the node becomes member of the scope by fulfilling the static properties. Is the node becoming a member it will join the scope. Also if a scope definition with dynamic properties is fulfilled the node joins the scope. Does the evaluation of the dynamic properties, now or later, not match the definition it will leave the scope. The scope will still be added to the routing table as it may later match the dynamic properties.

The routing algorithm is notified about state changes of a scope. The states are *added*, a scope is just added to the scopes table, or *joined*, the node is member of a scope. The added state is used by the routing modules to do all necessary steps to establish a connection with the root node of the scope, so messages can be exchanged. Messages in a scope are not yet delivered to the local tasks unless the node joins the scope. In the added state the node can evaluate dynamic properties and act as intermediate node. If the stat is joined, messages can be delivered locally.

When the scope management module receives an SCR some checks are done to decide if the node is member of this scope. You can see this in figure 2.6. The first checks make

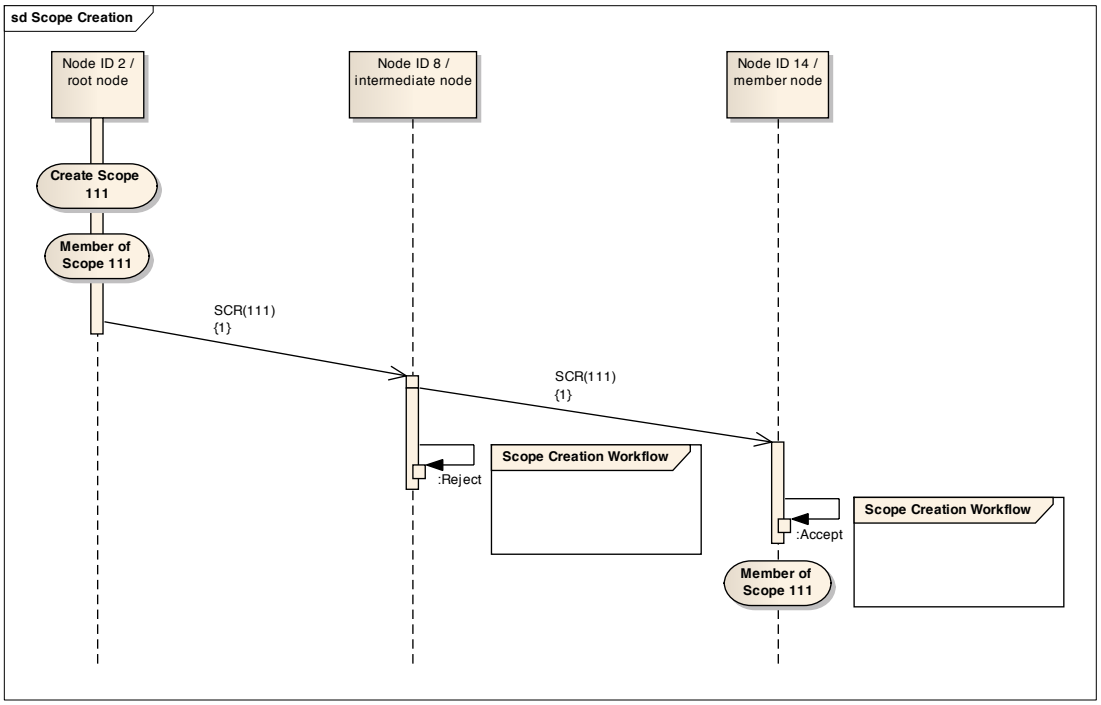



**Figure 2.5:** Scope Creation Message Sequence

sure that the message was received the proper way. It is checked if the specified super scope is known and if the node is member of it. The next step checks if the module handles an SCR or a scope refresh (see next section, 2.5.2), here we will go on with the SCR tree. For an SCR it is then checked if the target task is registered and available. If these steps are all answered positive the scope definition is evaluated. Does the node match the definition it adds the scope to the scope table and joins it. Does it not match the criteria, but the scope definition contains dynamic properties, the scope is just added to the internal scope table. If neither is the case the node ends the processing of the SCR.

### 2.5.2 Maintenance Phase

The maintenance phase starts after the initial creation of the scope and ends either when the scope is deleted from the root node, or no refreshes are sent to the scope and so the time-to-live for the scope ends.

After the establishment of a scope it can be used to exchange data. Data can be sent from the root node of the scope to the member nodes or from the members to the root node. To maintain the scope in a working state in regular intervals *Scope Refresh (SR)* messages are sent by the scope's root node. These intervals can be configured and are one of the design parameters of a sensor network that uses the Scopes Framework.

The cause for regularly sending SRs is to handle nodes joining or leaving the WSN, where a node can leave a scope intentionally or as result of a failure. Handling new nodes in the network with SRs is similar to the SCR, therefore the SR is also sent via the super scope. If a SR is received the scope definition is evaluated and the node

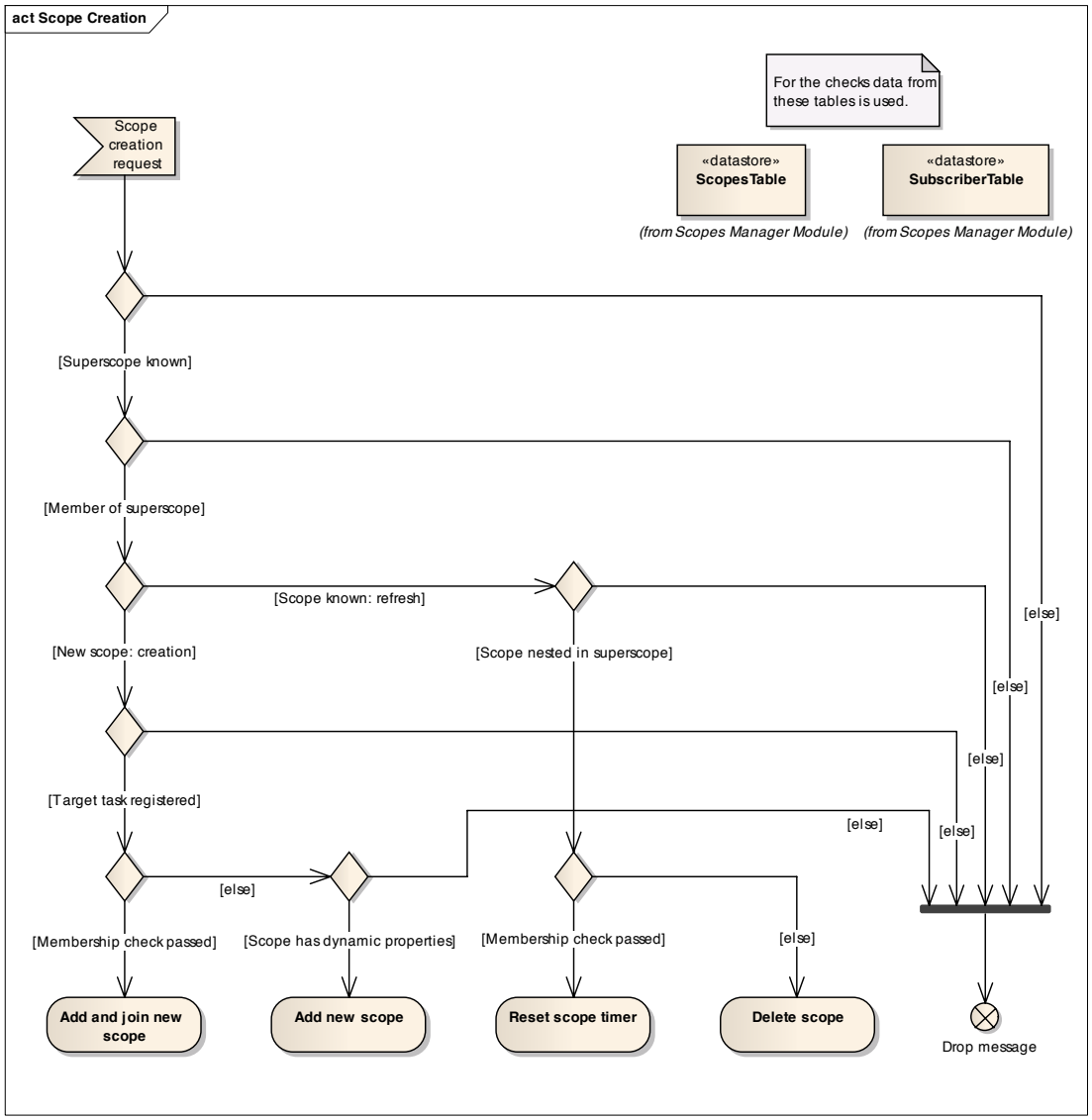



**Figure 2.6:** Scope Creation Workflow

may join the scope or not. Handling leaving or failing nodes results in reevaluating the routing path to possibly reconnect interrupted paths. How to do that depends on the used routing algorithm. But the SR can be used as trigger for this reevaluation.

Basically SRs are SCRs, but for SRs two different kinds of messages are possible. A *full SR* includes the same data as an SCR and is the standard case. It combines all the header data plus the scope definition. A *simple SR* only consists of the header information.

Of course there is a difference in functionality of the two refresh messages. The full SR can be used to help with new nodes in the network and also with leaving or failing nodes. The simple refresh cannot integrate new nodes into the network, this is obvious as the scope definition is missing in this message. But still it can keep the network alive and can trigger the renewal of the routing path as nodes that already know the scope have stored the scope definition and so can use the simple refresh message like a full SR. We introduced this mechanism to reduce the load that a full SR may introduce to an intensively used network or if a large scope definition is used. For a large scope definition the difference between a full and a simple SR can be like a few bytes vs. some 100 bytes, as we will see after introducing our security measures.

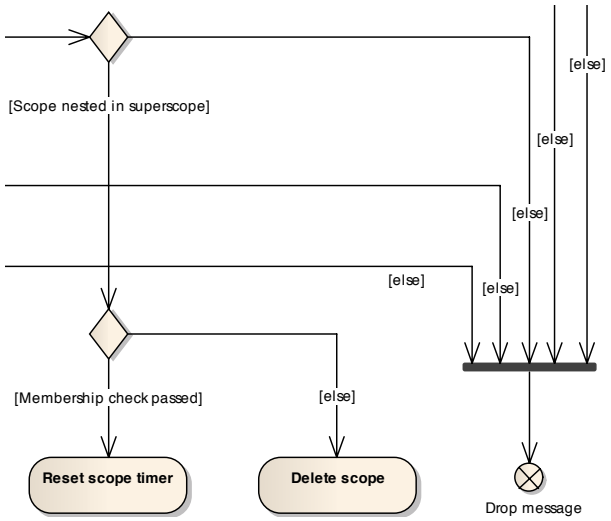



**Figure 2.7:** Scope Maintenance Workflow – detail

The detail shown in figure 2.7 is known from the SCR processing in figure 2.6. So we concentrate on the new sub tree. After taking the decision that the received message is a SR, it is checked if the parent scope of the scope to be refreshed is the same, so we can be sure that we update the correct scope. For a full SR the scope definition is evaluated and if necessary the scope status changed. If the status changed the routing, tasks are notified about this change. If the node is a member of the scope its time-to-live counter is reset. This counter is further explained in the next section, 2.5.3.

Besides the maintenance part, in this phase the scope is operational for its actual job: the message exchange. Since the actual exchange highly depends on the algorithm used, be it a gossiping-, tree- or mesh-approach, the mechanisms how tasks get to send messages and how they receive them are defined in the Scopes Framework.

Before a message is handed over to the routing module it is checked if it is valid and can be transmitted, see figure 2.8(a). Therefore, the task that tries to send the message has to be registered and the scope the message is sent to has to be known to the framework and the node has to be a member. Is one of the previous checks negative, the message is not handed to the routing module and dropped. To receive a message, it is handed from the routing module to the scope manager module and here we also apply the previously mentioned checks, but with the difference that we have to filter the intended target of the message first, member node or root node, see figure 2.8(b). After the look-up of the scope and if the node is member in this scope, it depends on the intended target of the message how it is further processed. If it is for all member nodes of the scope, it is only checked that the receiving task is properly registered and then the message is delivered. Is the message targeted at the scope's root node, first it is checked that the local node is the creator of the scope and then after the registration of the receiving task is assured it is delivered. As seen before with sending a message, if any of the above checks is negative the message is dropped and not delivered to a task.

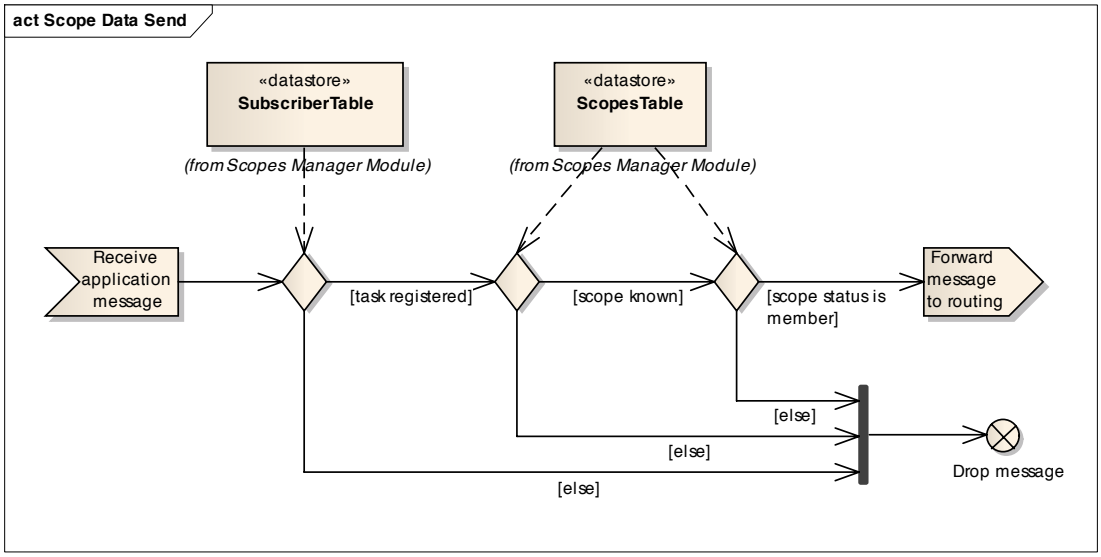
### 2.5.3 Deletion Phase

The deletion phase starts with the task that created the scope on the root node issuing the deletion command, or if the time-to-live counter of a scope runs out because of not receiving any SRs. The expiration of the counter can occur because of, e.g., physical problems with the wireless network, or the loss of the deletion message in the network after a regular deletion and no SRs being sent anymore by the former root node.

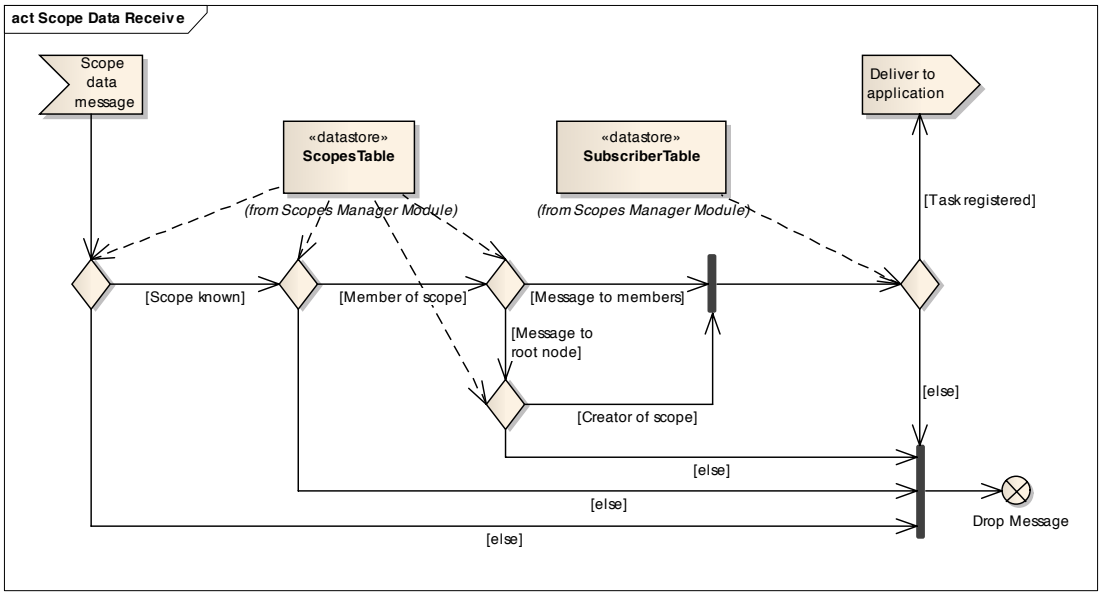
As just mentioned, a scope can be deleted in two different ways. First, the root node of a scope sends a deletion message to all members of the scope. In this case the scope is deleted instantly and the used resources are freed. To reach also nodes that are just potential members because they do not meet dynamic properties, the deletion message is sent like a data message through the super scope. With the deletion of a scope also all its subscopes are recursively deleted.

The second way is, that for some reason no SRs are received by a node for a configurable time. This scope *time-to-live (TTL)* is also one of the Framework's design parameters. If no SR is received before the TTL expires, the scope is assumed to be dead. This means that there is no route to the scope root node or the root node itself has left the scope. In principle this means that no one is interested in the data transmitted via the scope and so it can be deleted safely to save resources. When the TTL runs out it is deleted as if a deletion message was received and resources are freed without any further notice to other nodes in the network.

If a deletion message is received by a node, it is checked by the scope manager to determine if it is a valid deletion message. If it is valid the scope is deleted. This check is depicted in figure 2.9 and does some sanity checks if the scope to be deleted is valid by checking scope and super scope, and if the message was sent from the root node of the scope.




(a) Sending.




(b) Receiving.

**Figure 2.8:** Scope Data Exchange Workflow

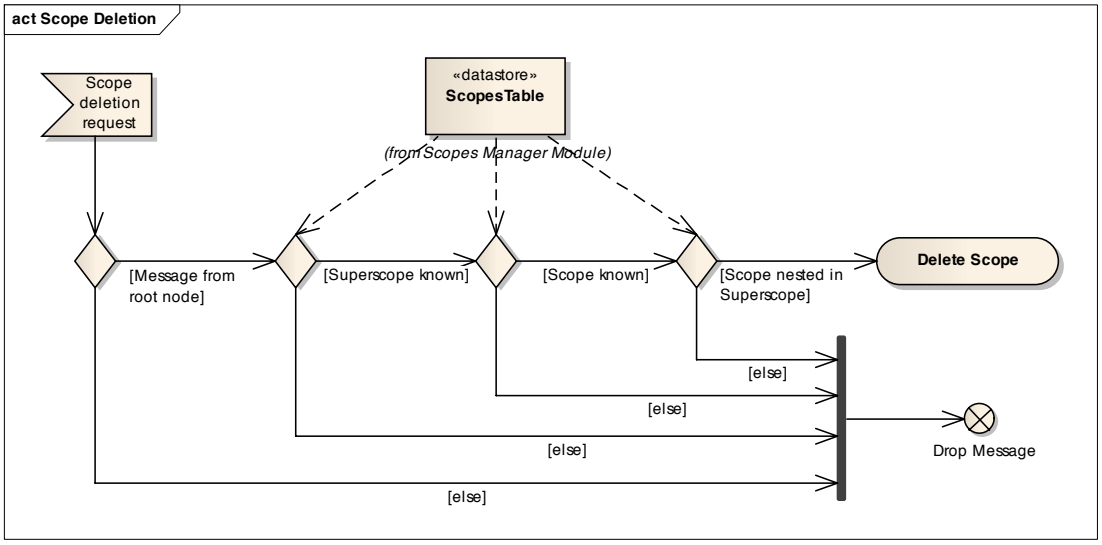



**Figure 2.9:** Scope Deletion Workflow

# 3 Securing the Scopes Framework

## Contents



Security is an important concern for WSNs.

For the Scopes Framework we identified two major security issues. First, access control based on the properties of the scope's definition is a mandatory step to ensure only authorized nodes join a scope. Second, a secure group communication is needed to allow confidentiality and integrity in data exchange. Therefore, the secure distribution of a group key has to be supported. All this has to be integrated in a way that takes the distributed nature of the Scopes Framework into account.

In the following the security assumptions and models used are defined and after that the security architecture introduced into the Scopes Framework is described. In the end the security integration points within the framework are explained and shown in detail.

But first, we will show a short introduction to security, the algorithms used and the theory behind it.

## 3.1 Prerequisites

### 3.1.1 Security Algorithms

Today security is an important issue and it gets increasing attention. But security is also a question of trust. The user has to trust the manufacturer of, for example, an

embedded device that secure algorithms are used and implemented correctly. Often security is achieved by using proprietary solutions or just as 'security by obscurity'. Proven secure (open source) solutions are available. This does not mean that open source is always better than closed source, but it opens the security design for review by everyone who is interested, which can lead to a more robust security and increased trust.

Today more and more flaws in security designs and algorithms are discovered and embedded systems are becoming targets for industrial espionage. With this in mind, security patterns in embedded systems that are based on hiding system details and not really implementing security are obsolete, as more and more often embedded systems are deployed unattended. This thread is getting even worse for WSNs as they are meant to work in distant areas or depending on the scenario may be just dropped at a remote location [67].

Security goals that have to be achieved by a system are quite different as they often depend on the usage scenario. But the possible goals are common over all computing systems. Still, they may have different characteristics on different computing systems, for example, data secrecy on a wired network may be achieved by restricting physical access to the infrastructure, where data secrecy in a wireless network cannot be achieved by these measures. In [19,135] a set of security goals related to WSNs are presented. Security goals are not only relevant for a whole security design, but also for the algorithms used by it. Therefore, we will take a closer look at possible security goals.

Authentication ensures that when a node communicates with one of its peer nodes, this peer node is the one it pretends to be.

Authorization enables only authorized nodes to access other network nodes or resources.

Availability ensures that network services are at the disposal of authorized parties, even though denial-of-service attacks are carried out.

Freshness describes how current a message or key is. A system that ensures this security goal has to make sure that a node always gets the most recent message or key. Also replaying of messages has to be prevented.

Integrity ensures that data is not altered while transported from sender to receiver. This includes accidental and intended alteration.

Non-repudiation prevents a node from denying to have sent a message.

Secrecy or Confidentiality ensures that the content of a message is kept secret. This means that only authorized nodes can access the data and an eavesdropper is prevented from reading the message. For WSNs with high churn rates also *forward* and *backward secrecy* are important security goals. Forward secrecy ensures that a leaving node cannot access any future messages and backward secrecy that a new joining node cannot access content of messages before joining.

A common approach to secure a data connection between two parties is to use cryptography. There are two types of basic cryptographic protocols, symmetric and asymmetric cryptography.

Symmetric algorithms use the same key on both ends of the communication channel to encrypt and decrypt a message. The advantage of symmetric algorithms is that you need quite small key-sizes to get good cryptographic strength. But as on both sides the same key is used, there has to be a second secure channel to transmit the key from one end to the other. For WSNs symmetric cryptography is a good choice, as most of the sensor node platforms already have hardware blocks for symmetric cryptography. To get the symmetric key to the decrypting entity is a hard problem, as the key has to be transmitted via the unsecured wireless connection as a second channel. To secure the symmetric key in transit, asymmetric approaches are used.

Asymmetric algorithms do not need a second secure channel to transmit their keys, as in this scheme a public/private key-pair is used. The private part stays at the receiver of a message and the public part can be made available to everyone who is interested to send a message to the receiver. The message can then be encrypted with the public key and only the holder of the private key can decrypt the message. For sensor networks there are two problems using asymmetric cryptography. On the one hand there may be not enough space to store all public keys for all other nodes in a larger network. With nodes leaving and joining the network updating this database on every node would not be feasible. On the other hand there could be a public key infrastructure (PKI) to manage and make keys available on demand, but this introduces more traffic inside the sensor network and it removes the strength of an ad-hoc network to just deploy new nodes and self-configuring the network.

It seems natural for WSNs to use symmetric cryptography for message encryption. In our approach we introduce dynamic groups based on properties to have a secure way to distribute the encryption key. We use attribute-based encryption (ABE) schemes, that were previously only used on systems larger than WSN nodes. ABE is based on elliptic curve cryptography (ECC) and pairing in elliptic curves. Therefore, we will explain the basics of ABE and the used ABE scheme in the next chapters.

#### 3.1.1.1 Elliptic Curve Cryptography

This subsection is taken from [45] and will briefly introduce elliptic curve cryptography (ECC).

ECC is an asymmetric cryptographic scheme and is based on the *elliptic curve discrete logarithm problem (ECDLP)* in a finite cyclic group.

First we clarify what a group is: An *abelian group* $(G, *)$ consists of a set $G$ and a binary operation $* : G \times G \rightarrow G$ satisfying these properties:

1. *(Associativity)* $a * (b * c) = (a * b) * c$ for all $a, b, c \in G$.
2. *(Commutativity)* $a * b = b * a$ for all $a, b \in G$.

3. *(Existence of identity)* There exists an element $e \in G$ such that $a * e = e * a = a$ for all $a \in G$.

4. *(Existence of inverse)* For each $a \in G$, there exists an element $b \in G$, called the *inverse* of $a$, such that $a * b = b * a = e$.

The group operation is typically called addition ($+$) or multiplication ($\cdot$). For the addition as operation, the group is called *additive* group, with identity element denoted by 0, and inverse element of $a$ denoted by $-a$. In case of multiplication as operation the group is called *multiplicative* group, with identity element denoted by 1, and the inverse element of $a$ denoted by $a^{-1}$. The group is *finite* if $G$ is a finite set, in which case the number of elements in $G$ is called the *order* of $G$. The tripple $(\mathbb{F}_p, +, \cdot)$ is a *finite field* denoted further $\mathbb{F}_p$.

If you have a finite additive group $G$ of order $n$ and $g \in G$, the smallest positive integer $t$ with $t \cdot g = 1$ is called the *order* of $g$, where $t \cdot g$ denotes $t$ times adding $g$. $t$ always exists and is a divisor of $n$. The set $\langle g \rangle = \{t \cdot g | 0 \leq i \leq t-1\}$ of all powers of g is itself a group under the same operation as $G$. This is called a *cyclic subgroup of G generated by* $g$. The same is true for multiplicative groups. Has $G$ an element $g$ of order $n$, then $G$ is a cyclic group and $g$ is called a *generator of G*.

With the group definition as basis we define an *elliptic curve* $E$ over $\mathbb{F}_p$ as

$$y^2 = x^3 + ax + b$$

where $a, b \in \mathbb{F}_p$ satisfy $4a^3 + 27b^2 \not\equiv 0 \pmod{p}$, $p$ is a prime number and $\mathbb{F}_p$ is the field of integers modulo $p$. A pair $(x, y)$ with $x, y \in \mathbb{F}_p$ is a *point* on the curve if it satisfies the given equation. The set of all points on $E$ is denoted by $E(\mathbb{F}_p)$. The point at infinity $\infty$ is also said to be on the curve $E$. The set of $(p, E, P, n)$, with $E$ being an elliptic curve over $\mathbb{F}_p$, $P$ being a point on the curve and $n$ the prime order of $P$, are called the *domain parameters* of an elliptic curve.

There are well known methods to add two points on an elliptic curve $(x_1, y_1)$ and $(x_2, y_2)$ to get a third one. The set of points $E(\mathbb{F}_p)$ forms with this addition method an additive abelian group with the infinity element $\infty$ as identity. A cyclic sub group of these elliptic curve groups can be used to define a discrete logarithm system.

Given the public domain parameters $(p, E, P, n)$ of an elliptic curve, an integer $d$ selected at random from the interval $[1, n-1]$ is denoted the private key of a public/private key-pair. The corresponding public key is $Q = dP$. The problem of determining $d$ from the given domain parameters and $Q$ is the *ECDLP*. The hardness of this problem ensures security of elliptic curve cryptography. To solve this problem algorithms like number field sieve (NFS) [37, 104] or Pollards rho algorithm [103] are used. To ensure security choosing good domain parameters is vital. Therefore, a set of curve parameters expected to be secure are published by the industry consortium SECG [121] and recommendations regarding key sizes to be used can be found on the following Website [9],

where a set of recommendation papers from different organizations are compared. The largest ECDLP instance solved as of today is from 2015 and it was solved over a 113bit binary field [139].

Compared to other public key crypto systems like RSA, ECC has the advantage to reach a high security level with rather small key sizes. A key of 160bit ECC correlates to a 1024bit RSA key, or 512bit ECC correlate to 15,360bit RSA. Besides the key size itself the reduction of computation is the actual win. Computing 160bit mathematical operations save a lot compared to 1024bit operations [43]. As the reachable security levels and performance is as requested, the question for functionality remains. The typical operations for a public key system are key exchange, signage and encryption, and for all these ECC has a counterpart, e.g. elliptic curve Diffie-Hellman (ECDH), elliptic curve digital signature algorithm (ECDSA) and elliptic curve integrated encryption scheme (ECIES).

The public key approach aims at one-to-one communication. A sender encrypts a message for a specific receiver. There are approaches to let a message have multiple recipients, but they are not what we are looking for in WSNs. We want to use an encrypted group establishment where an unknown number of nodes can decrypt the group creation to possibly join the group. This is not feasible with classic public/private key approaches, but there are new approaches like Attribute-based Encryption (ABE) that allow this pattern. It is based on pairings in elliptic curves, so we will have a look at them in the next chapter.

#### 3.1.1.2 Pairing on Elliptic Curves

A good introduction to pairings (and bilinear maps) is [7] from Bethencourt, which we will summarize.

In 1993 pairings were first used in cryptography, but at that time they were used to break ECC crypto by reducing the ECDLP to the DLP in finite fields. The first use in a cryptographic protocol was in 2000 for a 3-party Diffie-Hellman by Joux [59], followed by the first practical identity-based encryption scheme by Boneh and Welsh [10] in 2001. This is the predecessor of attribute-based encryption, that we will discuss in the next section.

Bilinear maps are also called pairings because they associate elements from one cyclic group $G$ to elements of another such group $G_t$. Bilinear maps are defined over two input groups $G_1, G_2$, but most of the time these groups are the same. [1] This group $G$ is also mostly an elliptic curve or a sub group of it. $G_t$ is a finite field. Now, a *bilinear map* from $G \times G$ to $G_t$ is a function $e : G \times G \rightarrow G_t$ such that for all $u, v \in G; a, b, \in \mathbb{Z}$,

$$e(u^a, v^b) = e(u, v)^{ab}.$$

[1] This is the way it is also used in the algorithm in this work.

One additional condition that we assume is that being $g$ a generator of $G$ the map $e(g, g)$ generates $G_t$.

Bilinear maps can be computed with the Tate or Weil pairing, while the Tate pairing is normally faster. These algorithms are computationally expensive and very complex. This is why we skip details about them at this point. For our system we used the Tate pairing for performance reasons.

Concerning security: using these algorithms makes it mandatory to ensure that the elliptic curve, or its mapped finite field, has a bit-size that is assumed secure. On the previously mentioned website [9] are also recommendations for the finite field size.

In the following we will show encryption algorithms that are based on pairings in elliptic curves.

#### 3.1.1.3 Attribute Based Encryption

Attribute-based encryption (ABE) evolved from identity-based encryption (IBE) which was first mentioned by Shamir [117]. The IBE crypto system uses a public/private key approach, where the public key is the user's identity. Here almost everything that uniquely identifies the user can be chosen as public key, e.g. email address, name and postal address or a phone number. A key generation center then generates from the identity and a secret seed the private key that the user can use to decrypt messages or sign emails, like a regular public key crypto system. But there is no Infrastructure needed to distribute the public keys as they are implicitly given by the identity of the receiver. Sahai et.al. introduced *fuzzy IBE* [113], an IBE system that can cope with small deviations of an identity, like they are common in reading biometric attributes. Additionally, they propose a second use case for fuzzy IBE as so called *attribute-based encryption*. Here not the identity is used, but a set of attributes that have to match with the private key to be able to decrypt a message.

There are two well known ABE schemes, ciphertext-policy attribute-based encryption (CP-ABE) [8] from Bethencourt et. al. and key-policy attribute-based encryption (KP-ABE) [38] from Goyal et. al. . CP-ABE uses the same approach proposed for fuzzy IBE. By encrypting a message under an access structure that tells which attributes are needed to be able to decrypt the message and having private keys with a set of attributes that have to match the access structure used for encryption to decrypt a message. KP-ABE in contrast uses a reverse approach. Here the message has a set of attributes and the private key contains an access structure where is stated which attributes can be decrypted. For our approach we chose the scheme of Bethencourt et. al., which is explained in detail in the next section.

**Ciphertext Policy – Attribute Based Encryption**

As mentioned above we were looking for a security algorithm that works well in a distributed environment and nicely integrates with our scopes property structure. With the elliptic curve based *ciphertext policy – attribute based encryption* (CP-ABE) proposed

by Bethencourt et al. in [8] we are able to provide secret keys associated with a set of attributes. These are used on the sensor nodes to decrypt matching ciphertexts. And we gain ciphertexts associated with a policy which is used to specify an access structure that needs to be fulfilled by a node to decrypt the message.

In [150] another ABE scheme is proposed for the use in wireless sensor networks, key policy attribute-based encryption (KP-ABE). In KP-ABE the secret keys have an access policy and the ciphertext includes properties. This is used for fine grained access control to sensor data. For our framework this is not suitable, as we need the access structure with the ciphertext and properties on the keys. In this sense KP-ABE works the wrong direction as we need it, in contrast CP-ABE works the way we need it. How this is applied to the Scopes Framework will be shown later in this chapter.

Bethencourt et al. defined five basic operations in CP-ABE of which we implemented four: *setup*, *key generation*, *encryption* and *decryption*. The fifth operation is used to generate a sub key of a given secret key with a restricted set of attributes, which is not used in our Framework. In the following we describe the four implemented operations and show the parameters of the different keys and the ciphertext. For an in-deep explanation of the parameters, refer to [8].

Setup. The setup does not take any input. It does select a bilinear group $\mathbb{G}_0$ with prime order $p$ with generator $g$ and two random exponents $\alpha, \beta$. With the given elliptic curve parameter list it generates the public parameters PK and a master key MK. $e(...)$ is the pairing operation.

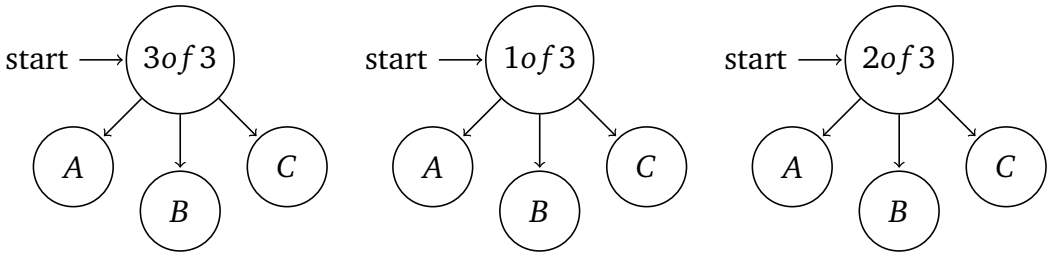
$$PK = \langle \mathbb{G}_0, g, h = g^{\beta}, e(g,g)^{\alpha} \rangle$$
$$MK = \langle \beta, g^{\alpha} \rangle$$

Key Generation. The key generation takes the master key MK and a set of attributes $S$ with a chosen random $r$ and $r_j$ for each attribute, and computes a matching secret key SK. The secret key SK includes each attribute from $S$ with a corresponding component needed for the decryption process.

$$SK = \langle D = g^{(\alpha+r)/\beta}, \forall j \in S : D_j = g^r \cdot H(j)^{r_j}, D'_j = g^{r_j} \rangle$$

Encryption. The encryption takes as parameters the public parameters PK, a message $M$ and an access policy $\mathbb{A}$ with a set of attributes and $Y$ as set of leaf nodes of $\mathbb{A}$. The message $M$ will be encrypted to the ciphertext CT in a way that only with a matching secret key SR which satisfies the policy $\mathbb{A}$ it can be decrypted. The cipher text CT contains implicitly the specified access policy.

$$CT = \langle \mathbb{A}, \tilde{C} = Me(g,g)^{\alpha s}, C = h^s, \forall y \in Y : C_y = g^{q_y(0)}, C'_y = H(\text{att}(y))^{q_y(0)} \rangle$$




(a) A and B and C (b) A or B or C (c) 2of3

**Figure 3.1:** CP-ABE Access Policy: Threshold Gates with three elements

Decryption. The decryption takes the public parameters PK, a ciphertext CT including an access policy $\mathbb{A}$ and a secret key SK which represents a set $S$ of attributes. If the attributes of secret key SR satisfy the access policy $\mathbb{A}$, the ciphertext CT will be decrypted and the message $M$ will be returned.

The above mentioned access policy in the CT is created using a tree structure that an SK with the associated attributes has to satisfy to be able to decrypt. The tree is built by so called threshold gates, for an example see figure 3.1, where each interior node is a threshold gate and each leaf in the tree is associated to an attribute of the policy. With threshold gates it is quite easy to model 'AND' or 'OR', e.g. `2of2` or `1of2`. Numerical attributes can be achieved by exploding the numerical value to its bit representation and have one attribute representing one bit. Together with the threshold gate approach, comparisons like $<, >, =, \leq$ and $\geq$ can be modeled. Now, to be able to decrypt a ciphertext there has to be an assignment of attributes from the SK to nodes of the tree such that the threshold gates are satisfied. This can be efficiently evaluated and if the tree is satisfied, the computation to retrieve the message M is executed.

CP-ABE is collusion resistant; this is typical in attribute based encryption [38, 113]. It means that an adversary can obtain as many SKs as he wants, as long as he does not obtain a key that decrypts the message by itself. With the obtained keys he is not able to successfully decrypt the CT. A general problem that comes with attribute based encryption is key revocation [10, 117], since the encrypting sender cannot verify that an encrypted message can only be decrypted by a non-revoked receiver and with CP-ABE there may be many secret keys that match a given policy. Bethencourt et al. refers here to a possible solution to include expiring attributes in the secure key attributes. In [150] a key update mechanism is proposed for a different ABE system. As we were able to introduce the idea in CP-ABE, see section 3.3, the mechanism will now be described in more detail.

#### Revocation via key update

The key update mechanism was proposed for key policy attribute-based encryption (KP-ABE) [38], which is related to CP-ABE. In the KP-ABE extended by the revocation algorithm from [150] the basic idea is to exchange the parameter $y$ in the MK, PK and

SK with a new value. As the parameter is also used in the ciphertext it can only be decrypted with a matching access policy and the respective $y$. The keys in KP-ABE are defined like[2]

$$
\begin{aligned}
MK &= \langle y, t_1, \cdots, t_{|N|}, \beta \rangle, \\
PK &= \langle G_1, g, e(g,g)^y, \{T_i = g^{t_i}\}_{i \in N}, g^{\beta} \rangle, \\
SK &= \langle g^{\frac{y-\theta}{\beta}}, \{D_i = g^{\frac{q_i(0)}{t_i}}\}_{i \in L} \rangle.
\end{aligned}
$$

Most of the parameters look familiar compared to CP-ABE. The important entries for key update are $y$ from MK, $e(g,g)^y$ from PK and $g^{\frac{y-\theta}{\beta}}$ from SK.

Now, to update the keys a new $y'$ is chosen, which replaces the value of $y$ in MK. In PK $e(g,g)^y$, $y$ is also directly replaced by $e(g,g)^{y'}$. The SKs are a bit different, as the $y$ in $g^{\frac{y-\theta}{\beta}}$ has to be replaced. To do this first the term $g^{\frac{y'-y}{\beta}}$ is computed and after a multiplication with the old term the result is $g^{\frac{y'-\theta}{\beta}}$. This last step has to be done at all SK holders that are not to be revoked. This also means that only authorized holders of secret keys have to get the $g^{\frac{y'-y}{\beta}}$ and SK holders that have to be revoked should be excluded in the distribution of the update term.

◊

In general, memory is a rare resource on sensor nodes regardless of flash or RAM space. To reduce the code needed on an individual node, we can limit the functionality depending on the use case of a node. A sensor node in the WSN has always to be able to decrypt messages. So, decryption has to be available on all sensor nodes in the network. Encryption has to be available on all nodes that may create a new scope to be able to encrypt a new scope creation request. Setup and key generation of the CP-ABE algorithm should take place outside the sensor network. This way the master key can be kept outside the network, which is important to security, as no one can retrieve the master key to generate arbitrary keys.

### 3.1.2 Security in Wireless Sensor Networks

Security in WSNs is an important issue, as all data is transmitted through the wireless broadcast medium and messages can be received over a wide area. This scenario includes a wide variety of potential thread areas, like key management, cryptography, secure routing, secure data fusion and other issues [19]. Compared to computer networks there are many common concerns, but with a different view on the topics. This is due to the differences of the environment in WSNs and common networks. Besides the medium, cable vs. wireless, the resources are very different. From power from a wall

[2] The equations of KP-ABE keys and the revocation algorithm were taken from [150].

socket vs. battery power to server-size computers vs. embedded hardware with some MHz computing power.

One of the basic concerns about security is to encrypt data to store or transmit it in a secure way. For a personal computer this is in principle a solved case. There are plenty of algorithms that may get insecure over time, be it because of design flaws or because of raw computing power that reduce the effectiveness of the algorithms' basic mathematical problem. Even special hardware add-ons for computers are available for commonly used cryptographic algorithms to unload the cpu from computation. Of course, there are replacement algorithms that are better suited to the new advances in their surrounding, but they have to be implemented and deployed on systems in the field before they can enhance security.

On a sensor node (and for embedded systems) there are different ways to get algorithms included. First, adding a custom chip or FPGA to the sensor node to do the computations, like in PC systems add-on cards are used, but on a much more integrated level. Second, a cryptographic function block inside a microcontroller, that enables a specific algorithm. Or, third, a software implementation that is running on the microcontroller. On a sensor node mostly one of the last two options are found, or a combination of both, as the function blocks available on microcontrollers are most of the time symmetric algorithms with basic functionality. To prevent problems with key exchange of the symmetric key, asymmetric algorithms may be used to share the key for the function blocks. The type of algorithm is very important for feasibility in WSNs. Generally speaking the use of symmetric cryptographic algorithms, this means the same key is used on both sides, puts less load on a microcontroller than an asymmetric algorithm, e.g. using a public/private key pair, given the same cryptographic strength. Asymmetric algorithms are normally executed by the cpu, so the small symmetric key is transferred using the asymmetric cryptography and the symmetric algorithm is then used for the larger data to be exchanged.

Besides using asymmetric cryptography, there are also specific key management protocols in WSNs that reduce the computational load for the sensor node and ensure the secure key distribution in the network. A common approach is to deploy different kinds of keys on sensor nodes to establish different kinds of security domains. So, a routing algorithm may establish pair-wise keys to secure the data transmission between neighboring nodes. Alternatively it can establish group keys, e.g. for local clusters, a neighborhood or for all backbone nodes. This has the advantage that less storage is consumed for storing keys and sensor nodes can communicate with a larger number of nodes. But as a drawback, if one of the keys is compromised the communication of a larger group of nodes is accessible to the adversary. The larger the group a key is distributed to, the larger is the risk that a key gets compromised. As the key is stored in more sensor nodes it has a larger possibility to be retrieved by an adversary that compromised a node in the field. Key management protocols can use quite different approaches with probabilistic or deterministic characteristics. A probabilistic approach preloads a different set of keys to every node in the sensor network from a global set of keys. The

distribution and the number of keys are chosen based on the probability to enable communication between two neighboring nodes, as the two nodes must have the same key. In contrast, a deterministic approach ensures that two nodes have corresponding keys. E.g., LEAP [153], Localized Encryption and Authentication Protocol, provides four different keys, each one for a different kind of communication pattern found in WSNs. The keys are a group key that is shared with all other nodes in the network to send broadcast messages, a cluster key shared with neighboring nodes grouping to a cluster, a pairwise key shared with another node for one-to-one communication and a key shared with the base station.

Secure routing algorithms use a wide variety of well known cryptographic methods to achieve different security goals. Besides encryption of plain texts and decryption of cipher texts, the computation of hash functions or message authentication codes (MAC) are common building blocks of these algorithms. Also the different modes of operation of cryptographic algorithms, like cipher block chaining (CBC) or counter mode (CTR), are commonly used. [62] One of the first secure routing algorithms in WSNs is SPINS [98]. It consists of two components that provide authentication ($\mu$TESLA) and data confidentiality and data freshness (SNEP). With these it provides an authenticated routing protocol. It employs a symmetric block cypher to retrieve stream encryption/decryption and a MAC to achieve its security goals.

A problem for secure routing is the operation of aggregation. It is very popular in WSNs, as it enables saving of communication bandwidth and energy by reducing the amount of data to be transferred. If data is requested, e.g. by the base station, data aggregation takes the data from a node's neighbors and combines it with its own data to a new value. This combination can be done by mathematical or statistical functions, like simple adding or computing an average. In general there are two choices in a WSN where a secure routing scheme is deployed for applying aggregation. The encrypted data can be decrypted, then the aggregation function can be applied to the plain text of the data values and afterwards the data gets encrypted again. This is a very simple approach that can be used in almost every encryption scheme. This approach makes it mandatory to have every decryption key available on a node that may be used by encrypted data passing this node. Otherwise it may not be possible to compute an aggregation value. Deploying many keys on a single node increases the risk of physical attacks on a sensor node. A more advanced way to apply aggregation functions is to apply it on the ciphertext of the data [36]. This uses properties of special encryption algorithms that match transformations on a ciphertext to the encrypted plain text data. This means the same transformations that were applied to the ciphertext are applied to the encrypted plaintext value, so after decryption the transformed value is obtained. This provides the advantage that no decryption key has to be distributed in the WSN to allow aggregation. This saves transmission bandwidth and keeps the number of keys distributed in a WSN at a low number.

In WSNs a broad range of attacks can be encountered. Besides brute force attacks like Denial of Service (DoS), where a network is flooded with artificial data and legitimate

requests get dropped because of the mass of other data, or jamming of the wireless medium also more specific attacks can be observed. We will discuss a selected set of attacks occurring in WSNs.

A *Sybil attack* is carried out by a node claiming multiple identities to the rest of the network [90]. This may be used in geographic routing where a node claims to be located in several locations and attract more traffic to himself thus enabling the node to monitor traffic traveling in different areas.

To execute a *wormhole attack* usually two colluding nodes in different areas of the WSN are needed. Then messages from one side of the network are forwarded via a low-latency link to the colluding node, where the messages are replayed. This attack is used to fake that these two areas are close to each other.

A *node compromise* is the basis of many other attacks. With a node compromise an adversary gains low level access to a sensor node. This may be done via physical access and a programming tool or via the network and the ability to inject code to take over the node. Compromising a node can have the goal to retrieve data stored on a node like a cryptographic key or to reuse the node's hardware and to reprogram the node with code from the adversary.

A *node replication attack* has the goal to clone an existing sensor node in the network. The cloned node is placed in a way that it can react faster than the original node to network requests. Then the new node can take over all the communication of the original one. The cloned node may damage the WSN by misbehavior or by retrieving confidential data addressed to the original node.

Therefore, we discuss common security problems and challenges in the next section and related cryptographic libraries and security frameworks are shown later in section 5.5.

#### 3.1.2.1 Problems and Challenges

There are several articles regarding problems and challenges in WSNs [96, 97, 135]. They all refer to the same basic problem classes.

First, the scarce resources. The limitations in computing power, storage and memory not only force the application code of a WSN to be small, it also mandates limitations to security algorithms, as there have to be enough resources left to operate the nodes' original tasks. Another important limiting factor is power consumption. Transmitting data or doing heavy computations pushes the energy consumption to levels where a battery may be drained after some hours, which demands light weight security algorithms.

The next class is unreliable communication. Unreliable transfer of data makes it mandatory to carefully include error handling to security protocols. As such protocols are well defined they have to cope with lost or crippled messages. Especially crippling of messages that overlapped while sent leaves protocol data easily with values that are out of scope of combinations that are possible by design. This problem may render a whole transfer as a fail if just one packet in between got changed, as it may

not be decrypted anymore. The denser the WSN the more vulnerable it is to this kind of problem. Also synchronization between nodes in a WSN is important when secure key distribution is used. If not all nodes in a WSN use the same keys, nodes with different keys may not be able to communicate with each other, as they are not able to decrypt messages of each other. In this case the network gets fragmented, even though the physical network is still connected.

The last one is unattended operation. Obviously unattended nodes are very vulnerable to physical attacks as an adversary can try to compromise a node with low risk of disclosure. Even remote management of nodes may not detect compromises, as modifications of sensors or similar cannot be detected but may influence the security of the network.

A big challenge is to find the right security algorithms for WSNs in general and to select a matching one for a given set of requirements. Most of the time it is all about trade-offs. Secure algorithms are either computationally intensive or introduce another security problem, e.g. ABE algorithms vs. symmetric encryption. Especially if there are specific functional requirements, like in our Scopes Framework, computational efficiency is not always possible and it has to be evaluated in which environment the combined system is feasible. Another challenge is to secure a WSN in a way that it still can carry out its original tasks on one side and on the other the multitude of possible attacks sketched above have been taken into account at design time. Also countermeasures for the strongest threats have to be included into the small program space of a microcontroller. Last and most important challenge for real-world deployments is the question of lifetime of the network. Most of the time this question cannot be answered reliably beforehand. In spite of this fact the WSN application has to be designed in a way that it can be tuned to comply with functionality and also expected life-time.

## 3.2 Models and Assumptions

### 3.2.1 Network and Trust Model

A WSN is considered that executes the Scopes Framework with the proposed security enhancements. It is composed of scope root nodes for accessing the network and many sensor nodes. Additionally at least one *trusted authority* must be part of the network on special actions, like in the event of a revocation of a sensor node, or in the bootstrapping phase to issue secret keys for the sensor nodes. As this trusted authority is holder of the network's master key it is offline if the WSN is in normal operation to keep this key out of reach of an adversary.

To ensure security of cryptographic keys on sensor nodes it is essential to take measures to prevent physical compromise of sensor nodes. If tamper resistant hardware is too expensive for the planned scenario, alternatives like tamper resistant packaging or similar may be possible. The measures to be taken also depend on the expected threat level.

### 3.2.2 Adversary Model

The adversaries are considered to attempt accessing a scope they are not authorized to access. The attackers may be external intruders or users of the network that are not authorized to access the target scope. The internal users have access to their root nodes, but none of them has an encryption key that complies with the target scope. The adversaries are assumed to have active and passive capabilities. In particular they can eavesdrop on all message traffic in the wireless sensor network. Attackers can collude with each other and, for example, exchange keys of their sensor nodes. In general it is assumed that attackers have an interest in keeping the network up and running.

### 3.2.3 Security Requirements

To secure a distributed scope, some common security aspects like data confidentiality and integrity, that are also desirable in other sensor network security schemes, should be provided. The following requirements were recognized in conjunction with access control in the Scopes Framework.

SR1: Secure group key exchange – The basis for group- or scope-level access control is to be able to securely exchange scope keys to establish a trusted group communication. This scheme should provide a secure exchange of a symmetric scope key to enable efficient cryptography between scope members.

SR2: Group-level access control – The scope-level access control extends the secure key exchange and adds access control. This ensures that only authorized sensor nodes can join a scope and exchange data. This supposes that the access control scheme provides a system to precisely describe the properties of potential scope members and define expressive attributes and special security levels for the sensor nodes.

SR3: Collusion resilience – As described in the adversary model, users of the sensor network may cooperate to access a scope without being authorized . Therefore, it is vital to include in the access control scheme mechanisms that prevent colluding unauthorized users from gaining an advantage over what they can get from individual attacks.

SR4: Forward secrecy – Managing the sensor nodes and to be able to exclude malicious or malfunctioning nodes is an important functionality in most scenarios. To support such a functionality the access control scheme should ensure that after revoking a malicious or leaving node it should not be able to join a secured scope afterwards, even though the access structure might be satisfied. Also the access to scopes it is currently a member of should not be possible anymore.

## 3.3 Secure Scopes Architecture

To secure computer systems there is the choice between two cryptographic methods: symmetric and asymmetric. Symmetric cryptography is efficient, but it is not secure to transmit the common keys via a wireless channel. Asymmetric cryptography on the other side is computationally more demanding, but is better suited for a distributed system with its public/private key-pair. Unfortunately in wireless sensor networks with radio communication as one of the largest consumers of energy a public key infrastructure (PKI) with all the exchanges of public keys imposes a huge burden on the sensor network. To achieve access control in the Scopes Framework we use the existing scope properties as criteria. Properties in combination with a public key system that may not need a PKI in its background led us to the attribute-based encryption (ABE) systems. They provide many features that we need, but for the cost of heavy computation. To reduce the computation effort we combine the security features of ABE systems with the light-weight symmetric cryptography. This is what we are looking for.

The CP-ABE algorithm [8] is based on elliptic-curve cryptography. Although elliptic curves were shown to be feasible in sensor networks [72], its application here results more expensive because CP-ABE also uses pairings on elliptic curves (an evaluation is shown in section 4.3.1.3). Pairings are computationally expensive. This means, it is too costly to encrypt data traffic directly with it. For this purpose, symmetric algorithms are better suited. Additionally, the AES algorithm is available as hardware module on nearly all sensor nodes that use wireless radio communication, e.g., [85, 154]. Therefore, we use CP-ABE for secure key exchange and access control to scopes, and the exchanged key is used for AES encryption to enable secure data communication between the scope members.

Having CP-ABE and AES as cryptographic primitives, leaves us with at least three keys. These are the public and secure key (PK/SK) of CP-ABE and the AES key. To differentiate them we call the AES key the *scope key* as this key is distributed among the members of a scope and secures the communication between them. This key is generated automatically and will not leave a sensor node in plain text. The other two keys are supplied by a task running on the sensor network. They are used to establish an access policy for a new scope (PK) and to be able to evaluate this policy on other nodes (SK), we still refer to them as public and secret keys. If there are multiple applications in the network there may be also multiple sets of PK/SK pairs.

The integration of CP-ABE and the Scopes Framework introduces changes in all three phases of the original framework and adds a new phase before WSN deployment. To differentiate between the original Scopes Framework and the security enhanced one, we refer to the latter as *SecScopes*. First, in the new bootstrapping phase all the necessary keys are generated and distributed to all the nodes before they are deployed. The second phase is the scope creation, where the scope definition is divided in static and dynamic properties. The static attributes are used to encrypt the scope key with CP-ABE. The

dynamic properties are encrypted using the scope key. The next phase is maintenance phase, where the scope is still maintained, but the scope keys are distributed and may be automatically refreshed with the regular scope refresh mechanism. Last, in the deletion phase it is ensured that the deletion is only triggered from an authorized node, while the timeout mechanism is still in place to help saving resources.

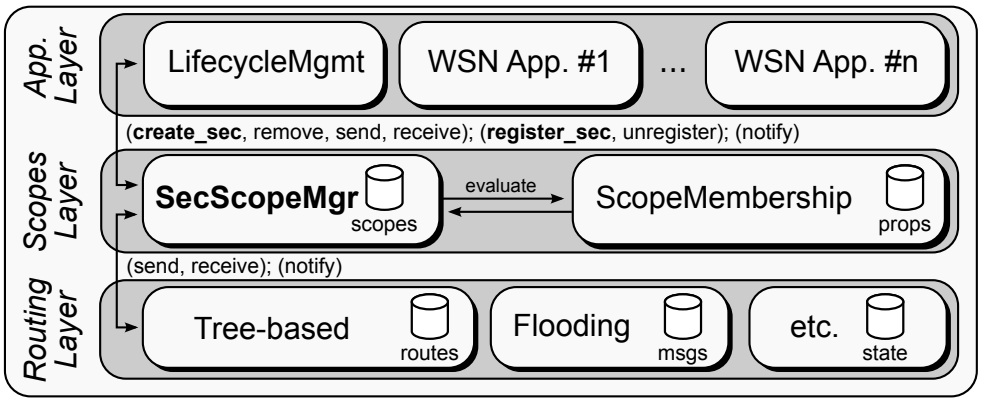



**Figure 3.2:** Layered Architecture of SecScopes

Now, for example, the scope definition for scope `ferdexMotion` from figure 2.4 on page 19 is separated into a dynamic and a static part. This means the terms of the acceleration sensor are included in the dynamic part and the other terms are in the static part. The conditions of the static part are then used to encrypt the scope key, where the dynamic part is encrypted using this key. Besides simple boolean terms the Scope Framework allows also the definition of mathematical functions or scenario specific functions to be used in the scope definition. To do so threshold gates are used, see section 3.1.1.3. These gates have a number of child gates and a threshold, e.g. `2of5`. Are more child gates satisfied than the threshold, the gate is satisfied. As long as the mathematical function , e.g. $<$ or $>$, can be expressed with a tree of threshold gates and they use static properties, they are included in the static part of the scope definition. If they cannot be represented as threshold gates they are included into the dynamic part. To ensure that the dynamic properties cannot just be ignored so called *placeholder properties* are introduced for each dynamic term in the static properties part. E.g. for scenario specific functions a term `HAS_FUNC_SCENARIO1` can be used or for sensors a `HAS_SENSOR_xyz` can be introduced. This way we make sure that only nodes that are able to evaluate the dynamic property part are able to decrypt and maybe join the scope.

An important feature for lasting security is the ability to renew the used scope keys. Every message encrypted using the scope key has an id attached that identifies the used scope key. With the scope refresh mechanism we can distribute a new scope key just by replacing the current key with a new one. For each scope two keys are stored, this enables a slow roll out of a new key and after some time switching the used key. This mechanism can be done automatically in predefined intervals. The interval length is a design parameter of the SecScopes Framework.

If a scope key is compromised by an attacker, rekeying in this way is still a secure way to exclude unauthorized nodes, as the new key is distributed encrypted by CP-ABE. It can only be extracted from authorized nodes. In the worst case scenario, where CP-ABEs master key is compromised, the only way to regain security is to generate a new master key and replace all public and secret keys on all nodes of the sensor network. This can be done in one-to-one connections, e.g., using ECDH which is part of TinyECC[3] [72] to establish a secure connection with a single node and exchange the secret key. This assumes an additional, temporary ECC key pair that has to be generated on all nodes in the network. But as the master key is kept outside the WSN at the trusted authority this scenario is assumed unlikely.

Another possible security breach is the compromise of a node's secure key. This allows the attacker to act as if he were the compromised node and may also join the same scopes. If such a compromise is detected it is important to revoke this specific secure key. To enable revocation in CP-ABE we adapted a mechanism proposed in [150] for a key policy attribute-based encryption scheme (KP-ABE), see section 3.1.1.3 for the original scheme. The principal idea is to update a parameter in all keys (including the MK) except the ones to be revoked. With this mechanism it is possible to revoke as many keys as needed in one step.

#### Revocation Algorithm

We have the given master key $MK$, public key $PK$ and the set of secret keys $SK_i$, as previously discussed in section 3.1.1.3, where $i \in N$ with $N$ the set of nodes in the network. To update the keys the trusted authority updates the $g^{\alpha}$ in $MK$ with a random $\alpha' \in \mathbb{Z}_p$ to $g^{\alpha'}$. For the $PK$ $e(g,g)^{\alpha}$ is replaced by $e(g,g)^{\alpha'}$ and

$$\Delta = \left(g^{\alpha'} \cdot (g^{\alpha})^{-1}\right)^{\frac{1}{\beta}} = g^{\frac{\alpha'-\alpha}{\beta}}$$

is calculated. To update all $PK$ and $SK$ the new $e(g,g)^{\alpha'}$ and $\Delta$ parameters are distributed among the group of nodes that should be authorized even after the key update. With the Scopes Framework this may be achieved by a scope excluding all nodes that are to be revoked. The nodes to be revoked can be excluded from the update scope because of a not matching access structure $\mathbb{A}$, e.g. over the set of valid node ids. Even a compromised node is not able to join this scope. To update an $SK_i$ its parameter $D$ has to be updated with the distributed $\Delta$ value the following way:

$$D' = D \cdot \Delta = g^{\frac{\alpha+r}{\beta}} \cdot g^{\frac{\alpha'-\alpha}{\beta}} = g^{\frac{\alpha'+r}{\beta}}.$$

$\Diamond$

After a node is revoked it is not able to join any scope anymore, as its old secure key cannot decrypt any new SCR. Also if other nodes get revoked and the node receives the

[3] TinyECC was ported to Contiki as part of this work.

$\Delta$ from the new revocation it is not possible to regain a working secure key, as the new $\Delta$ depends on the previous $\alpha$ value. But for the scope it is currently member of it can still en-/decrypt messages, as it owns the current scope key. To ensure *forward secrecy* the revocation mechanism is used in conjunction with the scope key refresh mechanism. As the node is not able to decrypt SCR and SR messages it is unable to obtain a renewed scope key and is not able to communicate anymore with scopes he was member before. As the node is now excluded from the network and the scopes it was member of, it can still act as a regular intermediate node.

Talking about intermediate nodes. Besides the already mentioned root node of a scope and its members there are two other kinds of nodes. The intermediate nodes, that are not members of a scope, but are used to route messages; and nodes that do not participate in a scope at all. The nodes that do not participate will not route any messages and so are passive. Intermediate nodes are used for routing and get to see many messages of scopes. But as they are not members they are not able to decrypt messages as they do not own the valid scope key. This also excludes the possibility for these nodes to aggregate values on the way through the network.[4]

Colluding users are in general a threat to crypto systems, but as the security architecture here is built on top of the CP-ABE algorithm, SecScopes are not susceptible to collusion attacks as CP-ABE is not [8].

Considering our container harbor scenario from section 1.1.2 the question arises, who will be the *trusted authority*? As SecScopes supports per task public and secret keys each company using the WSN could be its own trusted authority, issue keys and define the possible properties. As the keys need some storage space it would make sense to have only one key per sensor node and the different companies just issue new properties to be used on a node, which they can then use with their scopes and tasks. There are publications in this direction, like [88], but they impose much more computational load than the architecture described here and therefore are currently not feasible. Therefore, here a single trusted authority is assumed with a common set of properties that can be issued to the sensor nodes.

A single trusted authority helps deploying new tasks, as it can make sure that only tasks from authorized parties, e.g. logistics companies, are allowed and installed on the nodes. For this purpose a module to manage the life-cycle of installed tasks is available on all sensor nodes. To deploy tasks to a subset of nodes a scope with the Life-Cycle Manager task (LCM) is created and the task then distributed. This ensures that the task is securely transmitted and a cryptographic hash value ensures that the task was not altered or no transmission errors occurred. With the LCM, tasks can also be deleted or updated.

[4] The Scopes Framework is currently not supporting aggregation, but we have shown in [57] that it is possible to extend scopes to do so.

## 3.4 Integration in Scopes

With the integration of security into the Scopes Framework, the major concern was to preserve the distributed and modular capabilities of the framework. Therefore the integration focused on the scopes manager module and the application interface. This way it is possible to use all provided security measures independently from scenario specific modules like membership conditions or the routing. Keeping the interface to the routing modules eases the integration and supports the usability of scopes, as existing routing algorithms don't have to be adapted and new algorithms just have to include the existing interface.

The changes to the task interface are also limited. The task registration was extended to supply the needed cryptographic keys for CP-ABE. An additional parameter was added to the scope creation to supply the access policy for the newly created scope. With these minor changes a task can utilize the full security functionalities of scopes.

The scopes manager module was subject to extensive changes. Besides the straightforward interface changes described above, the management structures were extended to include management data for the encryption, namely CP-ABE keys, scope keys and status information. The data transmission was enhanced by an encryption and decryption mechanism. It ensures that, if a valid scope key is present, the sent data is encrypted using the available AES encryption in counter mode (AES-CTR) [30], which is obviously also used for decryption. Most changes were done in the scope creation and refresh mechanisms, as here the access control to a scope, scope key exchange and key refresh system are integrated. More details on the implementation are provided in section 4.3.2.1.

Performance is discussed in section 4.3, but some precautions for good performance can be taken in advance by introducing the following design decisions, especially as the computation of CP-ABE encryption and decryption is a major issue. To keep the computational overhead to a minimum we trade it off for storage space. Since for each scope refresh the same encrypted scope key is disseminated (unless the key is refreshed) we store the decrypted scope keys in memory to save further computation costs. Additionally, the dynamic scope definition part is saved in plain text to simplify the repeated evaluation of the dynamic properties.

The following subsections will detail the newly added bootstrapping phase and the extended creation, maintenance and deletion phases.

### 3.4.1 Bootstrapping Phase

Bootstrapping is performed before the sensor node is deployed. In our scenario this would be before the container is shipped, e.g., while it is loaded with goods, information is updated on the node. First, the master key MK and the public parameters PK are generated at the trusted authority, if not already present. Then, a secret key SK is gen-

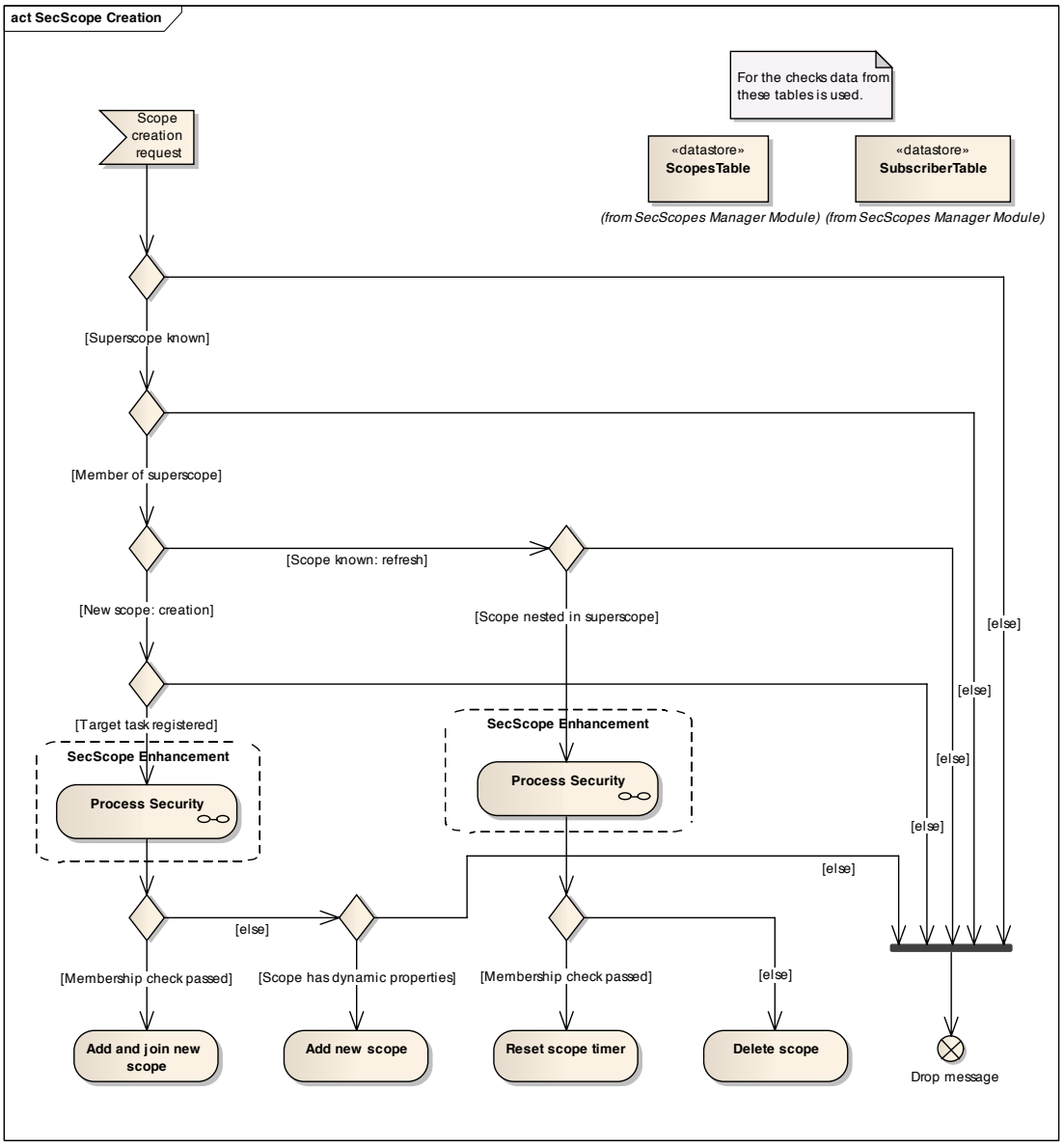



**Figure 3.3:** Scope Creation Workflow – Security Enhancements

erated for the container node with suitable properties. Here we would have properties like `company` or `containerType` or types of sensors that are available. The SK and PK are then stored on the node, ready for deployment. If the SK has to be installed on a node in an untrusted environment this may be achieved using ECDH [8] to establish a common key to encrypt the transmission.

This phase enables the establishment of a secure system in a trusted environment before the network is deployed. After initializing security it is now possible to establish secure connections and apply the access-control mechanisms. As multiple tasks with different sets of properties are possible in a WSN, each task may get its own secure/public key pair. This is also important to support multiple trusted authorities, but occupies more resources. Through the task registration at the Scopes Framework the SK and PK are made available.

### 3.4.2 Creation Phase

The scope creation procedure remains mostly unchanged, see figure 3.3, but the SCR message is altered. Before, it was mainly the binary representation of the scope definition, with some additional fields. Now there are two parts. First there is the encrypted scope key (encrypted with the specified access policy) for the symmetric encryption. And second, the dynamic portion of the scope definition and the additional fields encrypted with the scope key. The access policy reflects the static properties specified in the scope definition extended by static placeholders for the dynamic properties. This ensures that only nodes that can extract the scope key can access the full scope definition and so join the scope.

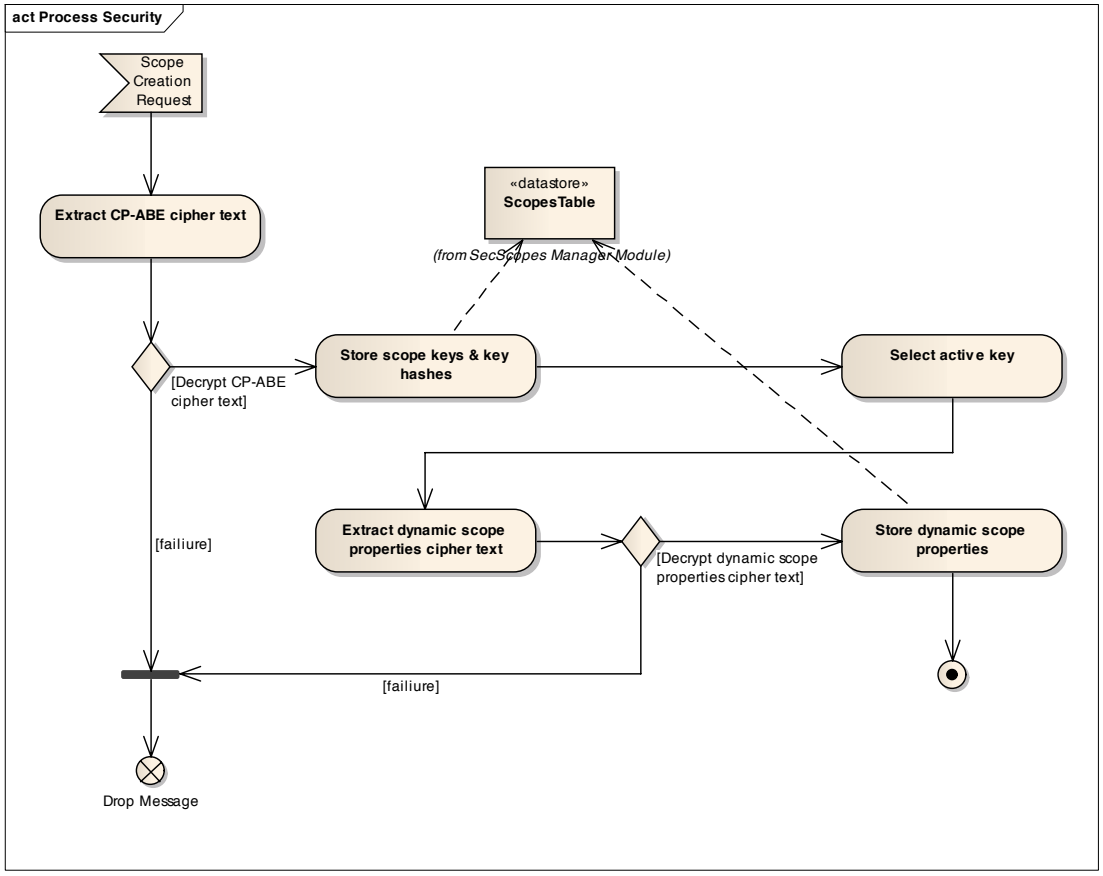



**Figure 3.4:** Security Processing Workflow

### 3.4.3 Maintenance Phase

The scope refresh messages that are used to keep a scope alive are similarly structured as the SCR, so they undergo the same changes. The simple scope refreshes that don't contain any payload and just serve the purpose of keeping the scope alive are not changed. Additionally, the processing of an SR is extended by the scope key refresh mechanism. Here the IDs of the scope keys are added with the ciphertexts. This way

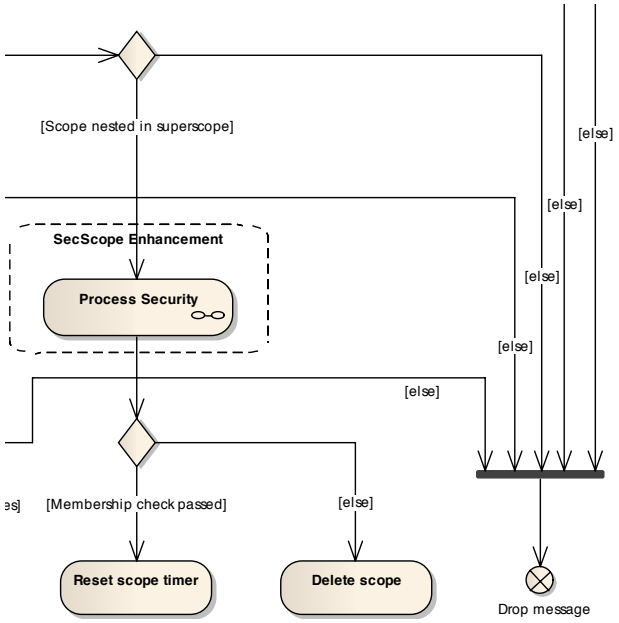
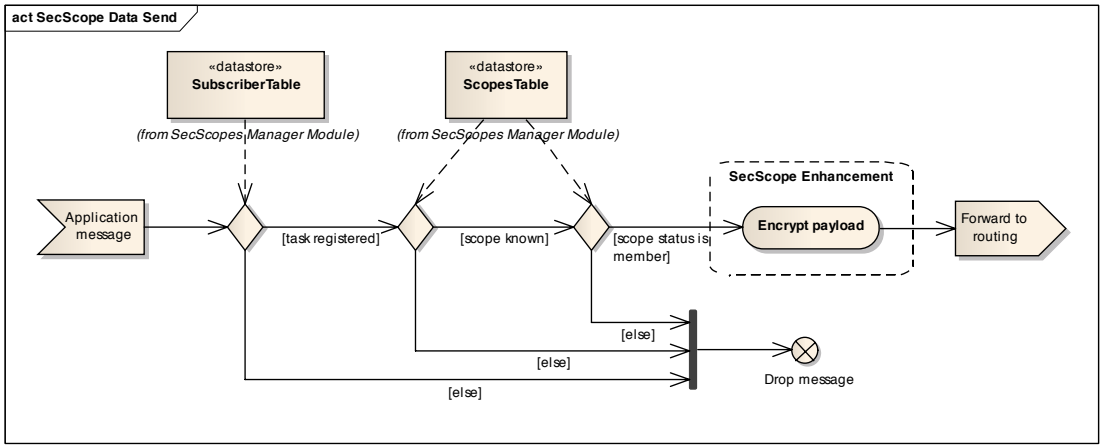



**Figure 3.5:** Scope Maintenance Workflow – Security Enhancements

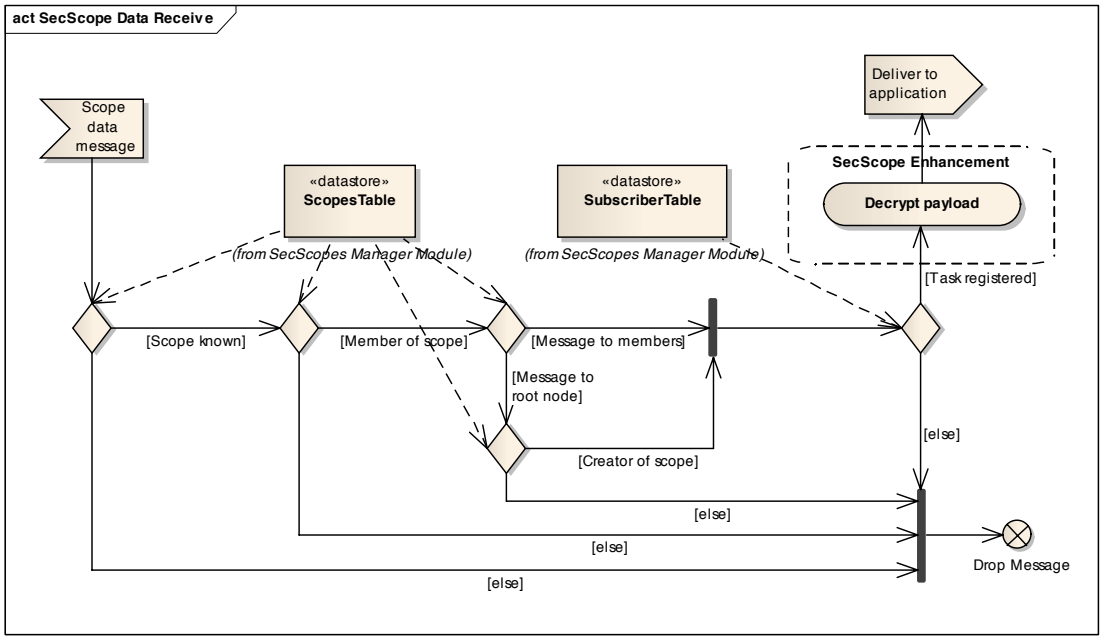
it can be decided in advance if new scope keys are distributed and it is necessary to decrypt them. This is shown in detail in section 4.3.2.1.

### 3.4.4 Deletion Phase

The deletion of a scope is not allowed to be triggered from outside the scope. Therefore, the root node of the scope to be deleted encrypts the scope id with the scope key. This way all members of the scope can ensure that the deletion was triggered from an authorized node by evaluating the ciphertext. Deleting a scope from a scope member other than the root node is possible, but it does not allow a proper deletion of the scope, as only member nodes are reached that are children of the deleting node. The root node will refresh the scope that was deleted in the next cycle and reestablishes the scope. With the current architecture this cannot be efficiently prevented as every secret that the scope root may send to the scope members has to be known by the member.




(a) Sending.




(b) Receiving.

**Figure 3.6:** Scope Data Exchange Workflow – Security Enhancements

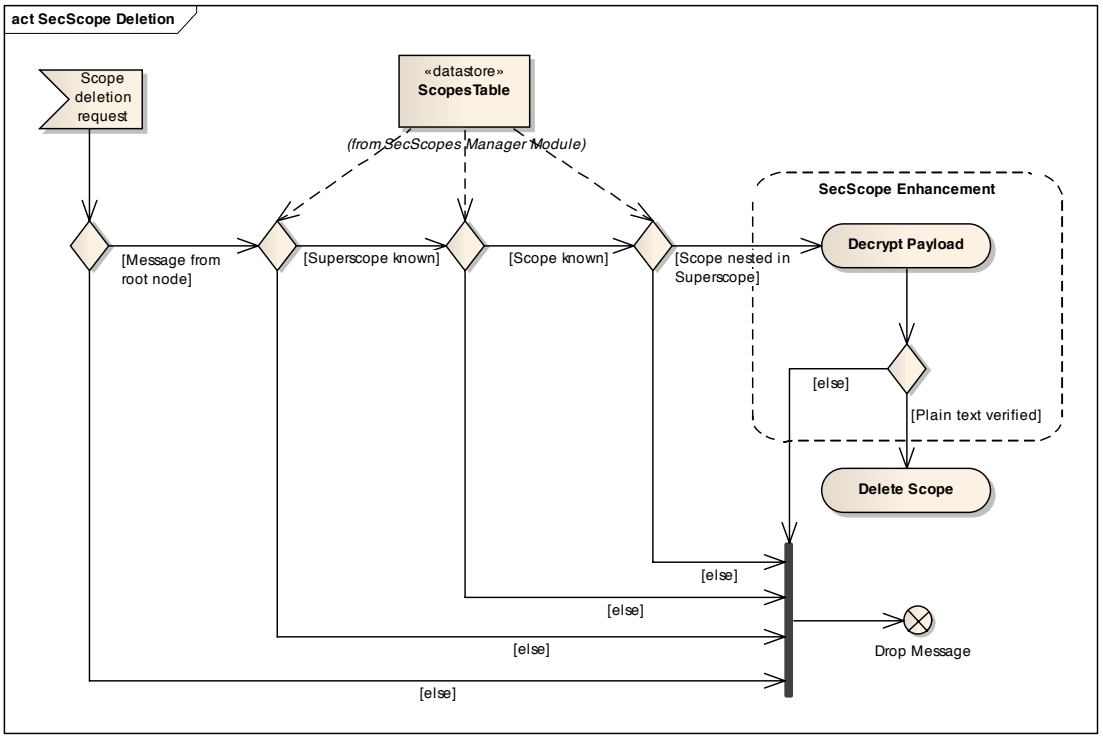



**Figure 3.7:** Scope Deletion Workflow – Security Enhancements

# 4 Evaluation

## Contents



For the evaluation of the Scopes Framework and SecScopes two prototype implementations were used. The first one was the result of a diploma thesis [53] and was implemented for a simulator. It was optimized and extended to be the first incarnation of the Scopes Framework to be run in a real WSN. Utilizing the lessons learned from the first prototype that relied on the SOS operating system, we built a new version from scratch for the Contiki operating system as we reached the limits of SOS with the first implementation. The Contiki based implementation was extended to the SecScopes Framework. Working with these different implementations made it reasonable to have a separate implementation sections for each of them. They are located in the different evaluation sections. The results of all systems are gathered in a real life testbed.

## 4.1 Testbed

The testbed used in this work is deployed at the Piloty building, Technische Universität Darmstadt. The TUD$\mu$Net-testbed [42] is built with 32 TelosB sensor nodes on floor 1 and was extended to a second floor and an additional area at the TIZ building lab of GKmM[1]. All sensor nodes are connected to a router backbone network. We used only floor 1 of the testbed, as it is the largest connected area, see figure 4.1. During the time we used the testbed it was constantly upgraded.

While doing the evaluation of the SOS implementation, the testbed in its first deployment had 30 TelosB nodes that were only connected to an active USB-hub to power the nodes and to be able to program them on a per room basis. The logging was

[1] DFG Research Training Group 1362: Cooperative, Adaptive and Responsive Monitoring in Mixed Mode Environments (GKmM); `http://www.gkmm.de`

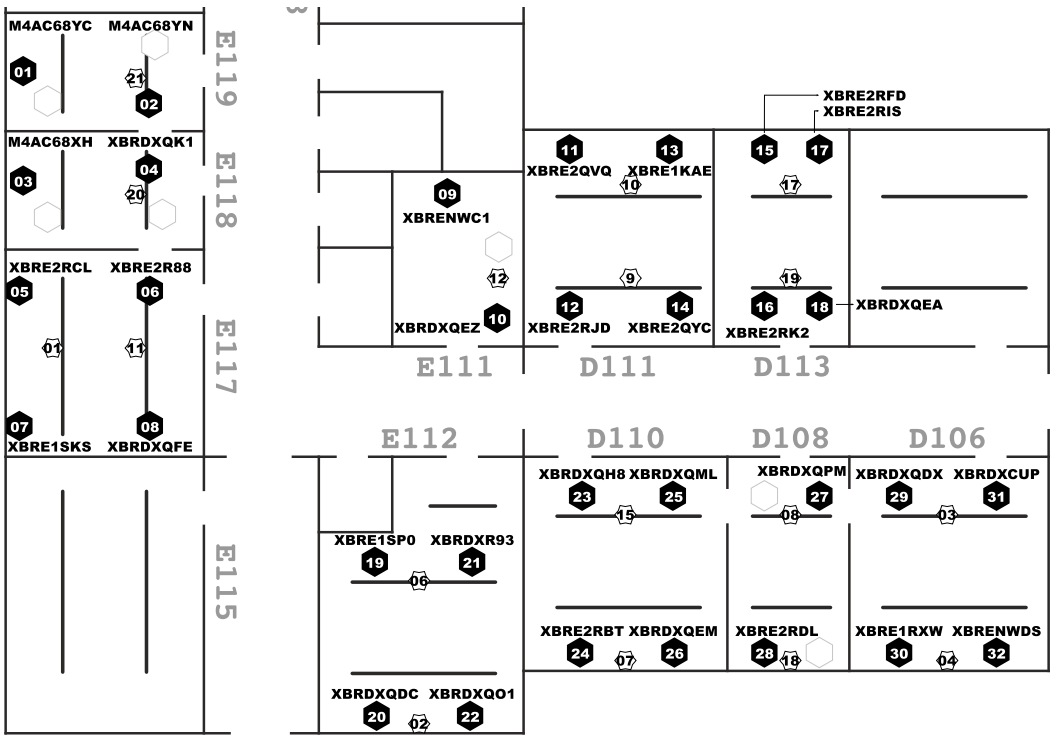



**Figure 4.1:** Piloty building testbed

implemented on the nodes and the log files were retrieved after a test run by downloading them over the 802.15.4 wireless connection. To be able to compare the logs from different nodes we implemented a simple time synchronization.

Taking a timestamp network wide of each nodes' local time before and after a test run, the different node local times can be translated to a test run time. This copes with the different node local times and removes problems with small deviations in the nodes' timer accuracy. The nodes were deployed in the same places as shown in figure 4.1, here the full testbed is shown. The Testbed used for the first evaluation can be seen in figure 4.2, nodes 1 and 2 were not used here and the corresponding room is omitted.

All other tests with Contiki were run on the full testbed deployment using not only 32 nodes, but also a router back-bone network with a modified version of the MoteLab [141] software. This version supports our specific network infrastructure and is able to deploy Contiki code to the nodes in the network. The rest of the functionality, like logging of output on the serial port and test scheduling is the same as the original.

This means the serial log data of a node is output via USB to the connected router. From there the data is forwarded to a central DB server that stores the log entry and assigns a timestamp. To transfer the data a separate network segment is used to avoid influences in log data transmission. Besides that we have not experienced any difficulties regarding the log functionality. And as we just need correct relative timestamps, this is a feasible solution.

## 4.2 Scopes Framework

In this section we will compare the SOS and Contiki versions of the Scopes Framework.

We use abbreviations for the different routing algorithms used in the evaluation. The common *flood routing* is referred to as FLD. *Gradient-based routing* which is used in the SOS implementation is referred to as GBR. And *Selective flooding, unicast return (SelfUR)* used in the Contiki build is referred to as SEL. Details of the routing algorithms will be shown in the different implementation sections.

### 4.2.1 SOS Operating System

We built the first proof-of-concept of the Scopes Framework using the SOS Operating System [44]. This was the first operating system to enable runtime code distribution and loading, a feature other operating systems did not support at the time. This enables distributing and loading of modules over the air once the sensor network is rolled out. Additionally, SOS provides the abstraction of modules and allows modules to register functions. So, every module can provide methods to other modules and ease the implementation of APIs between conceptual layers. SOS offers two inter-module communication mechanisms: message passing and registered methods. Where message passing is arbitrated by the OS scheduler (asynchronous processing) and registered methods, like callback functions, are directly called and blocking the further processing until the sub-routine is finished (synchronous processing).

#### 4.2.1.1 Implementation details

The SOS version of the Scopes Framework evolved from the proof-of-concept version developed in [53]. This version was written only for simulators and was not functional on a real sensor network. Also, it was missing some integral parts of Scopes, like dynamic scopes, scope hierarchies or a scope definition language. To get the code working in a real environment a big part of the code had to be reviewed and partly rewritten. The problems solved range from alignment issues of memory structures on the 16bit microcontroller to problems with the timing of sending/receiving messages in the routing modules.

The SOS version of the Scopes Framework already includes the most important features related to scopes that are found in the current Contiki version. First, this is the layered and modular architecture that makes it easy to adjust modules to the given environment using defined interfaces between the modules. The scopes could also form hierarchies to structure the network by using only static properties. Using sensor data as a property is possible, but it is not evaluated further if the values change, which gives it a static character. The properties can be expressed using a declarative language that is defined in section 2.4.1. The definition of a scope is then parsed to a pre-order

byte representation that is compact and can be easily exchanged between sensor nodes. This version also includes the small and full refresh mechanism explained earlier to save some bandwidth. For a regular scope refresh message this saves only some bytes of the scope definition, but for a scope with the security enhancement, this can save some hundred bytes per refresh cycle. As routing algorithms we have implemented a simple flooding algorithm as a baseline for our tests and a more sophisticated multi-hop routing that uses a tree structure. The tree routing is an extended gradient-based routing [114] which is based on directed diffusion [51].

The general approach of the extended gradient-based routing is the same as directed diffusion. There are request, data, acknowledge and reject messages. A request is sent during the first phase of the algorithm to announce the scope in the network. Therefore, the refresh is flooded through the sensor network or the super scope of the new scope. The transmission of the request messages is also used to build the routing tree for the direction from the member nodes of a scope to its root nodes by storing the node ids of where another node got the request message from. Up to 10 parent node ids can be stored. If a node matches the scope property it replies with an acknowledge message. So the parent scope knows which nodes are its children nodes. Each node can also have up to 10 child nodes. This and the former limit are parameters that can be changed at compile time. Is a child node sending an acknowledge to a parent that already has completed its limit of child nodes, the parent node answers to the acknowledge with a reject. In this case the child node tries to send an acknowledge message to the next entry in its parent node list and removes the node that rejected it. With the limits of the child and parent lists we restrict the maximum memory size used, which is very important on the constrained sensor nodes, and allows some kind of load balancing. After this mechanism completes, data can be exchanged via the scope. To manage the scope memberships over time the refresh mechanism was also used here.

#### Available framework APIs.

APIs were defined to be able to interact with the Scopes Framework. Besides the most important one, the application API, there are also APIs for routing algorithms and for the scope membership evaluation. First we show the application API:

```
1  void send_scope_create(parentScope_id, scope_id, propLen, properties);
2
3  void send_scope_data(scope_id, sendDirection, srcApp_id, destApp_id,
4                       msgType, payloadLen, payload);
5
6  void send_scope_remove(parentScope_id, scope_id);
```

**Listing 4.1:** Scopes Framework application API (SOS version).

To create a scope, applications call the `send_scope_create(...)` method. The `ScopeMgr` module sends a scope creation message to all potential scope members: if it's a new top-level scope, it is sent to all nodes; otherwise it is only sent to members of the parent scope. Whenever a node becomes a member of a scope, this is signaled to applications that have registered to receive this information. Once a scope has been created, a bidirectional data channel between the scope root and its members was established, which can be used by invoking `send_scope_data(...)`. During the lifetime of a scope, refresh messages are sent in regular and configurable intervals to allow automatic maintenance and to keep the scope alive (this mechanism is explained in detail in section 2.5.2). After finishing its task, an application can delete its scope by calling `send_scope_remove(...)`. A removal message is then sent to the member nodes. In addition, applications can register for notifications about the scope membership changes. These are asynchronously delivered to them, to avoid polling.

A received membership definition is forwarded by the scopes manager to the membership module. The membership module can be replaced by custom ones as mentioned earlier. The API used to communicate with the module is shown below.

```
1   bool check_scope_membership(scopeDefinition, length);
```

**Listing 4.2:** Scopes Framework membership evaluation API (SOS version).

The definition in binary form is handed over to the membership module by calling its `check_scope_membership(...)` method, including the definition's length. The membership module then parses the data and decides if the node fulfills the definition or not. Depending on the result a boolean is returned to the scope manager module.

The routing API, listing 4.3, is composed of two parts, the communication part and the scope status signaling.

```
1   void scope_send(scope_id, parentScope_id, sendDirection, destApp_id,
2                   destMsgType, payloadLen, payload);
3
4   void scope_inserted(scope_id);
5
6   void scope_removed(scope_id);
```

**Listing 4.3:** Scopes Framework routing API (SOS version).

In the process of scope creation, management and deletion the membership state of a node can change. A node may join or leave a scope or the scope itself is deleted. To be able to save resources and tell the routing mechanism if it has to maintain a route or not, two methods were introduced in the routing API: `scope_inserted(...)`

is called if a node joins a scope; if the node leaves a scope or the scope is deleted `scope_removed(...)` is invoked by the scope manager module.

The third method of the API is `scope_send(...)`. This one is called for every message to be sent to a scope. It makes no difference what kind of message it is, SCR, SR, data or a deletion message.

### 4.2.1.2 Methodology and Setup

To evaluate the Scopes Framework we have conducted numerous experiments on our live testbed. The Piloty-building testbed comprises 30 TelosB nodes, powered through USB hubs (see Figure 4.2), distributed over 9 offices and spanning 544m$^2$. In each room, nodes are located either next to the windows or above fluorescent lamps. The building's thick walls greatly reduce radio ranges, forcing a multi-hop behavior even at the maximum transmission power level.

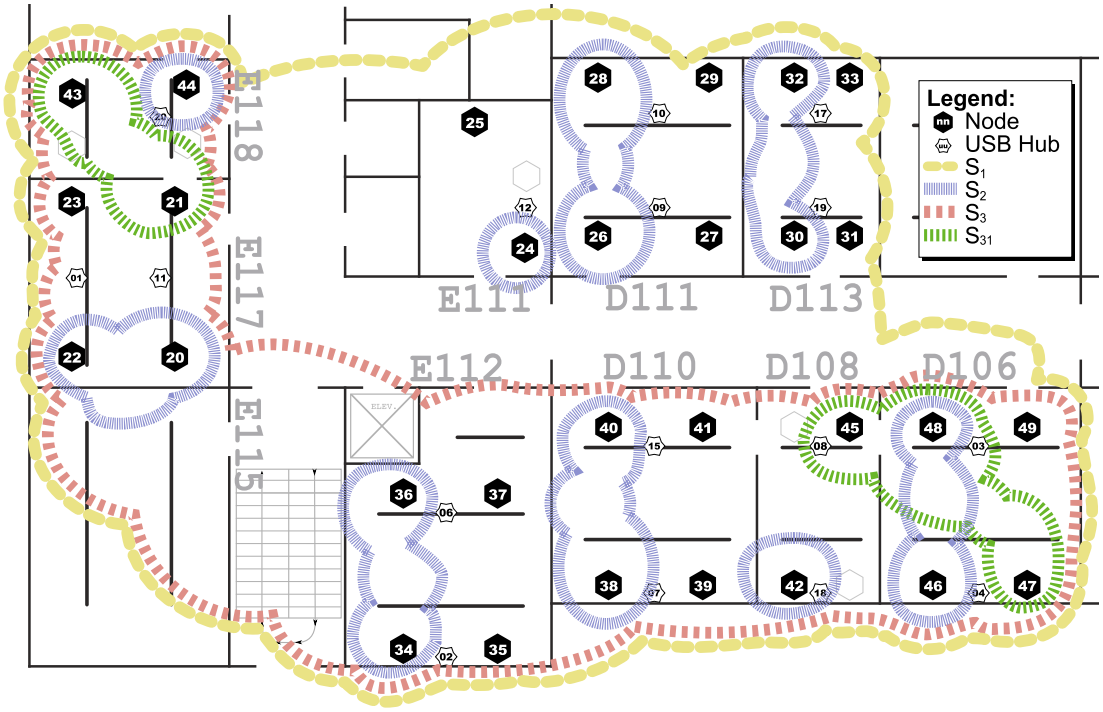



**Figure 4.2:** *Piloty* building testbed and scopes used for evaluation on SOS

**Scopes.**

In our tests we used the scopes depicted in figure 4.2. These represent reasonable, real-life scopes. The scope $S_1$ covers all nodes. Such scopes are necessary, e.g., for massive software updates. Scope $S_2$ emphasizes that membership is not necessarily given by physical proximity, instead scopes build overlays. Finally, $S_3$ and $S_{31}$ illustrate nested scopes. These occur in any hierarchical relation, for instance, in domain organization. The scope definitions were based on operations on node IDs, although as explained

before, other relations over node properties such as location could have been used. Table 4.1 summarizes the scopes that were used. The last column presents the size of scope definitions and the entire creation message. These highlight the compactness of the representation. Lastly, all scopes were created at the network boundary, namely node 43 (top left). This forced a minimum of 2 hops and an average of over 3.

| Scope | Description | Parent Scope | Size | |
|---|---|---|---|---|
| | | | Scope Def. | OS Message |
| $S_1$ | All nodes | *world* | 4 bytes | 18 bytes |
| $S_2$ | Nodes with even id | *world* | 74 bytes | 88 bytes |
| $S_3$ | Nodes whose $id < 24 \mid\mid id > 33$ | *world* | 9 bytes | 24 bytes |
| $S_{31}$ | Node id $\in \{21, 43, 45, 47, 48\}$ | $S_3$ | 24 bytes | 38 bytes |

**Table 4.1:** Summary of employed scopes

**Distributed Logging.**

In order to track node activity in each experiment, we have extended the framework with a *logging* module. Its main task is to log events as requested by other modules. It registers a function at the OS's kernel:

```
1  void log(uint16_t scope_id, uint8_t event, uint8_t size, uint8_t data*);
```

**Listing 4.4:** Scopes Framework log API (SOS version).

The parameter `scope_id` indicates the scope to which the log entry will be associated, whereas `event` is the concrete event to be logged. A set of events was defined which includes `SCOPE_CREATED`, `SCOPE_DELETED`, and `SCOPE_DATA_RECEIVED`. If additional information needs to be logged, the parameters' `size` and `data` can be used. Modules that need to log events register to the function to invoke it.

Once invoked, the function obtains a timestamp from the OS and buffers a new log entry before storing it in the data flash. There are two reasons for storing log entries in the flash. First, in many cases, due to the length of our experiments, log data didn't fit in RAM. Each TelosB has only 10KB of RAM, but 1MB of data flash. Second, making log data persistent greatly helped debugging our code when nodes crashed while debugging. Since SOS didn't provide a driver to access the data flash, we had to implement one. The module can be queried wirelessly for the logged entries. Queries can require actual entries or a full dump. This might trigger the transmission of hundreds or thousands of messages, thus retries were implemented to better tolerate message failure.

To make sense out of each node's logged data, log entries must be time synchronized with each other. Instead of adjusting all clocks to a common time base, we let clocks run unsynchronized. At the beginning and at the end of each experiment, we apply unidirec-

tional synchronization [109] from a mobile base station acting as reference clock. This station sends its local time, which gets logged at each node. Once data is collected after an experiment, these two reference timestamps are used to shift and scale all logged entries accordingly, indeed transforming all of them offline to a common scale.

There are two important aspects to the framework, namely *reliability* and *efficiency*. In the next subsection we discuss how reliable the scope mechanism is and its relation to the underlying routing protocol.

#### 4.2.1.3 Performance Evaluation

**Scope Creation, Removal and Maintenance**

We first investigate the two main scope operations, *creation* and *removal*, together with the scope *maintenance*. Consider the scope membership described by a function, $\phi(t)$, that outputs the percentage of nodes which belong to a scope out of an expected set at a given time $t$; $\phi\colon \mathbb{Z} \to [0,1]$. The ideal membership $\phi_I(t)$ is a function of time which equals 100% when the scope is alive and 0% otherwise, this resembles an instantaneous scope creation and removal. The values observed in practice, $\phi_R(t)$, lag behind the ideal values, i.e., once a scope is created it takes time for nodes to reach a high membership percentage, and later when a scope is removed it takes time to drop down back to 0%. This is natural and due, for example, to message propagation delays and medium loss.

**Definition 4.1.** When multiple test repetitions are considered, we refer to $\phi_{R_{max}}$, $\phi_{R_{avg}}$ and $\phi_{R_{min}}$ as the *maximum*, *average* and *minimum* membership values, respectively.

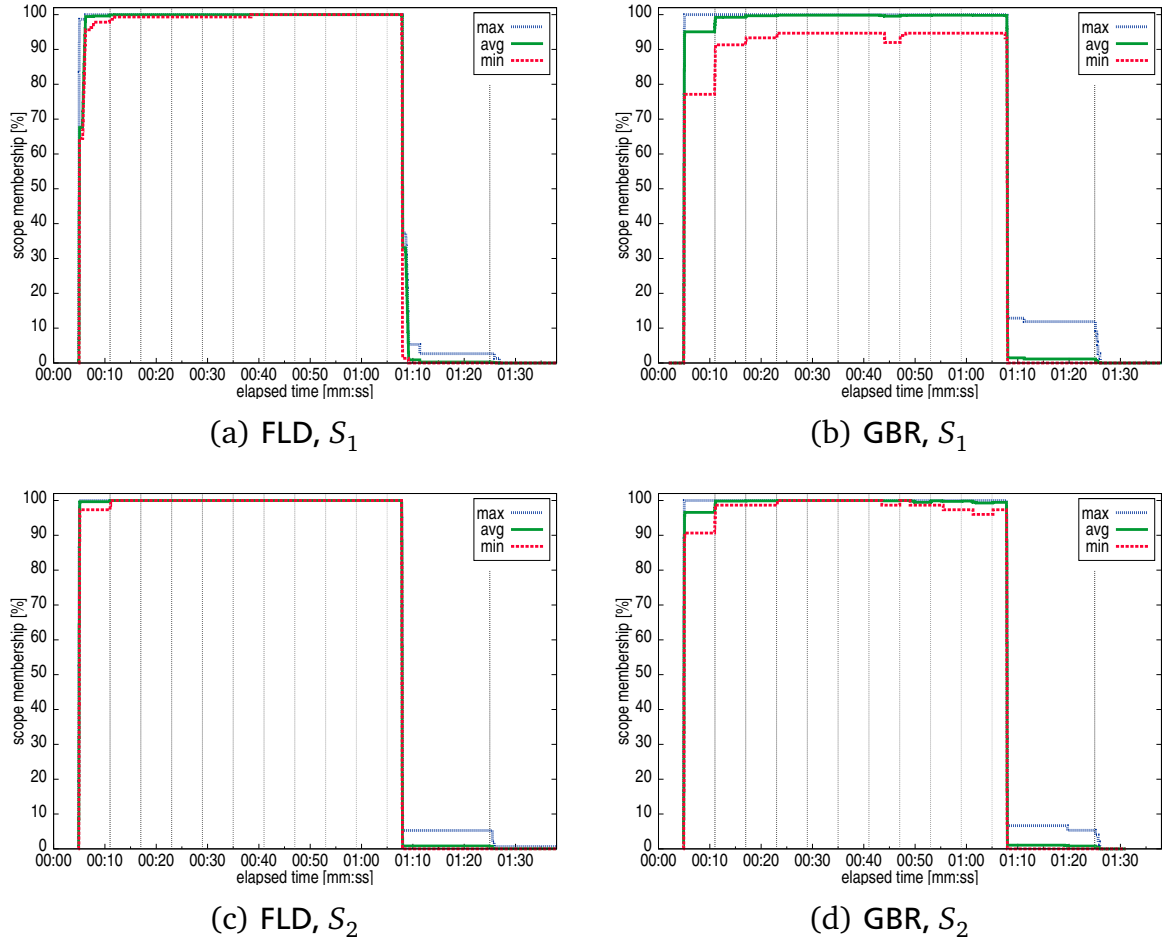
Scope reliability is described by the following parameters:

**Definition 4.2.** The *reliability value* stands for the deviation of the average measured membership values from the ideal ones. To quantify the reliability value, the area of the curve $\phi_{R_{avg}}$ is calculated and compared to the area of $\phi_I$. The closer the value to 1 is, the closer $\phi_{R_{avg}}$ is to $\phi_I$, thus the higher the reliability value, with 1 reflecting the ideal curve.

**Definition 4.3.** The *stability* of the scope mechanism indicates its tolerance to network dynamics. This is quantified by calculating the area between the curves $\phi_{R_{max}}$ and $\phi_{R_{min}}$. The closer the value is to 0, the more stable the memberships are.

**Definition 4.4.** The *quickness* in achieving an expected membership percentage is important. We quantify the *creation* and *removal delay* by measuring the time it takes for $\phi_{R_{avg}}$ to reach >98% after a scope creation, and to 0% after a scope removal, respectively. Clearly, the lower these values, the better.

In Figure 4.3 we characterize the reliability of the framework for $S_1$ and $S_2$ (the behavior of nested scopes is evaluated in detail later). The figures present $\phi_{R_{max}}$, $\phi_{R_{avg}}$ and $\phi_{R_{min}}$ as defined above. A scope is always created at t=0:05 and remains alive for 63 seconds, thus, $\phi_I(t)$ is 100% between t=0:05 and t=1:08, and 0% elsewhere. This




(a) FLD, $S_1$ (b) GBR, $S_1$

(c) FLD, $S_2$ (d) GBR, $S_2$

**Figure 4.3:** Scope creation, removal and maintenance for FLD and GBR

scope lifetime allows for 10 refreshes at a period of 6 seconds (indicated with vertical dashed lines drawn at regular intervals). The rightmost dashed vertical line indicates the last point at which all nodes should automatically remove themselves from the scope in case the explicit scope removal message wasn't heard. This occurs 20 seconds after the last refresh (t=1:25).

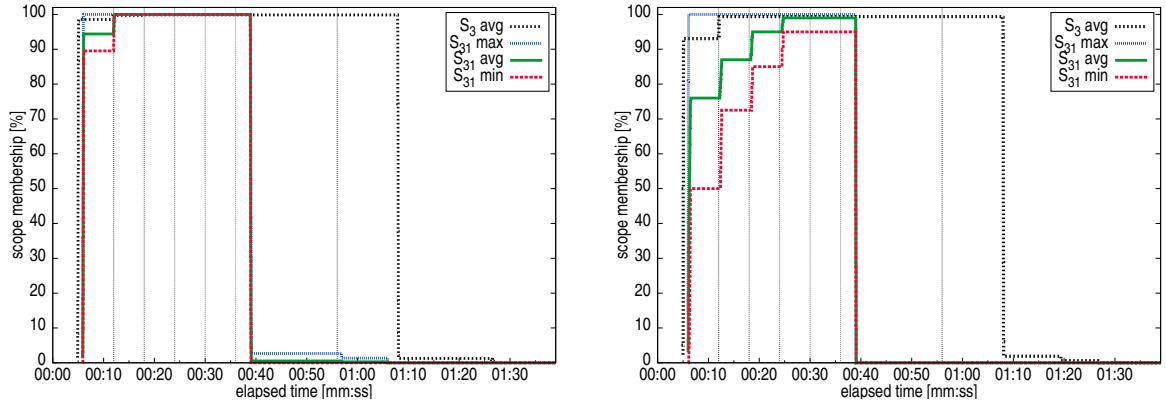
The plots for $S_1$, 4.3(a) and 4.3(b) show that the creation delay was quite low: it occurred immediately with Flooding, further called FLD, while it took 3 refresh periods with gradient based routing, further called GBR. For both algorithms, the scope membership was kept up at around 100% until the explicit removal was requested. Later, once a scope was removed, almost all nodes heard the explicit removal message. Virtually no nodes resorted to the lease expiration timer with FLD (0.26%), while for GBR it was below 5% – an acceptable value. The removal delay of GBR was thus higher than FLD's (only after the lease expires did nodes automatically release resources). While the reliability values were almost the same with 0.9938 for FLD and 0.9937 for GBR, FLD's stability (0.0209) was better than GBR's (0.0632). As mentioned before the values for reliability and stability are the deviation to the ideal or min/max curves. The plots for $S_2$ exhibit a slightly lower reliability value. The scope creation delay worsens for FLD,

requiring 1 refresh period, and GBR showed an unstable scope membership percentage. The results show that both approaches are viable in practice.

**Nested Scopes.**

We now evaluate the behavior of hierarchical relations among scopes. In these tests, we created scope $S_3$ at $t_1$, and one second later created subscope $S_{31}$. While $S_3$ remained alive (as before) during 63 seconds, $S_{31}$ remained alive for 33 seconds, which allowed for 5 refreshes to be issued.

Figure 4.4 presents the respective reliability results for FLD and GBR. Again, while both protocols reached high scope membership percentages, GBR (99%) was slightly lower than FLD (100%). The reliability values were high in both cases: FLD achieved 0.9949, while GBR got 0.9705. Also, FLD was more stable than GBR (0.0178 vs. 0.0889). The scope creation delay was 1 refresh period for FLD and 3 for GBR, whereas the removal delay was 20 seconds for FLD and 0 seconds for GBR. These results point out that nested scopes exhibit similar reliability properties to those of top-level scopes.




(a) FLD (b) GBR

**Figure 4.4:** Scope creation, removal and maintenance for nested scopes $S_3$ and $S_{31}$

**Network Density.**

We now discuss the effects of network areal density $\rho_A$[2] on scope operations and maintenance. Particularly, we consider the effects of density with respect to the reliability of the scope operations. This is of importance because not all setups will have the same density as ours, or even a constant one.

There are two main effects that influence the reliability. On one side, the more nodes there are, the higher the redundancy available to build the underlying scope overlays, thus the reliability increases. On the other side, the more nodes there are, the more collisions can occur, which decreases the reliability and stability.

[2] When we refer to density the areal destiny is meant.

| $\rho_A$ | #Nodes | Nodes per area |
|---|---|---|
| 4 | 30 | $1/18.1m^2$ |
| 2 | 18 | $1/30.2m^2$ |
| 1 | 10 | $1/54.4m^2$ |

**Table 4.2:** Node densities

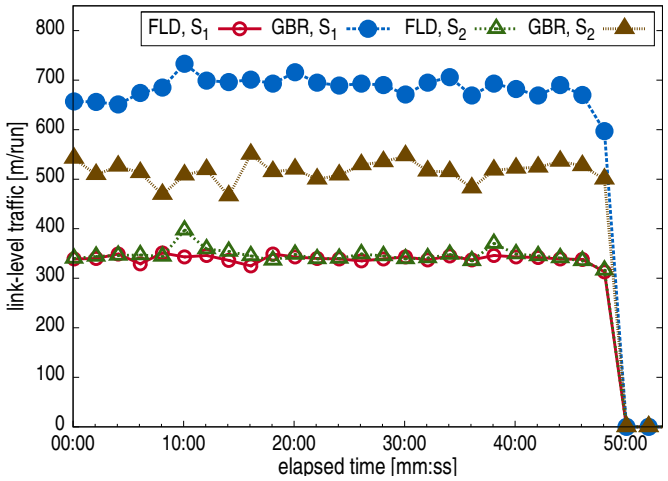



**Figure 4.5:** Scope operation costs

We inspected this issue by repeating all of our tests with three node density values. These configurations are presented in Table 4.2, with the resulting density in nodes per $m^2$. The amount of nodes per room is indicated with $\rho_A$.

In general terms, we can say that the denser the network, the higher the reliability values. Also, it can be observed from the previous figures 4.3 that $S_1$ is more reliable than $S_2$.

**Scope Operation Costs.**

The energy costs of the scope operations are largely due to messages sent and received. Evidently, energy efficiency depends on the routing mechanism chosen. We studied the operation cost in isolation; in a typical use these become marginal compared to data exchange. Figure 4.5 plots link-level traffic for $S_1$ and $S_2$. Each measurement point represents one test run (2 minutes), there were 25 runs. The curves confirm the expected routing algorithm behavior. The behavior exhibited by FLD is deterministic, since the decision whether a node is member of a scope or not is totally local: a test run requires around 340 scope management messages regardless of the scope definition. In turn, GBR pays the price of the increased reliability of the acknowledged reverse paths. This overhead, in contrast to flooding, does depend on the number of member nodes. Therefore, the GBR traffic for $S_1$ exhibits a constant factor of times 2, while for $S_2$ the factor is times 1.5, compared to the FLD traffic.

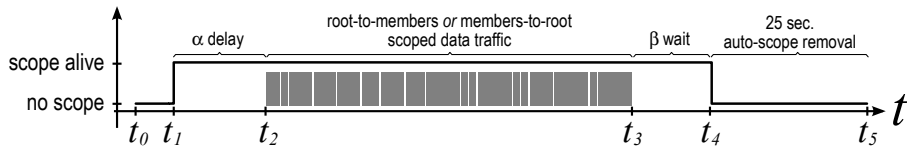



**Figure 4.6:** Test sequence for scope data traffic

### Scope Traffic

We proceed by evaluating the bidirectional data channel described in section 4.2.1.1. As important metrics, we consider *goodput* and *link level message exchange*. While the former refers to the end-to-end application layer traffic, the latter indicates the in-network traffic needed to obtain that goodput. We have designed our traffic tests as illustrated in Figure 4.6. During the first $[t_0, t_1)$ and last $(t_4, t_5]$ stages, there is no activity. A scope is created at $t_1$, and remains alive until it is removed at $t_4$. In order to allow the scope to stabilize after its creation and removal, we introduce an initial delay of $\alpha$ and a wait of $\beta$ seconds (respectively). Traffic itself occurs during $(t_2, t_3)$.

The two traffic directions, root-to-members (RtM) and members-to-root (MtR), were tested separately. For the RtM tests we chose $\alpha$=15s and $\beta$=3s. This large $\alpha$ value allows the scope to stabilize through two refreshes before data transmission starts. For the MtR tests we chose $\alpha$=3s and $\beta$=20s. Here, a short $\alpha$ was sufficient, but a larger $\beta$ was needed to account for removed (i.e., unstable) nodes. More importantly, since sensor network applications are diverse, we concentrate on two communication patterns: periodic and bursty traffic.

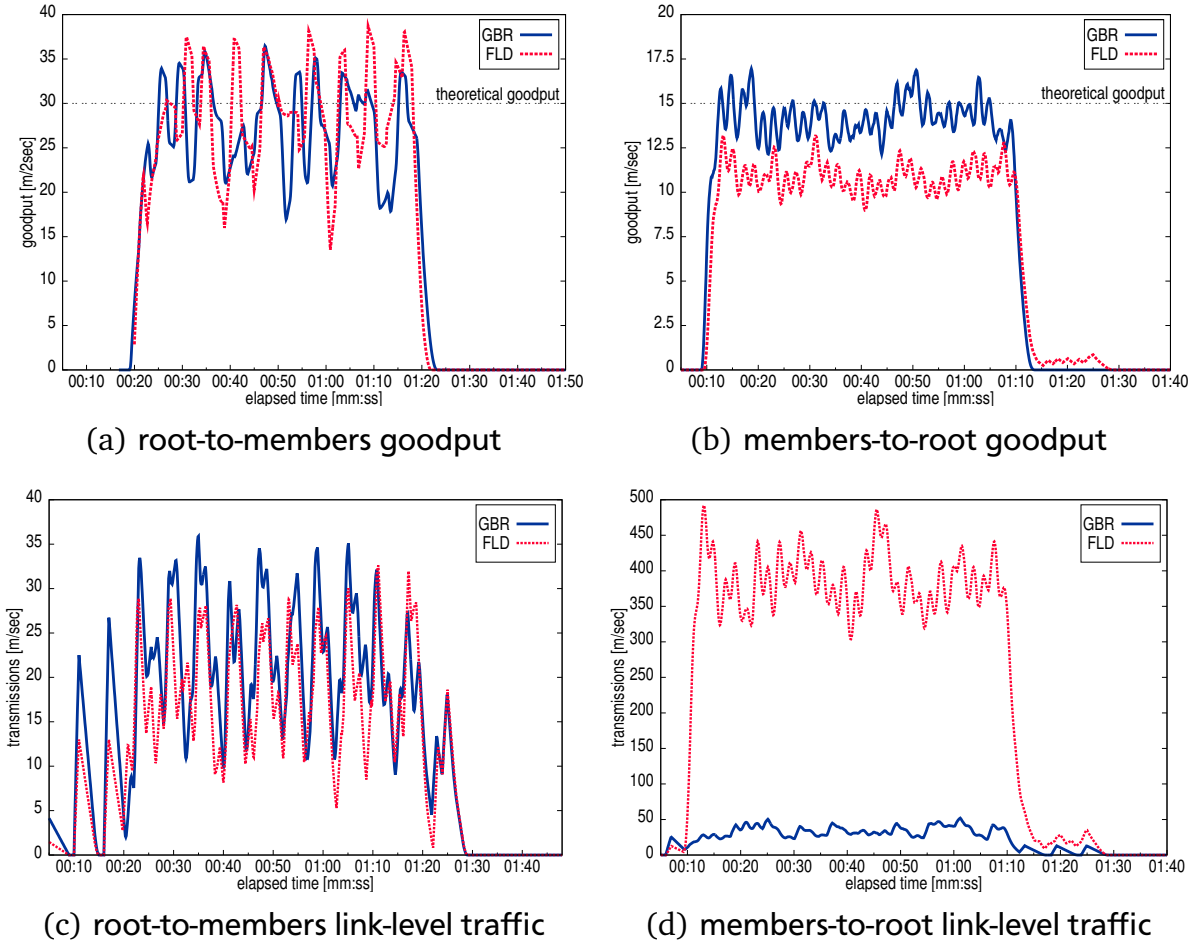
### Periodic Traffic.

These tests encompass, e.g., monitoring applications. Here, in RtM tests, the scope root sends $M = 30$ messages to the members, whereas in MtR tests each scope member sends $M$ messages to the root. Each sending node posts a timer that is triggered with a frequency $\lambda = \frac{t_3 - t_2}{M} = 2$ seconds. To reduce network collisions, message transmission is delayed randomly by $\varepsilon$ (with $\frac{\lambda}{8} \leq \varepsilon \leq \frac{7\lambda}{8}$) by the sending application. To further stress the routing protocols, we used scope $S_1$, which includes all 30 network nodes.

Especially the RtM plots have a high fluctuation in the measurements. This is the result of the predefined intervals in the test execution as explained above and the different clock drifts in the sensor nodes that lead to differently shifted intervals throughout the network. The result is a high peak in one interval and a deep valley in the following interval. These artifacts can be observed in all the experiments, more or less distinct.

The top two plots of Figure 4.7 present goodput results both for RtM and MtR. (Note that RtM was sampled every 2 seconds, thus the theoretical goodput is 30 [messages/2 seconds], while MtR was sampled every second, thus the theoretical goodput is 15 [m/s].) The RtM goodput plot, 4.7(a), shows that both protocols have similar goodput. FLD showed an average goodput of 96.44%, slightly above GBR's 91%. The MtR

goodput plot, 4.7(b), does however show a clear advantage for GBR (95.44%) over FLD (74.44%). This was to be expected since flooding causes too many collisions, while GBR restricts the data flow to those links whose gradient values are high. The tail exhibited by FLD was due to a node that got automatically removed from the scope at the middle of the test and then heard a refresh, hence re-started sending messages.




(a) root-to-members goodput

(b) members-to-root goodput

(c) root-to-members link-level traffic

(d) members-to-root link-level traffic

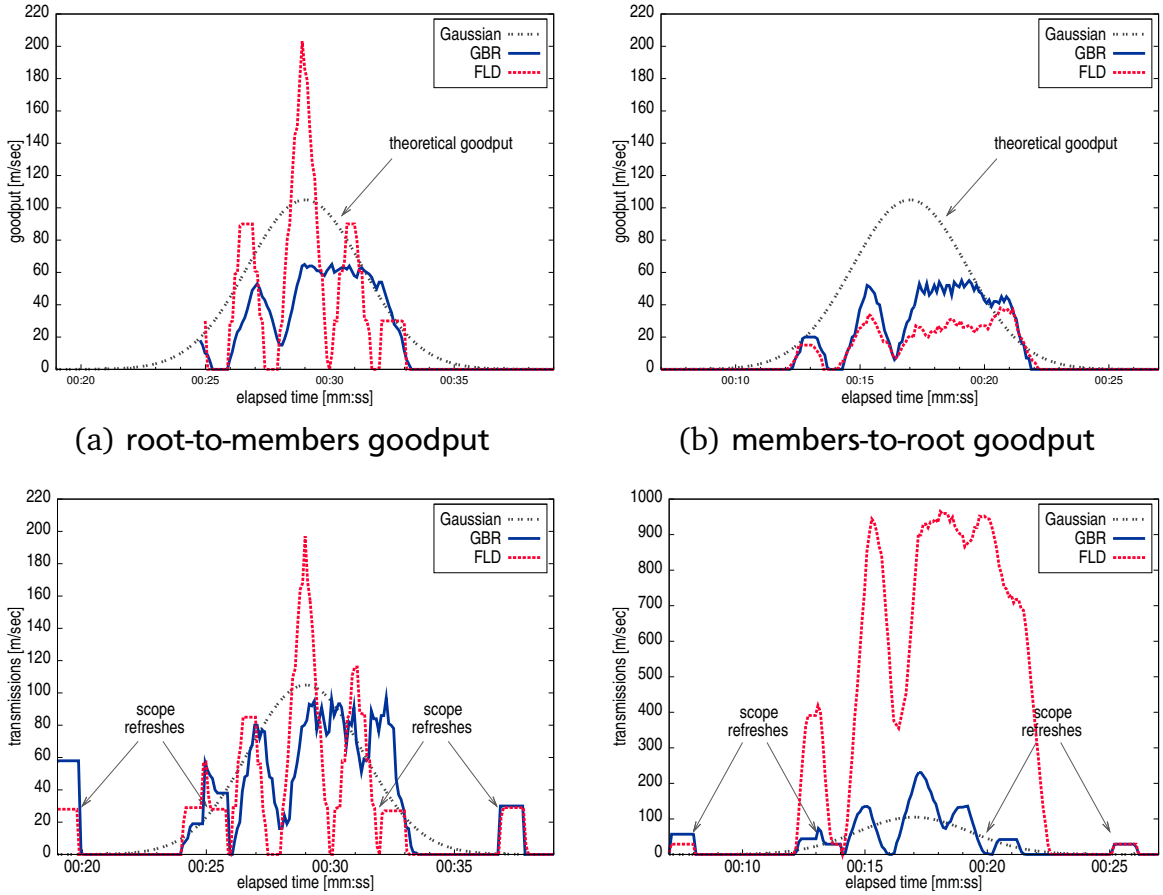
**Figure 4.7:** Goodput and link-level periodic traffic

In the bottom plots of figure 4.7, the link-level traffic is shown. These curves include both data and scope management traffic. The MtR plot, figure 4.7(c), exemplifies that the typical usage yields marginal management overhead. GBR transmits 15% more messages than FLD due to the acknowledgment and reject messages. This increase in link-level traffic constitutes an upper bound for GBR's overhead compared to FLD. The biggest difference between the routing protocols becomes evident in figure 4.7(d). GBR is one order of magnitude better than FLD: it induces less than 9% the messages required by FLD. This constitutes GBR's main strength and outbalances any scope maintenance overhead, even under the chosen adverse test conditions.

**Bursty Traffic.**

The following tests encompass event-based applications such as object-tracking, where bursty traffic is the predominant pattern. A message burst is produced in two cases: when an event-subscription is to be disseminated rapidly from the subscribers to the publishers (RtM traffic), and whenever a set of nodes detect an event and need to notify the subscriber (MtR traffic). To support bursts, however, several mechanisms such as route lookups and concurrent medium access via the MAC protocol must tolerate stress.

For the bursty tests, sending nodes produced a message burst which can be approximated by a gaussian function with center=5, standard deviation=2.3 and a peak value of 7. This amounted to a total of $M=15$ messages. As with the periodic traffic, the burst is ideally sent in intervals of $\lambda=2$ seconds, and scope $S_1$ was used for the experiments. In the RtM tests, it is only the scope root node who sends the burst, while in MtR tests, all 30 (member) nodes sent the burst back to the root.




(a) root-to-members goodput

(b) members-to-root goodput

(c) root-to-members link-level traffic

(d) members-to-root link-level traffic

**Figure 4.8:** Goodput and link-level bursty traffic

The results presented in figure 4.8 include both data and scope management traffic. The theoretical values are represented by the Gaussian curve. Figures4.8(a) and 4.8(b) display the goodput for RtM and MtR traffic, respectively. In the RtM tests, GBR produced a goodput of 78.62%, while with FLD 99.77% of expected messages were re-

ceived. In the MtR tests, FLD got 44.54% and GBR got 69.19%. Further, the graphs show that in terms of burstiness, FLD was superior to GBR. While FLD exhibited the expected behavior, GBR showed a high latency. This was due to GBR reaching a maximum bandwidth. While FLD uses the capacity of all communication links to send messages, GBR restricts the nodes' out-degree. This sets a capacity along the established routes. Hence, under the same load, GBR needs more time to transport the message burst.

Figures 4.8(c) and 4.8(d) show the total RtM and MtR traffic, respectively. All of the qualitative statements regarding the burstiness and latency made above hold here as well. As for throughput, we can claim that both FLD and GBR reach the maximum. While FLD utilizes the complete network capacity, GBR's traffic is lower. The significantly better GBR efficiency over FLD for MtR – an order of magnitude – is clearly visible in 4.8(d).

### 4.2.2 Contiki

With the previous implementation we reached the limits of SOS and had to cope with several stability issues. Therefore, we had to look for a new system to build further enhanced versions of the Scope Framework on.

The second part of our Scopes Framework evaluation will show the second, extended implementation of scopes built on top of the well known Contiki operating system. We evaluate the Scopes Framework on Contiki and relate the results to the former SOS version. We use the SOS version as a baseline for the performance evaluation of our new implementation.

#### 4.2.2.1 Implementation details

The extended reimplementation of the Scopes Framework had the goal to be based on a stable and widespread operating system and had to clean up problems encountered in the previous version. It provides the same functionality as its predecessor and got extended with additional features, where the security extensions are the most important improvements.

Our choice of the Contiki operating system is based on its broad and active community. It has support from industry and academia, even though it started as an academic project. From the implementation point of view, Contiki is implemented using C, as our previous version of Scopes was, and it supports concurrent tasks and over-the-air (OTA) task updates. With this background Contiki proved to be much more stable than SOS and supported all the features we need for the Framework.

The design for the new Scopes Framework was kept closely to the original, as the layered and modular architecture has proven to be a good choice for extendability and adaptability to different scenarios in our tests. The basic functionality of scope creation, maintenance and deletion was also transferred and basic membership functions were added. Developing on Contiki also introduced a different style of coding. The SOS

operating system wrapped everything in an event. This means a module had, beside some management methods a central event handler that managed the behavior of the module. In Contiki so called protothreads [29] and callback functions are heavily used to structure tasks. Also the separation of different tasks is not as sharp as the modules on SOS, so deploying a task OTA is more difficult on Contiki as references to external functions have to be managed.

One major issue to be solved in the new version was to reduce the complexity of code. Especially the GBR routing is very complex and memory intensive, as two routing trees have to be managed. Our solution is SelfUR (selective flooding, unicast return; we use SEL in the following). It is specified at length in [64], we will show here a short summary.

We identified two major issues with the former GBR algorithm. It is complex in managing the routing trees and it uses a large amount of memory for their management, where most of the memory is dynamically allocated. So, first, all the memory used for SEL is now allocated statically, this prevents resource problems at runtime. The basic idea of GBR is kept for the implementation of SEL, but we incorporated some observations: One of our observations was, that when creating a top-level scope the whole network has to get the SCRs and this is the same for all scopes of this kind. Second, creating multiple scopes resulted in multiple routing trees, two per scope. We changed this behavior to a single routing tree created for a specific root node. This ensures a low memory footprint and much less tree management messages in the sensor network. The routing tree is used for the member nodes to send data back to the root node. Sending data to the member nodes is done trough selective flooding, as in this direction all nodes are addressed. For the selective flooding not to send all data to the whole network each time, the nodes store their roles for the different scope routings. This means besides the root and member nodes there are so called forwarder nodes. These are in charge to connect the different member nodes and the root node to be able to communicate with each other. But the forwarder nodes do not participate in the scope itself. This means that they do not send messages to or receive messages from participants of the scope directly, they just forward. This way only member, root and forwarder nodes broadcast messages to the member nodes and it is ensured that these messages are kept in the vicinity of the scopes spacial distribution.

The APIs of the new Scopes Framework have been changed slightly from the previous version. The task API, listing 4.5, has been adjusted to the new task registration mechanism and some functionality was moved into the interface.

To receive data from a scope in the SOS version it was possible to address a task by its module id. Contiki does not know this kind of id. So, we introduced a registration mechanism where a task registers itself with the Scope Framework and a self-assigned id. This way the id is known and a task can be addressed. Additionally the Scopes Framework now has a `scopes_init(...)` method where the used routing and membership functions can be configured. This way it is possible to configure different routing algorithms for different scenarios. Nested scopes inherit the routing of their parent scope.

```
/* callbacks */
struct scopes_application {
  void (* add)(scope_id);
  void (* remove)(scope_id);
  void (* join)(scope_id);
  void (* leave)(scope_id);
  void (* recv)(scope_id, void *data, data_len);
};

/* interface function declarations */
void scopes_init(struct scopes_routing *routing,
                 struct scopes_membership *membership);
int scopes_register(subscriber_id, struct scopes_application *task);
void scopes_unregister(subscriber_id);
bool scopes_open(subscriber_id, superScope_id, scope_id,
                 void *specs, spec_len, flags, ttl);
void scopes_close(subscriber_id, scope_id);
void scopes_send(subscriber_id, scope_id, direction, void *data, data_len);
```

**Listing 4.5:** Scopes Framework task API (Contiki version).

With `scopes_open(...)`, `scopes_close(...)` and `scopes_send(...)` there are the methods as in the previous versions API. What was added are the scope status callbacks `add(...)`, `remove(...)`, `join(...)` and `leave(...)` that are called if a scope is added or removed from the ScopeManagers internal scope table, or if the node joins or leaves a scope. This way a task knows what scopes are available and in which state they are. This is important if the task is also executed when the scope is not alive and it has to wait until the node joins the scope again to exchange its data.

```
/* callbacks */
struct scopes_membership {
  int  (* check)(void *specs, spec_len);
};
```

**Listing 4.6:** Scopes Framework membership API (Contiki version).

The APIs for the membership function and routing (listings 4.6 and 4.7) are not changed. The notification interface of the routing API got enhanced to match the one from the task interface. This also allows a more fine grained resource management in the routing algorithms.

The property repository API, listing 4.8, allows access to the repository that is used for membership evaluation. It provides the ability to read and write values to the repository.

```
1  /* callbacks */
2  struct scopes_routing {
3    void (* send)(scope_id, direction);
4    void (* add)(scope_id, direction);
5    void (* remove)(scope_id);
6    void (* join)(scope_id);
7    void (* leave)(scope_id);
8  };
9
10 /* interface function declarations */
11 void scopes_receive(void* data);
```

**Listing 4.7:** Scopes Framework routing API (Contiki version).

```
1  /* interface function declarations */
2  int scopes_repository_value(index);
3  int scopes_repository_value_set(index, value);
```

**Listing 4.8:** Scopes Framework property repository API (Contiki version).

#### 4.2.2.2 Methodology and Setup

The evaluation of the Contiki version of the Scopes Framework was also conducted in our live testbed at the Piloty building. But it was enhanced with an additional room, compared to the previous SOS evaluation, as explained in section 4.1. Which leaves it with 32 TelosB sensor nodes in 10 rooms and a coverage of a bit less than 600 $m^2$. The testbed was also extended by a router per room that is able to deploy new node images in the network and provides a convenient logging mechanism. In the extended sensor network the node ids were reassigned, but this has no influence on our experiments, besides the renaming.

**Scopes.**

In our evaluation we have used the scopes depicted in figure 4.9. These represent the same scope definitions as in the SOS version for comparison reasons, listed in table 4.3. The scope $S_1$ covers all nodes. Scope $S_2$ focuses on the distributed properties of a scope. Finally, $S_3$ and $S_{31}$ illustrate nested scopes. The last column of table 4.3 presents the size of scope definitions in Contiki and the entire creation message. These show the same compactness of the representation as the SOS version of the framework. All scopes were created at the network boundary, the top left node 1, this forces a hop count between 2 and 3.

**Data logging.**

The distributed data logging is in the enhanced testbed far more advanced than our old approach. With the new Motelab backbone described in section 4.1 we are able

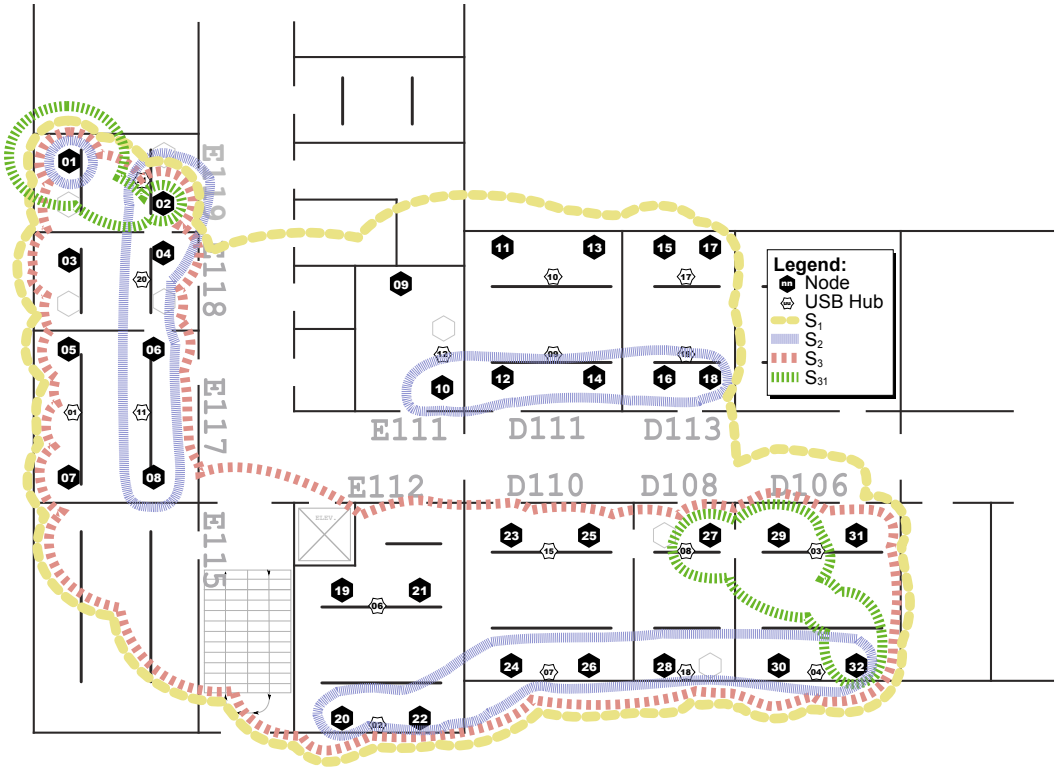



**Figure 4.9:** Enhanced *Piloty* building testbed and scopes used for evaluation on Contiki

| Scope | Description | Parent Scope | Size | |
|---|---|---|---|---|
| | | | Scope Def. | OS Message |
| $S_1$ | All nodes | *world* | 5 bytes | 19 bytes |
| $S_2$ | Nodes with even id | *world* | 5 bytes | 19 bytes |
| $S_3$ | Nodes whose $id \leq 8 \mid\mid id \geq 19$ | *world* | 11 bytes | 25 bytes |
| $S_{31}$ | Node id $\in \{1, 4, 27, 29, 32\}$ | $S_3$ | 29 bytes | 43 bytes |

**Table 4.3:** Summary of employed scopes

to log all serial output of all nodes in the network. Therefore, the sensor nodes are connected to the Motelab routers. These connect to the serial console of each node and send every output line to a database server, where the entry is stored with a timestamp in a database table. This introduces a small delay between issuing of a log entry on a sensor node and the storage and time-stamping at the database, but as all routers are connected to the same network and the data is transmitted via Ethernet the delay can be assumed constant over all entries.

With this approach our logging is independent from node failures or network issues, which we encountered with the wireless approach we used with SOS. Additionally, the complicated and error-prone computation to synchronize timestamps of log entries over all nodes in the network is obsolete, as we now have a central instance that assigns timestamps. With the logging function on the node we also were dependent on the

resources of the sensor node, not just memory space, but also computation time. With the logging on the sensor nodes influences on the system behavior cannot be ruled out. The external logging also removes these influences.

#### 4.2.2.3 Performance Evaluation

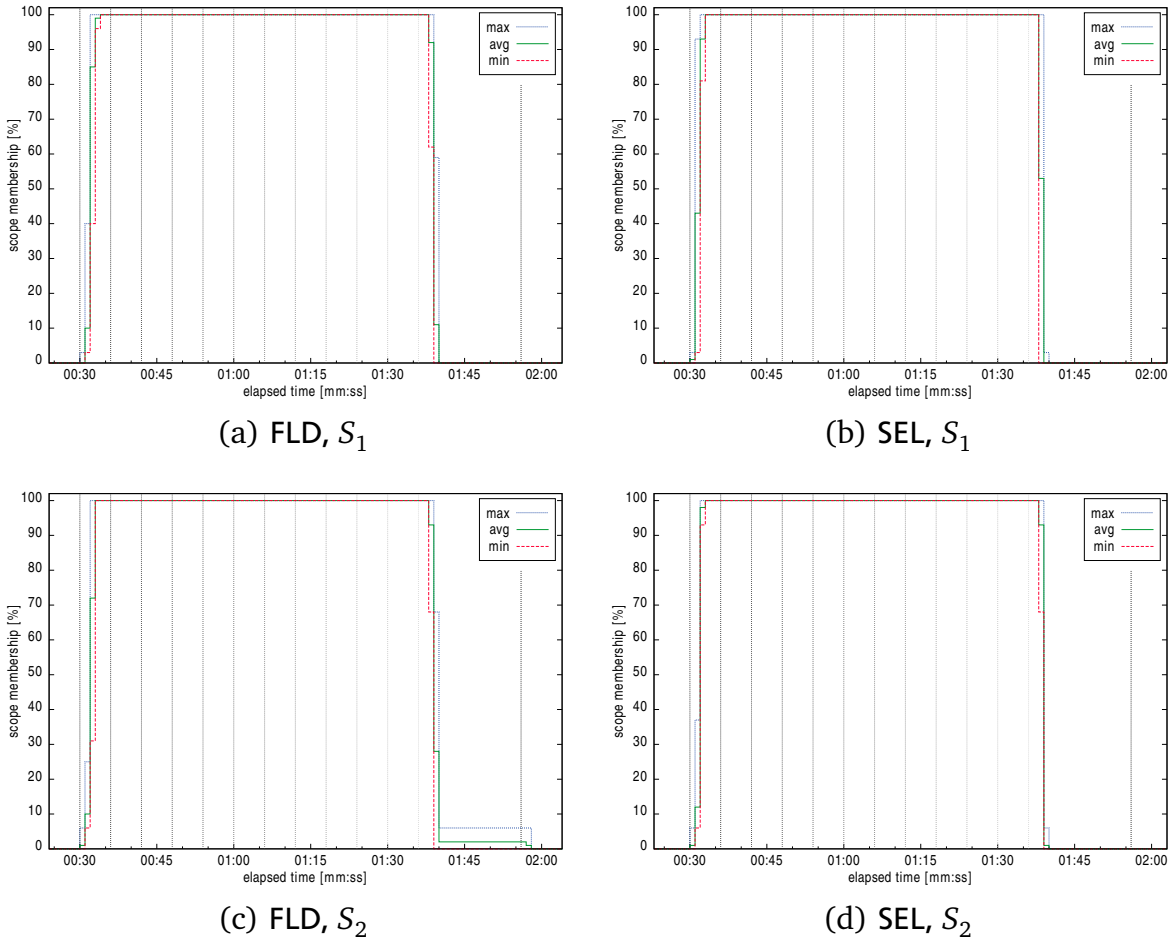
In conformance with the evaluation of the SOS version of the Scopes Framework we will look at the framework's *reliability* and *efficiency*, see section 4.2.1.3 for the definitions.

**Scope Creation, Removal and Maintenance.**

First, the reliability of managing a scope is investigated. We look at the operations of scope *creation*, *maintenance* and *deletion*. To evaluate the reliability we employ the same measurements as with the SOS results. We assume an ideal curve $\phi_I(t)$, where the percentage of member nodes is 0% if the scope is not created or is deleted and from time $t$ on, where the scope is created and managed, the number of member nodes is 100%. This leads to the function $\phi\colon \mathbb{Z} \to [0,1]$, that we have seen before in section 4.2.1.3. It describes the sequence of measured membership events and shows how many percent of the nodes in the network are members of a scope at a given time $t$ of the experiment. To gain a better understanding of our results we show besides the average of 5 test runs $\phi_{avg}$ also the minimum $\phi_{min}$ and maximum $\phi_{max}$ percentage of memberships achieved over all test runs. Of course this does not reflect the single experiment run, but gives an impression of worst and best case. The average can give additional guidance on how to interpret these two curves, as the distance of the average curve from the min and max curves relates to the probability of these two extremes. The shorter the distance, the higher the probability. In the optimal case all three curves match the ideal $\phi_I$.

In our experiments with the Contiki based framework we send the scope creation at $t = 30$ seconds. The scope is then kept alive for 68 seconds and is then deleted by a deletion message at $t = 98$ seconds. The scope refresh interval is set to 6 seconds like in the SOS experiments and reflected in the graphs by light grey vertical lines. The time span the scope is alive allows for 11 scope refresh cycles. If a node misses the scope deletion message the scope times out 20 seconds after the last refresh message and is then deleted at $t = 116$ seconds. This is also shown in the graph by the last vertical line on the right.

In figure 4.10 we show the results for the reliability tests for the scopes $S_1$ and $S_2$. First thing to mention here is that we achieve almost perfect results in these tests with our new SEL routing algorithm. For scope creation we measured an average of 3 seconds from sending the first SCR to achieving 100% membership. Deleting a scope was even faster with an average of 2-3 seconds form sending the removal message to all members leaving the scope. As can be seen in 4.10(c) sometimes nodes miss the deletion event, but then get cleaned by the scope time-out.




(a) FLD, $S_1$ (b) SEL, $S_1$

(c) FLD, $S_2$ (d) SEL, $S_2$

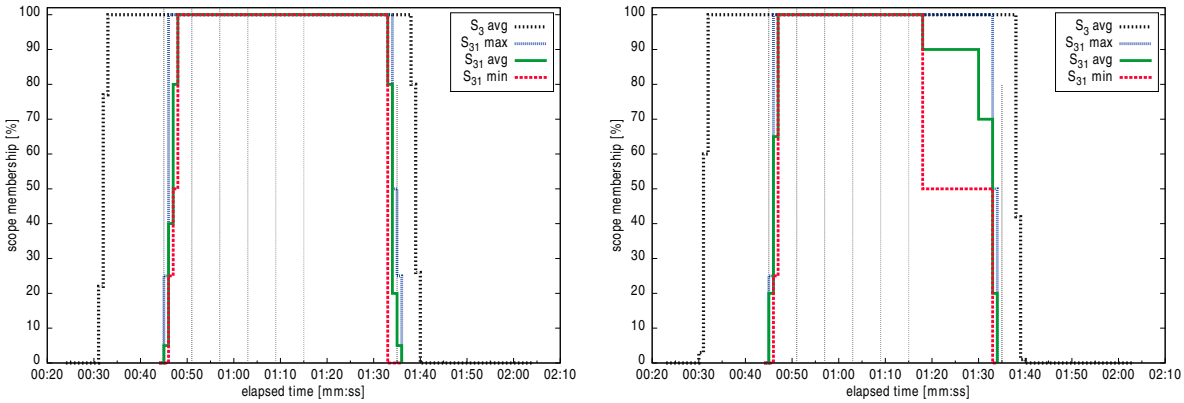
**Figure 4.10:** Scope creation, removal and maintenance on Contiki

Compared to the SOS version Contiki enables a much better stability, as for the flood routing no random scope leaves and re-joins of nodes can be seen in figure 4.10(a). The use of selective flooding in SEL for the root node to member node communication drastically improves the reliability of the scopes' memberships. For the scope $S_1$ selective flooding (fig. 4.10(b)) and flooding (fig. 4.10(a)) are the same, as here the whole network is included in the scope. But already scope $S_2$ profits from selective flooding's reduction of SCR resending and achieves the same reliability of the flooding algorithm with fewer messages sent. This can be seen by figures 4.10(c) and 4.10(d).

**Nested scopes.**

Now we show the evaluation of the reliability of FLD and SEL regarding nesting a scope in figure 4.11. Therefore, we create $S_3$ in the same way as for the experiments above, but we additionally create scope $S_{31}$ 15 seconds after $S_3$. It remains for 48 seconds and is then deleted, 5 seconds before $S_3$ is also deleted.

In figure 4.11(a) we can confirm the observation from above, that our Contiki implementation of the Scopes Framework enables us to get results matching the ideal case. Both scopes, $S_3$ and $S_{31}$, are alive with all member nodes from creation to deletion of




(a) FLD, $S_3$ (b) SEL, $S_3$

**Figure 4.11:** Scope creation, removal and maintenance for nested scopes on Contiki

the scopes. We also observe a slight increase in time to create the subscope, here we get an average of 3.6 seconds. Still this shows that the first SCR is received and processed by all nodes. With the SEL routing, figure 4.11(b), the connection to the two nodes far from the root node got lost in some experiments and is reflected in the lesser reliability before deleting the scope. The average value imposes that this problem only occurred in a small number of runs.

**Network density.**

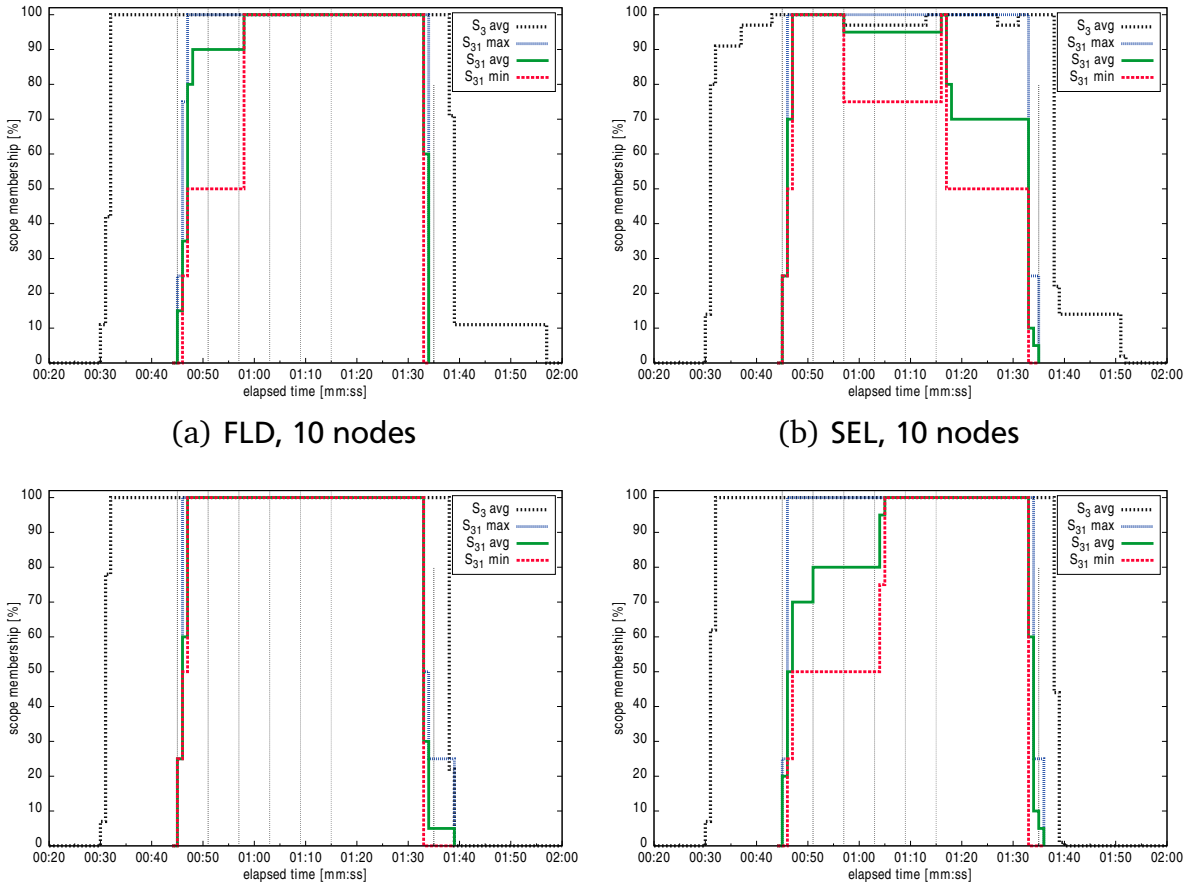
The areal densities $\rho_A$ changed slightly compared to the SOS evaluation. In table 4.4 the densities for our experiments are shown. $\rho_A = 4$ and $\rho_A = 2$ increased by two nodes and the area covered also changed slightly.

| $\rho_A$ | #Nodes | Nodes per area |
|---|---|---|
| 4 | 32 | $1/17.3m^2$ |
| 2 | 20 | $1/27.5m^2$ |
| 1 | 10 | $1/54.9m^2$ |

**Table 4.4:** Node densities for Contiki

The stability for density $\rho_A = 2$ for the scopes $S_1$ and $S_2$ show the same almost ideal behavior as we have seen for the $\rho_A = 4$ density seen in figure 4.10. Therefore, we omit these figures. Only for the $\rho_A = 1$ we recognize a slightly lower stability at the beginning of the scope creation. The fewer messages exchanged by SEL and the relatively sparse network resulted in some nodes joining with a later SCR then the first one. This implies that for the optimal performance of the Scopes Framework a higher density has to be chosen.

The figure 4.12 supports the conclusion that density $\rho_A = 1$ is not favorable for the Scopes Framework. In 4.12(a) and especially 4.12(b) it can be seen that this density




(a) FLD, 10 nodes

(b) SEL, 10 nodes

(c) FLD, 20 nodes

(d) SEL, 20 nodes

**Figure 4.12:** Scope density for $S_3$ and $S_{31}$ on Contiki

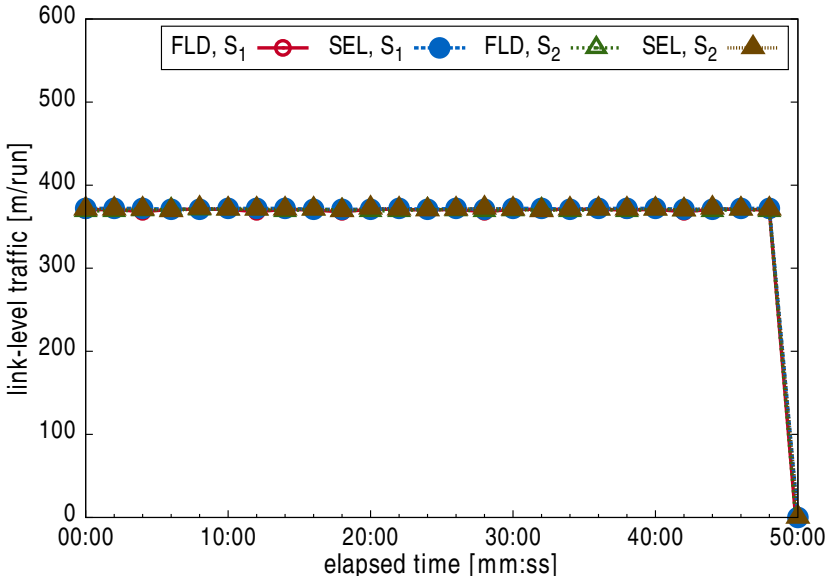
results in more unstable scope memberships, but still an average membership rate of nearly 80% is achieved. From the definition of scope $S_{31}$ can be seen, that these 20% missing relate to one node missing in the scope. Besides that the two other figures show the same results for the base scope as we have seen in the single scope experiments. Just the scope creation in figure 4.12(d) of the sub scope took some additional refresh cycles until the scope reached a stable state. From there on it is stable and the scope deletion is handled as expected and the scope is deleted after receiving the deletion message.

**Resource Costs.**

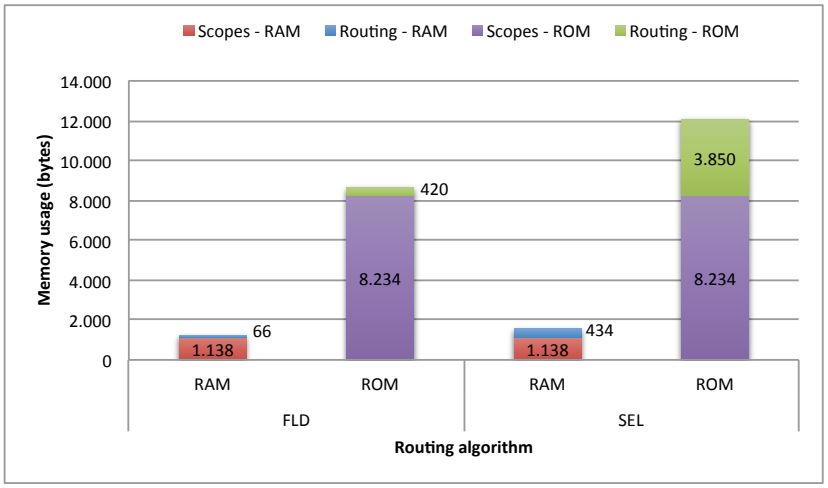
The efficiency metric for the scopes operation costs also used in the SOS evaluation is shown in figure 4.13(a) for the Contiki version of the Scopes Framework. The figure shows the link-level traffic for $S_1$ and $S_2$ over the scopes life-cycle. Again, each measurement took 2 minutes and represents all the messages exchanged in the tests run, there were 25 test runs. As the network is 2 nodes larger FLD uses around 370 scopes management messages. SEL used the same number of messages. This is to be expected, as SEL uses a flooding algorithm for the RtM communication direction and the scope management messages are only issued from the root node. Compared to GBR the man-

agement costs are reduced and are deterministic as they only depend on the number of nodes and not additionally if a node joins the scope, as this was the case for GBR.

To estimate the resource costs we sum up the ROM and static RAM usage of all modules implemented in this work. Dynamic RAM allocation is not used. Other modules provided by the Contiki OS are not listed here as these are resources that Contiki uses independently of the Scopes Framework. Especially the routing algorithms reuse modules provided by Contiki for basic communication patterns. This implies that the overall firmware size is the size of the Scopes Framework RAM and ROM, listed in figure 4.13(b), plus the size of the Contiki OS RAM and ROM usage.




(a) Scopes operation cost




(b) Scopes memory costs (RAM / ROM)

**Figure 4.13:** Resource costs

Figure 4.13(b) shows the amount of RAM and ROM used by the Scopes Framework. The size is independent of the routing algorithm being used and uses roughly 8kB ROM and 1kB RAM. It includes all the Scopes functionality, like handling of scopes, member-

ship checking, data store for membership properties and logging. The scopes modules plus the routing algorithms have a total size of 8.5kB ROM for FLD and 12kB for SEL. The RAM usage is roughly 1kB for FLD and 1.5kB when using SEL. These numbers leave enough room for additional code for Scope tasks.

**Scope traffic.**

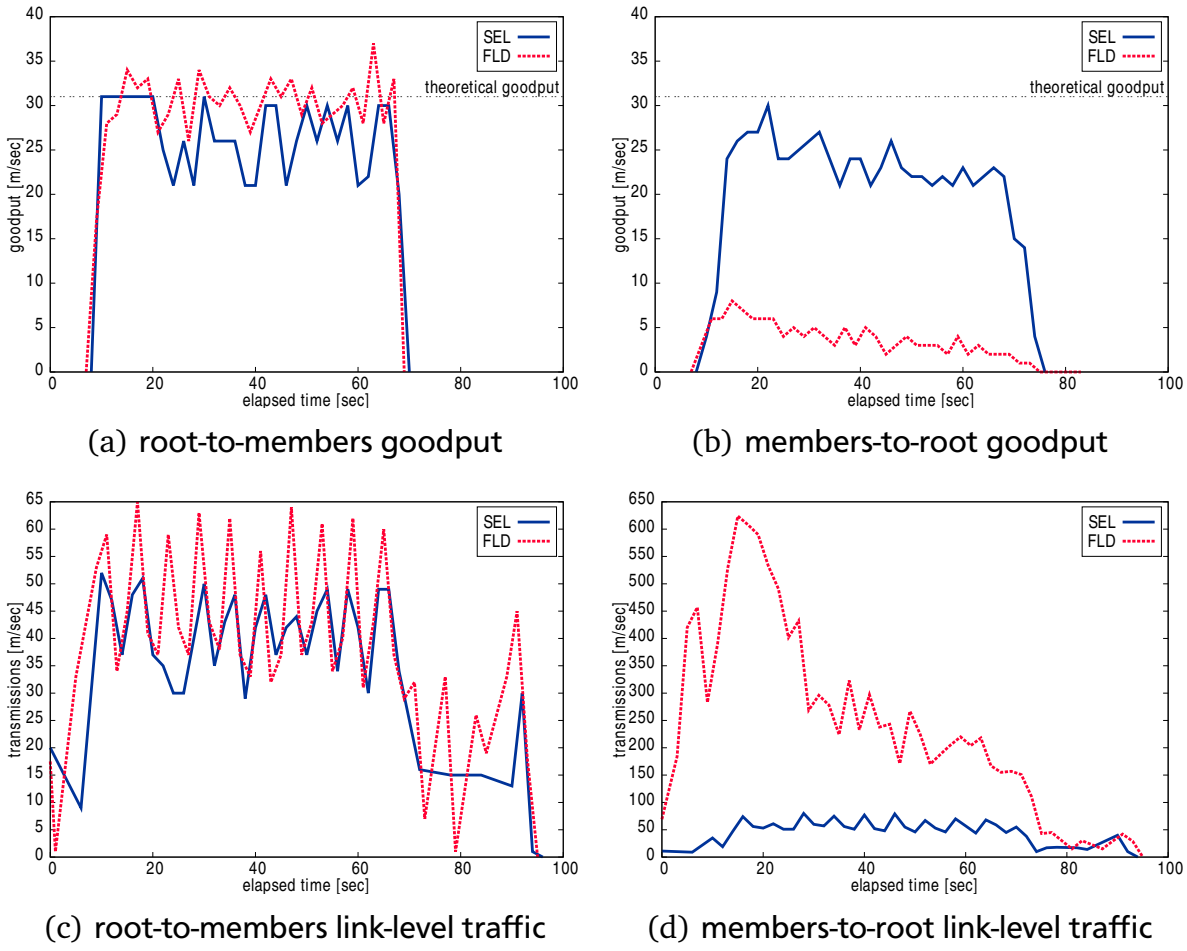
For the Scope traffic tests on Contiki we employed the same mechanisms that were used for the SOS evaluation. We have a setup and tear down phase at the beginning and end of every run to let the scope stabilize or be removed. In between our traffic tests take place. The detailed description given in the corresponding section of chapter 4.2.1.3 remains valid for the current evaluation.

We also evaluated the two traffic directions root-to-member (RtM) and member-to-root (MtR) separately with two communication patterns: burst and periodic, as these are very commonly used patterns in sensor networks. For all these tests the same delays are used for stabilization and teardown phase. These are $\alpha$=5s after creation of the scope to let it stabilize and $\beta$=23s for the tear down phase to let all the remaining messages be delivered before the scope is closed.

**Periodic traffic.**

In the RtM tests, the scope root sends $M = 31$ messages to the members, whereas in MtR tests each scope member sends $M = 31$ messages to the root. Each sending node posts a timer that is triggered with a frequency $\lambda = \frac{t_3 - t_2}{M} = 2$ seconds. To reduce network collisions, message transmission is delayed randomly by $\varepsilon$ (with $\frac{\lambda}{8} \leq \varepsilon \leq \frac{7\lambda}{8}$) by the sending application if sending direction is MtR. To further stress the routing protocols, we used scope $S_1$, which includes all 31 network nodes.

The first two figures in 4.14 show the results of the goodput experiments. The data is shown in messages per two seconds, as the send interval is two seconds. Both figures show a similar behavior we observed in the previous experiments with the SOS version. Figure 4.14(a) shows the RtM direction and a slightly better performance of the FLD routing (98.70%) compared to the SEL routing (87.50%). These results are better than the results under SOS: +3.5% for the FLD routing and +13% for the new SEL routing. Figure 4.14(b) shows the MtR results and also shows results comparable to the SOS experiments. It shows for FLD routing 14.98% and for SEL routing 78.07% of the theoretical goodput. This result reveals a higher sensitivity of the system to high network traffic. Compared to the former SOS results the relations between the routing algorithms are still valid, but the absolute numbers are worse: for FLD a loss of 59.46% and for SEL routing a loss of 17.37%. The next two figures 4.14(c) and 4.14(d) depict the link-level traffic and reflect the results of the previous two figures and their results. The SOS results show the same behavior, but with an even worse result for FLD performance. The figures show all link-level traffic, this means data and management traffic of the Scopes Framework. In figure 4.14(c) we can see quite similar traffic for both routing algorithms. This is as expected, as in the RtM direction SEL also uses a flooding




(a) root-to-members goodput

(b) members-to-root goodput

(c) root-to-members link-level traffic

(d) members-to-root link-level traffic

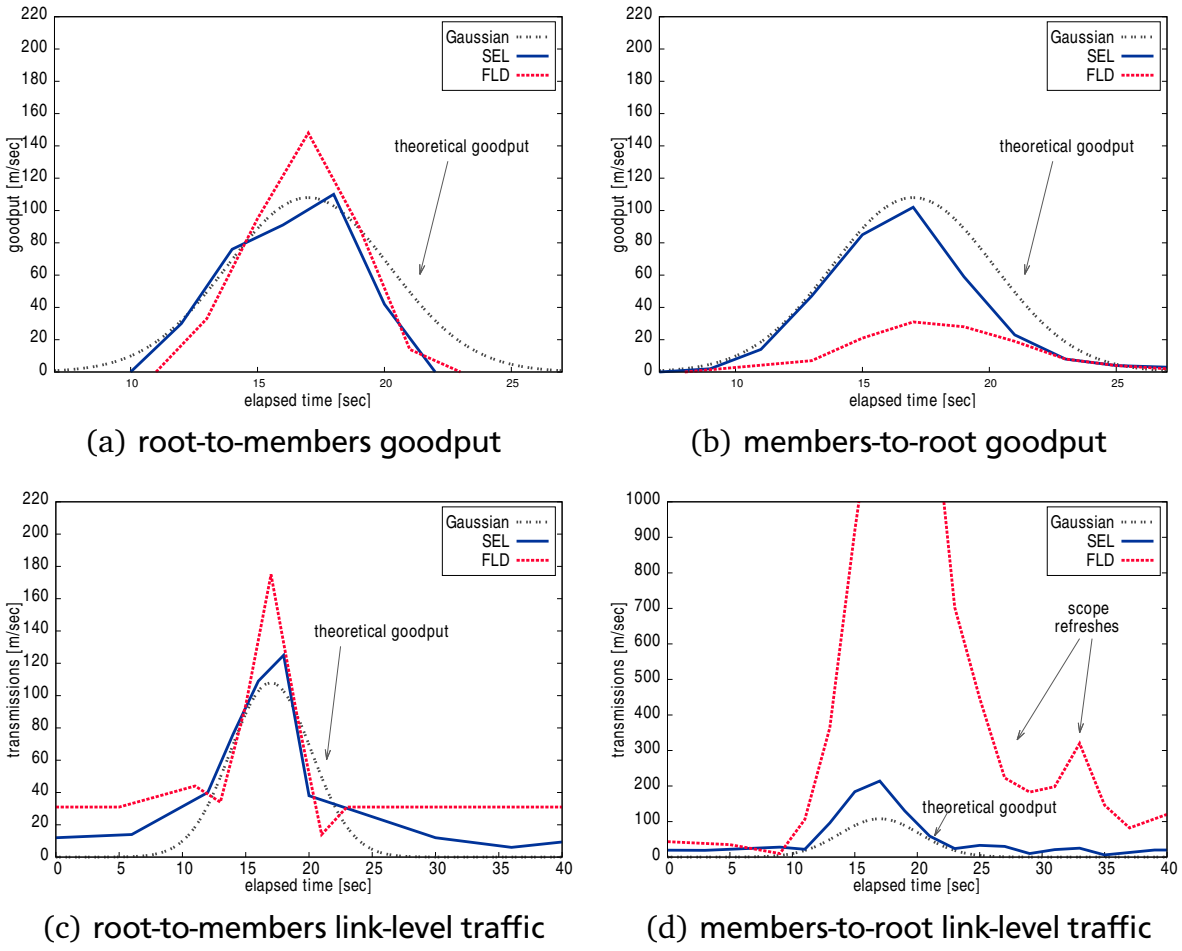
**Figure 4.14:** Goodput and link-level periodic traffic on Contiki

technique for routing. Still SEL uses 20.86% less messages than FLD. In figure 4.14(d) The traffic of FLD is up to a magnitude larger than the SEL traffic. Over the total time of the experiment FLD uses 562,68% of messages compared to the total SEL traffic.

**Bursty traffic.**

For these bursty tests, sending nodes produce also a message burst which can be approximated by a gaussian function, as with the SOS tests before. A gaussian function with center=5, standard deviation=3.2 and a peak value of 7 is used. This amounted to a total of $M$=15 messages. As with the periodic traffic, the burst is ideally sent in intervals of $\lambda$=2 seconds, and scope $S_1$ was used for the experiments. In the RtM tests, it is only the scope root node who sends the burst, while in MtR tests, all 31 (member) nodes sent the burst back to the root. For the figures the number of messages in each 2 second slot is evaluated. This reduces the resolution of the results, but better reflects the message count per slot. This is also the cause that the scope refreshes are only reflected when they generate a high load, which is only significant in figure 4.15(d).

The results presented in figure 4.15 include both data and scope management traffic for the link-layer. The theoretical values are represented by the gaussian curve and are




(a) root-to-members goodput

(b) members-to-root goodput

(c) root-to-members link-level traffic

(d) members-to-root link-level traffic

**Figure 4.15:** Goodput and link-level bursty traffic on Contiki

data traffic only. Figures 4.15(a) and 4.15(b) display the goodput for RtM and MtR traffic, respectively. In the RtM tests, FLD is slightly superior to SEL with 82.23% compared to 75.44% goodput. This is the result of redundancy of flooding combined with a small amount of traffic, which can clearly be seen in the link-level traffic figure 4.15(c). Here, in the RtM direction only the root node sends and the number of messages in the network is fair and introduces only some more management and routing overhead than SEL, 119.35% against 63.77% for SEL, compared to the MtR scenario. In the MtR tests, figure 4.15(b), FLD had a goodput of 27.52% and SEL reached 76.43%. Further, the graphs show that FLD had a similar behavior as the tests executed on the SOS platform, the SEL routing performs better than the old GBR. In total FLD lost some goodput compared to the previous tests, due to the same causes as before. The link-level traffic for FLD leads to congestion and message loss due to collisions or dropping of messages, compare figure 4.15(d). The significantly better SEL efficiency over FLD for MtR – an order of magnitude – is clearly visible in 4.15(d) and could be preserved from the former SOS implementation.

## 4.3 SecScopes Framework

The SecScopes Framework is the combination of the Scopes Framework for Contiki with the security measures proposed in chapter 3. First, we will evaluate the performance of the chosen security algorithms on common sensor network platforms, used by the newly introduced security measures in the Scopes Framework. Then we will have a look at the combined Framework with the security measures.

### 4.3.1 Security Algorithms

We have three sets of security algorithms available to our implementation. First, the AES Algorithm implemented in the hardware of the IEEE 802.15.4 radio chip. Second the TinyECC library that supports elliptic curve cryptography and provides algorithms, e.g. for encryption/decryption or sign/verify data. TinyECC includes a math library for calculations on elliptic curves. These math functions were enhanced by us to support our implementation of the CP-ABE algorithm that was implemented from scratch for this work. CP-ABE is the third.

In figure 4.16 the dependencies between the Scopes Framework and the security algorithms are shown. The dashed blocks were newly implemented for this work, the dotted blocks were enhanced to meet our needs.

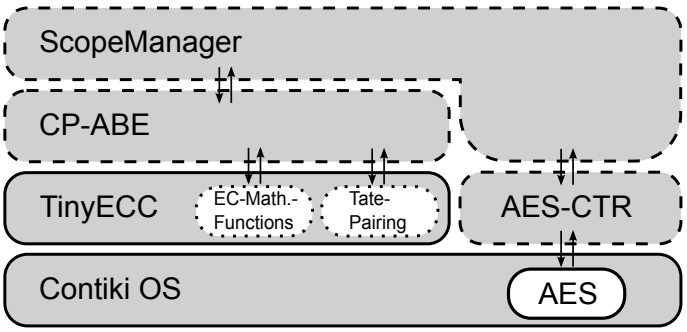



**Figure 4.16:** Relationships between Scopes and security algorithms

As the AES front-end is just an interface to the hardware algorithm we do not expect any performance issues in this direction. Therefore, we will show some implementation details for AES, but focus for our performance considerations on the tandem of TinyECC and CP-ABE.

#### 4.3.1.1 Implementation details

**Symmetric Cryptography (AES)**

The hardware AES blocks on Econotag and Tmote Sky only support encryption of data. Using AES in counter mode (AES-CTR, [71]) allows to use AES as block cipher and only AES encryption is needed to encrypt and decrypt data. As both hardware blocks have

different interfaces we defined a unified interface that implemented the AES-CTR mode with the specific hardware block, see listing 4.9.

```
1 /* typedefs */
2 typedef struct aes_key_s {
3         uint32_t val[4];
4 } aes_key_t;
5
6 /* Function prototypes */
7 uint8_t aes_init();
8 void aes_ctr_cipher(uint8_t *data, uint16_t len, aes_key_t *key);
```

**Listing 4.9:** AES Counter Mode API.

The `aes_init` function is only needed for the initialization of the advanced security module (ASM) of the ARM7 of the Econotags, for the Tmotes this function just returns. The logic of the AES counter mode is implemented in the `aes_ctr_cipher` function. The Tmote implementation is straight forward, as this hardware block just waits for a key and data to encrypt it. The ASM implementation needed quite more effort, as the ARM module needs to run a selftest before it can be used and it offers some configuration options that have to be set correctly, before the module can be used.

#### Elliptic Curve Cryptography (ECC)

The mathematical operations on elliptic curves are provided by the TinyECC library. It provides an extensive API but not all functions are used. Therefore, we will shortly summarize the APIs provided, and show in listing 4.10 a selected set of functions that are used in CP-ABE. Additionally, simple assignment functions are omitted.

The standard TinyECC distribution was enhanced to additionally provide a full Tate pairing and to compute random points on the elliptic curve, besides some basic operations.

TinyECC provides APIs for operations on big natural numbers (NN) and big natural complex numbers (NN2). These define arithmetic operations, data assignments and number theoretic operations modulo a number. They are used by the following APIs. ECC API (ECC) provides operations on the elliptic curve and the Tate pairing API (TP), which is used to provide the computation of a bilinear pairing on the elliptic curve. The SHA1 API (SHA1) that also comes with TinyECC, provides the calculation of the SHA1 digest. All these APIs are used for our CP-ABE implementation.

#### Attribute-based Encryption (CP-ABE)

The CP-ABE implementation for the sensor nodes was done from ground up. To ensure proper implementation and calculation we used the implementation of CP-ABE from the PBC library [75] as guidance and reference implementation for debugging. This way we

```
1  /* SHA1 */
2  void SHA1_Reset(SHA1Context *context);
3  void SHA1_Update(SHA1Context *context, const uint8_t *message, uint32_t len);
4  void SHA1_Digest(SHA1Context *context, uint8_t digest[SHA1HashSize]);
5
6  /* Functions on elliptic curves */
7  void ECC_init(void);
8  Params* ECC_get_param(void);
9  void ECC_Random_PointMul(Point *P);
10 void ECC_Random_Hash(Point *P, NN_DIGIT *h);
11 void ECC_mul(Point *P0, Point *P1, NN_DIGIT *n);
12 void ECC_add(Point *P0, Point *P1, Point *P2);
13
14 /* Functions on big natural numbers */
15 void NNModRandom(NN_DIGIT *b, NN_DIGIT *c, NN_UINT digits);
16 void NNModAdd(NN_DIGIT *a, NN_DIGIT *b, NN_DIGIT *c, NN_DIGIT *d,
17               NN_UINT digits);
18 void NNModSub(NN_DIGIT *a, NN_DIGIT *b, NN_DIGIT *c, NN_DIGIT *d,
19               NN_UINT digits);
20 void NNModMult(NN_DIGIT *a, NN_DIGIT *b, NN_DIGIT *c, NN_DIGIT *d,
21               NN_UINT digits);
22 void NNModNeg(NN_DIGIT *a, NN_DIGIT *b, NN_DIGIT *c, NN_UINT digits);
23 void NNModInv(NN_DIGIT *a, NN_DIGIT *b, NN_DIGIT *c, NN_UINT digits);
24 void NNMult(NN_DIGIT *a, NN_DIGIT *b, NN_DIGIT *c, NN_UINT digits);
25 void NNDiv(NN_DIGIT *a, NN_DIGIT *b, NN_DIGIT *c, NN_UINT cDigits,
26               NN_DIGIT * d, NN_UINT dDigits);
27
28 /* Functions on big natural complex numbers */
29 void NN2ModRandom(NN2_NUMBER *a, NN_DIGIT *b, NN_UINT digits);
30 void NN2ModMult(NN2_NUMBER *a, NN2_NUMBER *b, NN2_NUMBER *c, NN_DIGIT *d,
31               NN_UINT digits);
32 void NN2ModExp(NN2_NUMBER *a, NN2_NUMBER *b, NN_DIGIT *c, NN_DIGIT *d,
33               NN_UINT digits);
34 void NN2ModInv(NN2_NUMBER *a, NN2_NUMBER *b, NN_DIGIT *c, NN_UINT digits);
35
36 /* Tate pairing */
37 TPParams* TP_getTPparams(void);
38 void TP_TatePairing(NN2_NUMBER *res, Point *P, Point *Q);
```

**Listing 4.10:** TinyECC math library API.

could compare our algorithm's results with the intermediate and end results of the PBC library implementation and ensure proper functionality.

For the attribute-based encryption there are two APIs available. The algorithm API that supports the core algorithm functions and the serialization API that allows the serialization and deserialization of memory structures that have to be transmitted with the cypher text to allow proper functionality. Here we will focus on the algorithm API, see listing 4.11. The corresponding memory structures are omitted because of their textual size and number of attributes.

```
1  /* Function prototypes */
2  void cpabe_init(void);
3
4  void cpabe_setup(cpabe_pub_t *pub, cpabe_msk_t *msk);
5  void cpabe_keygen(cpabe_prv_t *prv, cpabe_pub_t *pub, cpabe_msk_t *msk,
6                    char** attributes);
7  void cpabe_enc(cpabe_cph_t *cph, cpabe_pub_t *pub, NN2_NUMBER * m,
8                    char *policy);
9  int cpabe_dec(cpabe_prv_t *prv, cpabe_cph_t *cph, NN2_NUMBER * m);
10
11 void cpabe_revocation_update(NN2_NUMBER *g_hat_alpha_prime, Point *delta,
12                    cpabe_pub_t *pub, cpabe_msk_t *msk);
13 void cpabe_pub_update(NN2_NUMBER *g_hat_alpha_prime, cpabe_prv_t *pub);
14 void cpabe_prv_update(Point *delta, cpabe_prv_t *prv);
```

**Listing 4.11:** CP-ABE Algorithm API.

As described in chapter 3.1.1.3 the first set of methods is closely related to the functions provided by the CP-ABE algorithm, where the `cpabe_init` prepares the memory structures of the algorithm. The `cpabe_setup` generates a public and a master key. `cpabe_keygen` generates a private key out of the public key and a set of attributes. Functions `cpabe_enc` and `cpabe_dec` allow the encryption and decryption of a message. In the Scopes Framework this message is the symmetric key for the AES encryption.

The second block of methods relates to the revocation of keys as described in section 3.3. `cpabe_revocation_update` updates the master key and returns the parameter to update all private and public keys. This is split, so that the parameter can be distributed to the nodes that will stay in the network and a coordinated update can be executed. To update the public and private keys the functions `cpabe_pub_update` and `cpabe_prv_update` are used. After the update, nodes with the old key cannot communicate with the rest of the network or new nodes, as all use the new parameter keys.

#### 4.3.1.2 Methodology and Setup

For the evaluation of TinyECC on Contiki we reconstructed most of the parameters of the original tests to only have changed the underlying operating system and programming language. Therefore, we used the Tmote Sky as test platform and ran the test cases that came with the TinyECC distribution with the same elliptic curve. We selected the best and the worst case in the sense of runtime. So all optimizations of TinyECC are enabled or all optimizations are disabled. This will give us an overview of the performance we then compare to the results of the original TinyECC implementation reported in [72].

The original results [61] for the Tate pairing algorithm are only given for Imote2 and MicaZ nodes, so we cannot directly compare these. We decided to use the Tmote Sky as our smallest platform as a reference to estimate the performance. We again used the test case and elliptic curves that came with TinyECC to evaluate the performance. This time with all optimizations enabled.

For the results of CP-ABE we evaluated it on four platforms, namely Tmote Sky, Z1, Stargate and a PC. On the Tmote Sky we ran the stable version 2.4 of Contiki. The Tmotes are similar to the well known TelosB nodes, the only difference is that TelosB have a maximum frequency of 4MHz. We used the Tmotes at 4MHz to get results that are equivalent to the TelosB platform and did additional tests at the Tmotes' maximum frequency of 8MHz. Similarly to the Tmote Sky nodes, the Z1 nodes use an MSP430 microcontroller, but a more recent, faster version. It's maximum frequency is 16MHz and we did tests here at 8 and 16MHz to compare the results to the Tmotes. On the Z1 we used Contiki 2.5, as the Z1s are supported by Contiki starting with this version. Additionally, to support the new MSP430 a recent C compiler is necessary. The Stargate was chosen, due to its similarity with the Gumstix Connex and Imote2 sensor nodes. The experiments were executed on an embedded Linux at 200 and 400MHz, were 400MHz is the maximum for the Stargate. As a desktop PC sized device we did tests on a 2,200MHz Core2Duo, where only one core was used for our experiments.

Because of the diversity of platforms, operating systems and build environments, not all experiments and analyses could be performed on all platforms.

| Sizes | Small | Medium | Large |
|---|---|---|---|
| Properties | 1 | 5 | 7 |
| Policy | 2 | 5 | 8 |

**Table 4.5:** Definition of property and policy set sizes

The most important part of our results is the runtime evaluation. It was done on all platforms and with three different sets of properties shown in table 4.5. One set represents an CP-ABE access policy used to restrict access to a ciphertext. If a node wants to decrypt the ciphertext it has to have a key with properties matching the access struc-

ture. These access structures are represented as threshold gate, see section 3.1.1.3. The *small set* has a property `attrib1` and the policy used is `attrib1 1of1`. The *medium set* has five properties and its policy is `attr1 attr3 attr9 attr4 2of4`. The *large set* has seven properties and its policy is the exploded version of `attrib4 > 6`, see section 3.1.1.3. This settles in eight terms for an 8bit unsigned integer. We did also experiments on ROM size and static and dynamic RAM usage and used a script from TinyOS to retrieve this data. This was done on all platforms except the PC. To estimate energy consumption on the Z1 and Stargate platforms, we calculated $U \times I \times t$ or $P \times t$ for the Stargate platform. Both equations return milli-joules and are based on voltage (U), current (I) and execution time (t). The power (P) equals $P = U \times I$.

The Tmotes are the only platform available in this evaluation where TinyECC provides assembler code to speed up computation, so we show a performance comparison for this platform. The Z1s MSP430 uses an advanced assembler instruction set that seems to be incompatible, as we could not get the assembler code to run on them. Also the X-Scale processor on the Stargate seems to have little differences from the Imote2 one.

For our experiments we obtained a 192bit and a 512bit curve that are both suitable for pairing operations on elliptic curves, one from the reference implementation of CP-ABE, the other from the TinyECC distribution. The 512bit curve is rated secure for pairings, but puts too much load on low power sensor nodes, so that we also did the experiments with the 192bit curve.

All our experiments were repeated 10 times and the results were averaged. The code used was the same for all platforms, except for methods provided by the operating systems and in the case of Tmote Sky the assembler optimization.

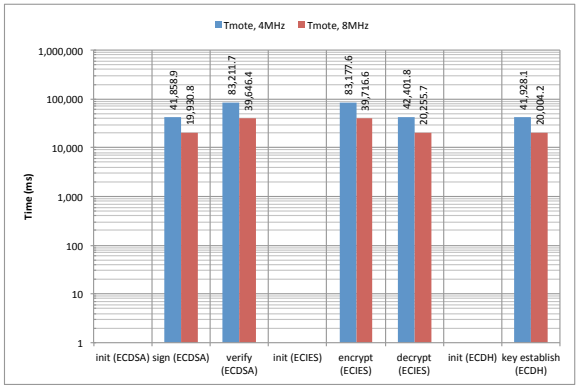
#### 4.3.1.3 Performance Evaluation

First, we show the performance evaluation for the TinyECC distribution and afterwards for the CP-ABE implementation.

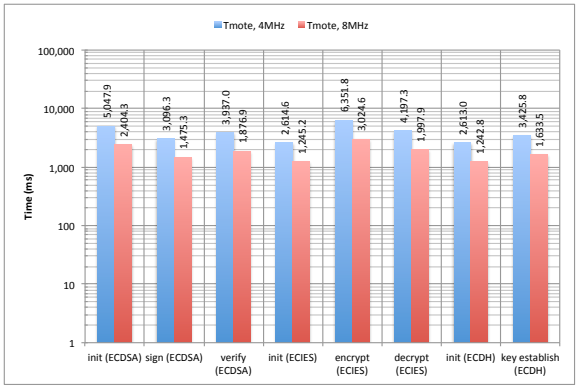
##### TinyECC on Contiki

The evaluation of TinyECC on Contiki was done on a Tmote Sky with 160bit elliptic curve parameters. We will first show our results in figures 4.17(a) and 4.17(b), and then compare them to the reported values in [72] in figures 4.17(c) and 4.17(d).

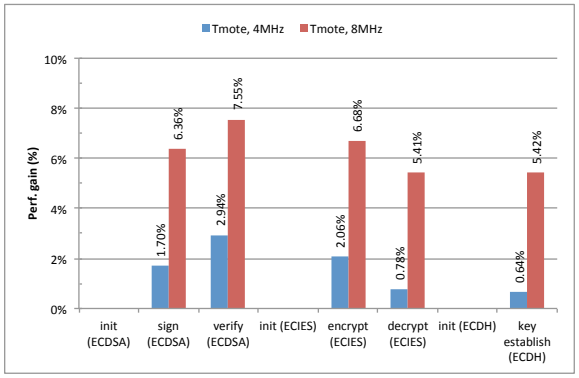
Figures 4.17(a) and 4.17(b) show the timing results of all three Tiny ECC algorithms. These are ECDSA, a signature scheme, ECIES, an encryption / decryption algorithm, and ECDH for key agreement using elliptic curves. In figure 4.17(a) all optimization switches of TinyECC were turned off. This results in zero runtime for the init functions, as no pre-computations are required. In this experiment encryption / verification, and signing / decryption take almost the same amount of time. Comparing this to figure 4.17(b) where all the optimizations are enabled, revealed that the encryption is here the



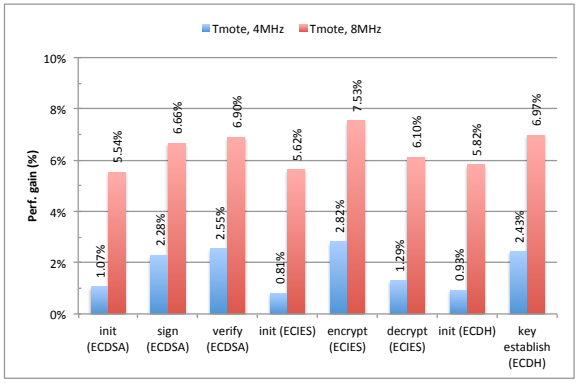

(a) Timing results, without all optimizations.




(b) Timing results, with all optimizations.




(c) Performance gain vs. original TinyECC, without all optimizations.




(d) Performance gain vs. original TinyECC, with all optimizations.

**Figure 4.17:** Results of TinyECC on Contiki.

most expensive operation. Also the performance increased for all operations more than ten times, note the different scales in the two figures. But with all optimizations the initializations also need a considerable amount of time. E.g. the initialization of ECDSA is the operation with the second highest costs overall. The advantage here is, that the initialization operation only has to be performed once. Having a look at all values the times at 8 MHz are consistently less then 50% of the times at 4 MHz.

Comparing the results with the originally reported ones can be seen in figures 4.17(c) and 4.17(d). First, we note that in all cases and operations we have at least a slight gain in performance. For the 4 MHz results the gain varies between 0.64% and 2.94%, which is not very significant and at least confirmed the results from the original paper. However the 8MHz results show a gain of 5.41% to 7.55%. This in contrast is a significant gain achieved by porting the code to C and Contiki.

Here we were able to show that TinyECC is performing slightly better on Contiki than on TinyOS and we made sure that our base library is fully functional.

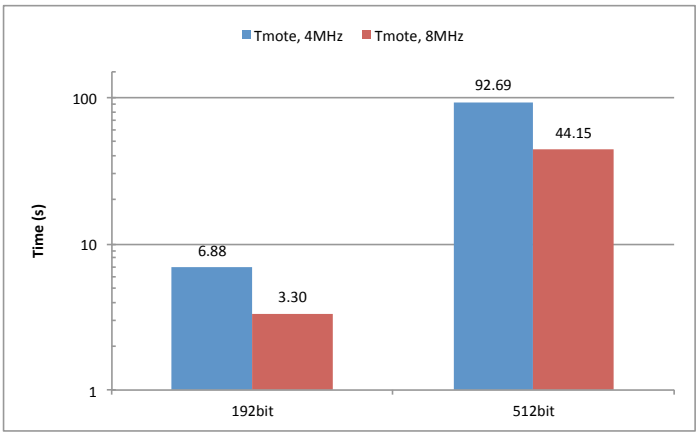



**Figure 4.18:** Timing of Tate Pairing with all optimizations.

**Tate Pairings in TinyECC.**

For our next evaluation we had no numbers to directly relate to. But still the results in figure 4.18 showed for a 192bit Tate pairing computation that it is roughly as expensive as the encryption operation from basic TinyECC. The time needed for the computation of a pairing in a 512bit curve took with 92.7s and 44.2s more than 13 times longer than for the small curve. Depending on the application, even these large values can be accepted in a WSN. E.g., for a monitoring application that runs for a long time it may not matter if from time to time such a long operation takes place.

## CP-ABE on Contiki

We started our evaluation of the CP-ABE algorithm with an estimate of the theoretical computation costs. Next we show detailed measurements of execution time on all platforms and, as far as possible, with all property sets and curves we presented earlier. After that we give numbers for the energy consumption on the Z1 and Stargate platforms. Last, we show ROM sizes for all platforms (except the PC), and RAM requirements for Tmotes and Z1s.

**Theoretical Operation Costs.**

Table 4.6 shows the used computations over all CP-ABE operations. We defined three classes of operations: multiplication, additions and pairings. As we have seen in the last section, pairings are very expensive, and so they are in a separate class. The setup operation is the only constant one. It uses 1 pairing and 4 multiplications. Key generation depends on the number of properties to be included in the secret key and it uses 2 multiplications and 1 addition, plus 3 multiplications and 1 addition per property. Encryption and decryption highly depend on trees, the policy tree for encryption and the minimal satisfied policy tree for decryption. The tree used for decryption is smaller or has the same size, this depends on the actual policy and property set used to satisfy the policy. The base costs for encryption are 2 multiplications and 1 addition. For each node

except the tree root we add 3 multiplications and 1 addition, plus 2 multiplications and 1 addition for every leaf node. Leaf nodes represent properties and inner nodes threshold gates. For decryption the costs are a bit higher, but in general this tree is smaller than for encryption. The base cost is 1 pairing and 2 additions, plus 6 multiplications and 2 additions for all nodes except the tree root, and 2 pairings, 1 multiplication and 1 addition for all leaf nodes.

| Operations | setup | keygen | enc | dec |
|---|---|---|---|---|
| Pairings | 1 | 0 | 0 | 1<br>leaf: 2 |
| Mult | 4 | 2<br>prop: 3 | 2<br>(n-1): 3<br>leaf: 2 | 0<br>(n-1): 6<br>leaf: 1 |
| Add | 0 | 1<br>prop: 1 | 1<br>(n-1): 1<br>leaf: 1 | 2<br>(n-1): 2<br>leaf: 1 |

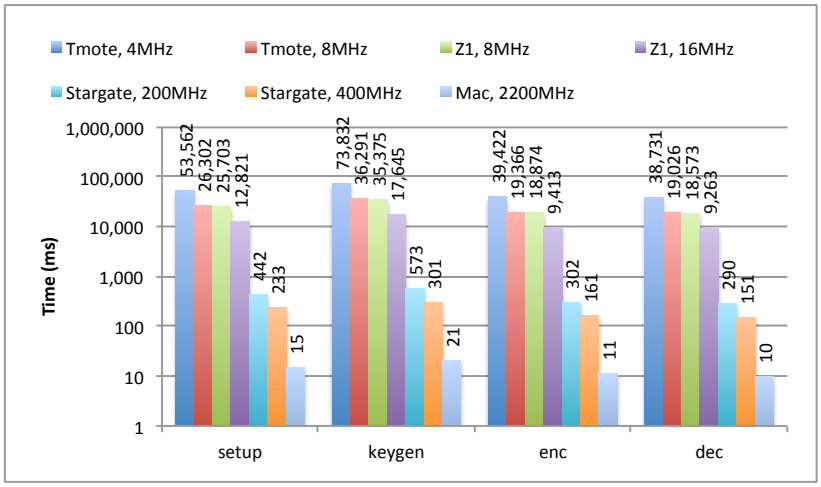
**Table 4.6:** CP-ABE theoretic cost estimation.

**Measured Operation Costs.**

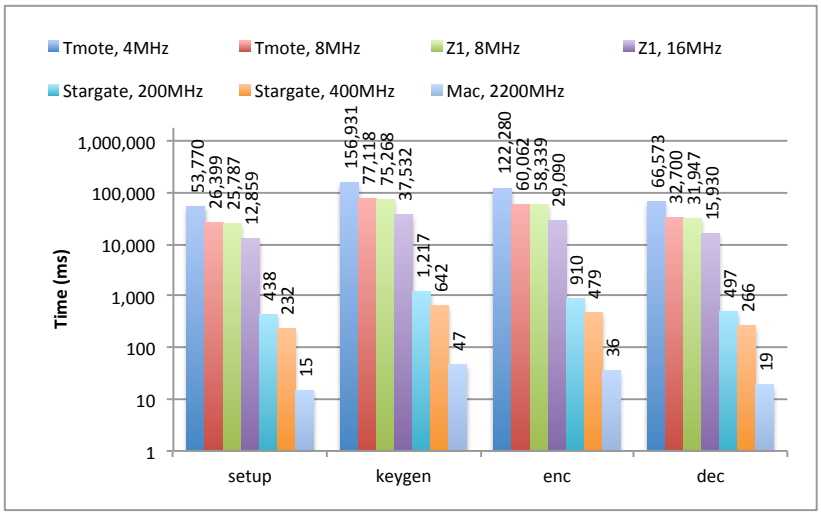
Next we will look at figure 4.19 which shows the execution times on all platforms with a 192bit elliptic curve. In our experiments we used all optimizations of TinyECC, except the assembler optimization, as this is only available on Tmotes and except the curve optimizations, as these are not available for the curves we use for pairing computations. Figures 4.19(a), 4.19(b) and 4.19(c) show results of the small, medium and large property sets, where for the large set Tmotes and Z1s had not enough RAM available, so they are missing in this figure.

The execution time of setup was constant over all three sets. This was to be expected, as the same keys with the same sizes had to be generated. Looking at the key generation we can see, that this was by far the most expensive operation in our setup, even though it was not using pairing. For a Tmote with 4MHz and the medium set it took more than 2.5 seconds to compute the secret key. Comparing the three sets for key generation and encryption we see a steady increase of execution time, as these two functions compute values on properties and the complete policy tree. At decryption execution time increased from small to medium, but for the large set it took the same time as for the medium. This was because for decryption only properties that were needed to satisfy the policy are included in computation. These trees were of the same size for the medium and large sets.

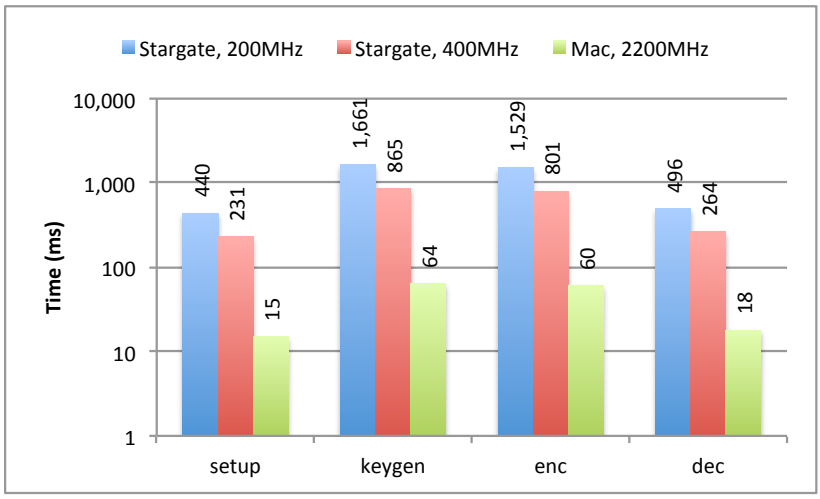
Comparing the different platforms, we recognized the different execution times. Of course for a desktop PC this computation is no big deal, so we got 21ms, 47ms and 64ms for the key generation, as the slowest, and 15ms for setup, as the fastest operation. The




(a) Small set.




(b) Medium set.




(c) Large set.

**Figure 4.19:** CP-ABE execution time for 192bit curves.

Stargate achieved also very good performance values with times between 0.5 and 1.5 seconds. For Tmotes and Z1s we see a constant improvement, doubling the frequency halves the execution time. At 8MHz, where we had data for both platforms, timings were almost equal. The small difference arises from the different compilers used, as the compiler for the Z1s produced smaller code than the one of the Tmotes.

Besides this the execution times for these small nodes were very high. But keeping in mind, that we only have to decrypt a new scope key once in a while for re-keying or when a new scope is created these may still be acceptable values. Of course nodes should run at full power for this kind of operation.

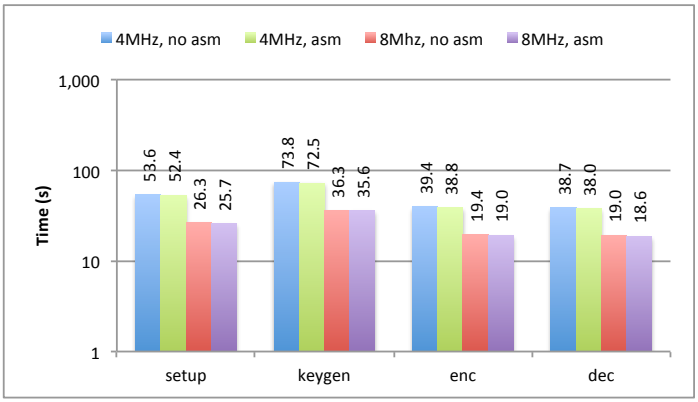


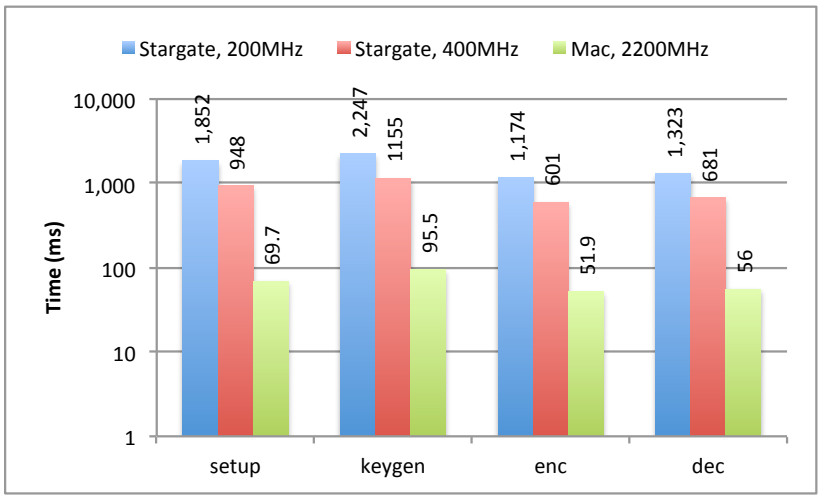

**Figure 4.20:** Influence of assembler optimization on execution time.

After obtaining these numbers we wanted to check if the provided assembler code for several methods inside the TinyECC library can speed up execution times. As we only had working assembler code for the Tmotes, we compared execution times with and without assembler code for the small property set, see figure 4.20 . In general using the assembler optimization speeds up the execution, but the gain we achieve is small. On average we gain 1 second. With execution times between 40 and 150 seconds this is no significant improvement.

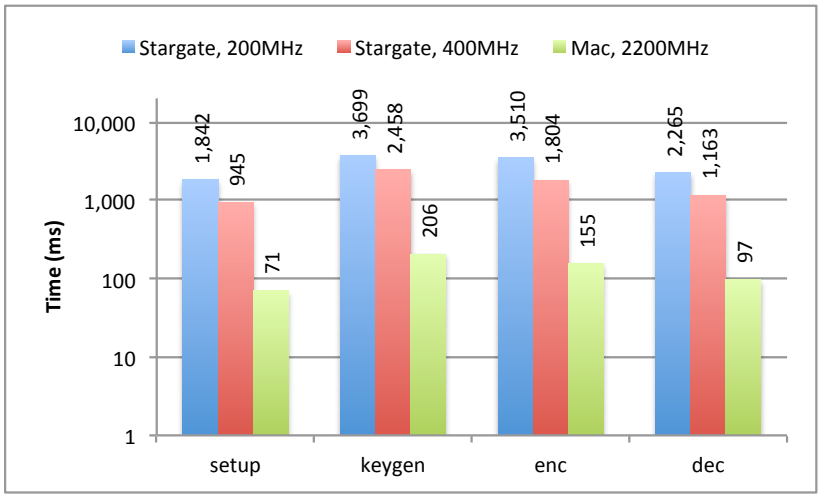
For execution times of 512bit curves figures 4.21(a), 4.21(b) and 4.21(c) show the results for the small, medium and large sets. As for the large set with the 192bit curve Tmote and Z1 did not have enough RAM to successfully run the experiments. The CP-ABE operations show a similar footprint in this experiments as for the 192bit case, just the absolute timings are higher. Looking at them reveals for the Stargate values of 1,848ms, 5,360ms, 4,801ms and 2,266ms for the four operations of CP-ABE with the large property set. This shows that the algorithm is definitely feasible on more capable platforms, and can handle even larger property sets with tolerable execution times.

**Energy Consumption.**

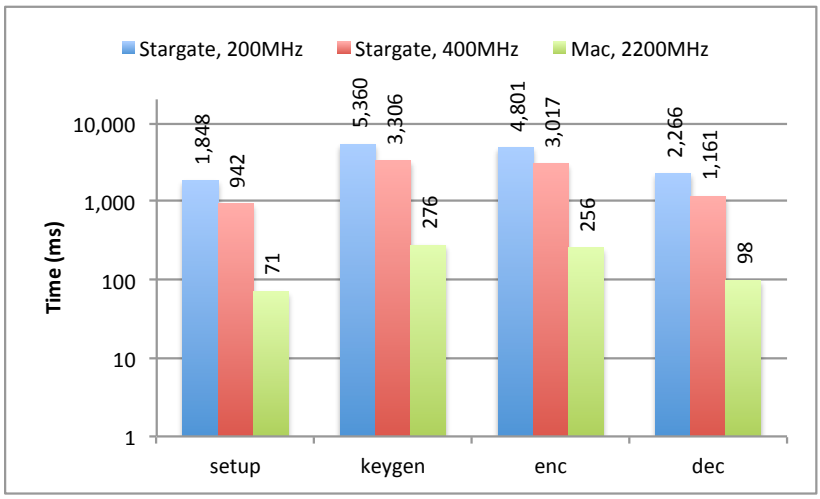
An important issue in WSNs is energy consumption. Therefore figures 4.22(a) and 4.22(b) show energy consumption estimations for Z1 and Stargate. For Z1 we used the ENERGEST framework [28] to retrieve the data for our calculations. For the Stargate we used the timings we got from the previous experiments. We omit the results for




(a) Small set.

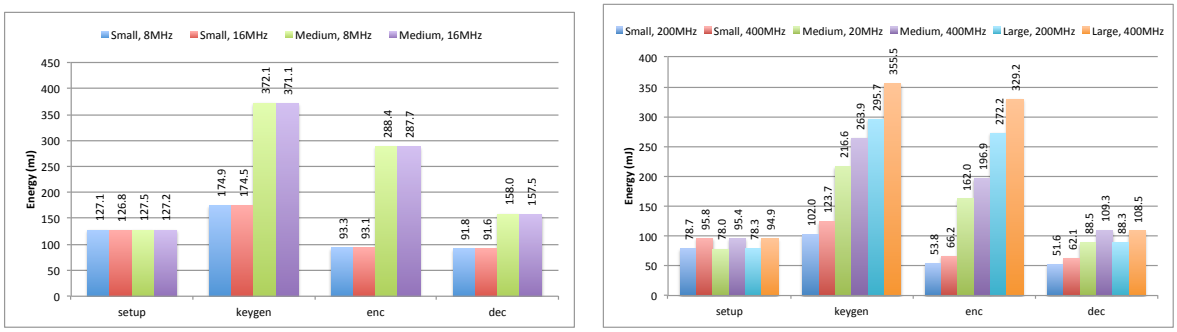



(b) Medium set.




(c) Large set.

**Figure 4.21:** CP-ABE execution time for 512bit curves.




(a) Z1, ENERGEST results

(b) Stargate, timed energy estimation

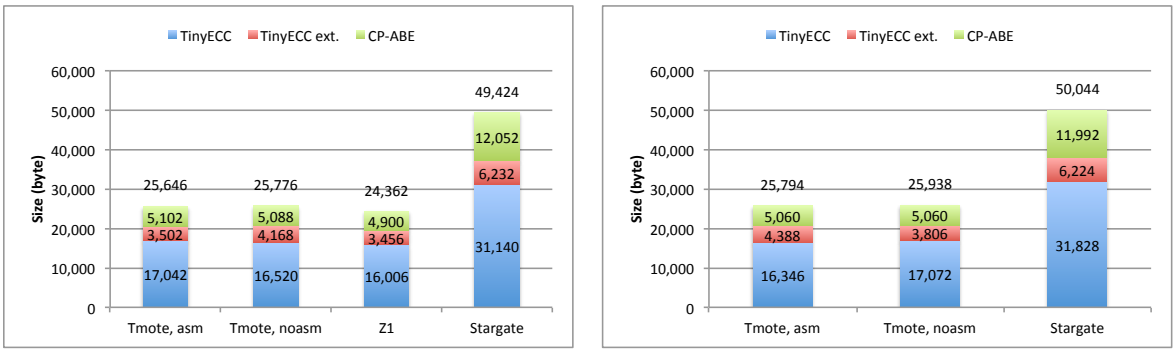
**Figure 4.22:** CP-ABE energy consumption.

Tmotes here as they use the same microcontroller family as the Z1s and their current draw by frequency scales linear, so the results for 8MHz are the same for both platforms. As mentioned before the datasheet for Z1s microcontroller [125] specifies a current draw of 0.41 for 8MHz and 0.82 for 16MHz what results, with the previously observed doubling of performance with a doubling of frequency, in the same energy consumption for 8 and 16MHz, shown in figure 4.22(a). For the Stargate we got 178mW for 200MHz and 411mW for 400MHz from the datasheet [52]. The results in 4.22(b) are all much smaller than the ones estimated for Z1s, even the values for the large properties set is smaller than the values for Z1s on the medium set. On the other side the current draw in sleep mode for the Stargate's microcontroller is 100 times higher than the one from the Z1s. What was also nicely reflected here was the same performance of decryption in the medium and large sets. Here you also have roughly the same energy consumption.

**Memory Consumption.**

After evaluating the runtime characteristics, figures 4.23(a) and 4.23(b) show the ROM sizes of the implementation. The data shown is for all platforms except the PC, as the PC platform is not constrained in memory. Compilation for Z1s with the 512bit curve did not work out, because of the static RAM demands. Three sizes are shown per entry. These are *TinyECC* for all code that comes from it. *TinyECC ext.* includes all extensions done to TinyECC, these are minor additions for CP-ABE, but also the Tate pairing. Last *CP-ABE* is all code implementing this algorithm. This categorization is also used for static RAM evaluation (figure 4.24).

First thing to mention, compiling the code for 192bit or 512bit introduces no big changes, so ROM size is stable. In general ROM sizes for Z1 and Tmote should be the same, however the code for Z1s is compiled using a more recent major version of the compiler that employs better code optimizations, which results in slightly smaller code. With the Stargate we change from a 16-bit to a 32-bit architecture which introduces also a doubling in ROM size. The 25kB for the Tmotes occupy more than half of its ROM, on the Z1s the ratio is better, here only a quarter is used. For the Stargate with 32MB flash the 50kB are no problem.




(a) 192bit curve parameters. (b) 512bit curve parameters.

**Figure 4.23:** CP-ABE ROM size.

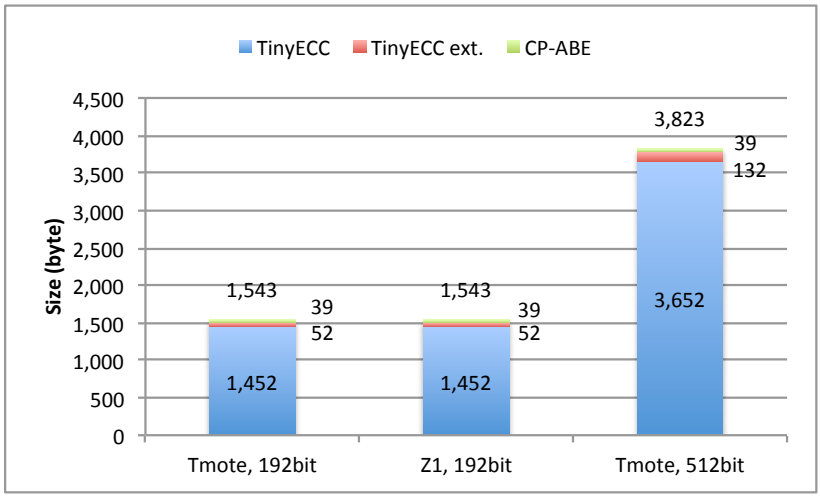



**Figure 4.24:** CP-ABE static RAM size.

For the evaluation of RAM we decided to split it into static RAM, that is included in the sensor node image, and dynamic RAM, that contains only the keys that are used. As we only have node images on Tmote and Z1 figure 4.24 contains only values for these platforms. Actually most of static memory is allocated by TinyECC for optimization purposes. CP-ABE or the Tate pairing use only a small amount of static RAM. Of course with the increase to a 512bit curve this amount also increases. The static memory is independent of the three property sets. In table 4.7 we show the different key sizes for the different elliptic curves and platforms. Besides the master key and public parameters, which have a static size, the secret key and ciphertext depend on the number of properties or elements of the policy tree. For example, together with table 4.5 we calculated, that the secret key in the large set, for 512bit on the Stargate had a size of 2384 bytes. For the small set these were still 424 bytes.

| Size in byte | 192bit Tmote, Z1 | 192bit Stargate, PC | 512bit Stargate, PC |
|---|---|---|---|
| Master key | 78 | 84 | 204 |
| Public params | 208 | 224 | 544 |
| Secret key | 56 | 64 | 144 |
| Property elem. | 108 | 120 | 280 |
| Ciphertext | 108 | 120 | 316 |
| Policy node | 130 | 164 | 324 |

**Table 4.7:** CP-ABE key structure sizes.

### 4.3.2 SecScopes Framework

In the following we will evaluate the SecScopes Framework, the cryptographic enhanced version of the Scopes Framework. As most of the Scopes Framework is reused, we will especially highlight the differences between the two Frameworks and discuss the integration of the security measures.

#### 4.3.2.1 Implementation details

The need for security in WSNs and for the Scopes Framework was motivated in chapter 1. While the need for integrating security and the Scopes Framework appeared to be obvious, finding the right algorithms that support the scopes concept was not at all obvious. Finally, we chose CP-ABE as described above, even if it stretches the capabilities of very small sensor nodes. On more capable nodes it can easily be handled and the advantage of perfectly fitting how the Scopes Framework works in a distributed way and using properties for authentication makes CP-ABE a very good choice.

The integration in the workflow of the Scopes Framework is shown in chapter 3. The integration in the code base is split in two parts. First, the extension of the Scopes Framework's API and second, the addition and extension of an elliptic curve math library and the implementation of the CP-ABE algorithm for small-scale devices, such as sensor nodes. The implementation details of the TinyECC library and the CP-ABE algorithm were explained earlier in section 4.3.1.1.

As of the extension of the SCR depending on the properties specified for a scope the message does not fit anymore into one packet. To be able to transfer such an oversized SCR we implemented an automatic fragmentation that splits messages that are larger than 90 bytes into maximum 6 fragments of the same size. This led in our experiments to a high number of dropped SCRs as the corruption of one network packet of the fragmented SCR disrupts the whole SCR. In consequence the join rate dropped drastically for the SecScopes Framework. Therefore, we inserted an integrity layer in the routing stack to provide a checksum for each SCR fragment. This way we can drop single cor-

rupted network packets and recover them from another source in the network. With this small extension we achieved comparable join rates to the previous implementations with no further influence on the protocol and almost no influence on performance as the calculation of the checksums is done in hardware.

Besides introducing security in Scopes it was also important to be backwards compatible with the Scopes Framework, so tiny sensor nodes can still be part of a network, even if they are not capable of encryption or do not require it. Therefore, the APIs shown in listings 4.5, 4.6 and 4.7 are still valid. We only extended the task API to support secured tasks and connections.

```
1 /* interface function declarations */
2 int scopes_register_sec(subscriber_id, struct scopes_application *task,
3                         cpabe_prv_t *prv, cpabe_pub_t *pub);
4 bool scopes_open_sec(subscriber_id, superScope_id, scope_id,
5                         char *sec_policy, void *specs, spec_len, flags, ttl);
```

**Listing 4.12:** SecScopes Framework task API.

With the `scopes_register_sec` call a task registers itself with the SecScopes Framework as with the previous `scopes_register` call. But in addition each task can specify its own private and public keys to be used by the CP-ABE algorithm. This allows different tasks to co-exist not just on the same network, but also on the same sensor node and keep their sent and received data concealed from each other.

To create a secure scope the call `scopes_open_sec` is used. Besides the standard parameters the CP-ABE decryption policy is added here. As described in chapter 3 the decryption provides the two AES scope-keys. Each message sent via the secure scope is encrypted by AES using one of the two keys. Which one to choose is stored in the key-id field of the message header. The key id is generated for each key by computing its hash value. The two keys enable the exchange of the used scope-key on a configurable regular basis or on demand, if a scope-key is suspected to be compromised.

Details of the different algorithms and libraries used to achieve the security needs are explained in section 4.3.1. Here we explain some peculiarities that came up while setting up the following evaluation. These are due to resource constraints on our sensor nodes. As we have seen before, the time for execution highly depends on the microcontrollers' frequency. Mostly sensor nodes are limited to 8 or 16MHz if you are looking for power saving chips and extended life-time of batteries. But you also have powerful microcontrollers with 50MHz, 200MHz or even more. They have way less life-time while on standard batteries, but this may be dealt with by using larger, non-standard batteries to achieve the same life-time or even rechargeable power supplies.

To support even quite small sensor nodes the CP-ABE algorithm is kept modular. Therefore, depending on the role and capabilities of a sensor node it may only be configured with a subset of functionality. For example, a small sensor node that just senses data and does not need to create a scope may only get the decryption routines

of CP-ABE to save space in the microcontroller's code space. It can be part of a SecScopes network, but it only joins a scope and not creating one. The node can still create data and send it encrypted, forward encrypted data and process it. Other, more capable nodes, can use encryption and decryption. Thus, they can additionally encrypt, they can create new scopes, but use more code space. Functionality like creation of keys is most likely located on a computer outside the sensor network for security reasons on the one hand, and the lack of use of these functions in the sensor network on the other hand, as these are only needed to enable new sensor nodes to join the network.

The sensor nodes available in our testbed are quite limited in code space. This means that the code for the complete Scopes Framework and the CP-ABE algorithm does not fit together on a sensor node. Even a reduced set, like the aforementioned limitation on the decryption only does not fit. Therefore, we implemented a stripped CP-ABE module that allows the use of all the functionality of CP-ABE, such as setting and evaluating the access-structures. While the stripped CP-ABE module has the same memory footprint as the original module, it allows us to leave most of the TinyECC math library out of the firmware image to save code space. The encryption and decryption functions of CP-ABE are replaced by a simple xor-function thus we do not use plain-text for transmission of the SCR data. The AES transport encryption is still in place and is fully functional. In summary, the only thing we replaced is the CP-ABE encryption/decryption calculation: the evaluation of the access structure is still in place and fully functional. The time needed by the nodes to calculate encryption or decryption of the CP-ABE algorithm is not assumed to have a significant influence on the evaluation results. Let us analyze this. First, encryption is done on a node that wants to create a scope. It has to encrypt the scope properties. As the encryption takes place before the SCR is sent, the difference in time of sending the SCR has no influence on the performance of SecScopes. The message is just sent some time later, when the encryption is in place. Second, when receiving an encrypted SCR the routing schedules a potential resending of the SCR independently of the processing of the SCR with SecScopes. This means, a received SCR can be forwarded instantly, after receiving it completely. Therefore, this is not influenced by a working CP-ABE decryption at all. Third, processing a new encrypted SCR in SecScopes with the working CP-ABE decryption takes some minutes, after which the node may join the scope. Iff it joined, the scopes group key was obtained and the following data exchange is encrypted with this key and AES in a secure way. With our stripped version the computation of the decryption is done very fast due to the xor-operation and the node may join the scope. Still, it may join the scope only if the node's CP-ABE key matches the access-structure, as with the full CP-ABE. After this operation, the scope group key is obtained, and the data traffic is also encrypted with AES. This way the evaluation below does not take into account that the decryption would introduce a delay of some minutes before the nodes join the scope, but this has no influence on the decision taken if the node joins or not, and it has no influence on the data traffic inside the scope as this traffic is AES-encrypted. Fourth, the computation time of a CP-ABE encryption or decryption operation takes several minutes. This may render the microcontroller

unresponsive to other requests, as Contiki employs cooperative multitasking. But, as the CP-ABE algorithm is not a monolithic computation, but consists of many calls to different functions that also highly rely on loop-processing, the code can be split into smaller parts with short run-time, handing control back to the main processing. Here processing of other requests or triggering the next processing step in CP-ABE computation can be executed. Our implementation is currently not prepared for respecting the cooperative multitasking, but this is a standard and well-known procedure in the embedded industry.

#### 4.3.2.2 Methodology and Setup

To evaluate the influence of the security measures we employed in the SecScopes Framework we repeat the same tests that we have done for the Scopes Framework version, see section 4.2.2.2 for test methodology and setup. We use the same scope definitions as listed in section 4.2.2.2. For data logging we use the Motelab infrastructure of our testbed as before.

#### 4.3.2.3 Performance Evaluation

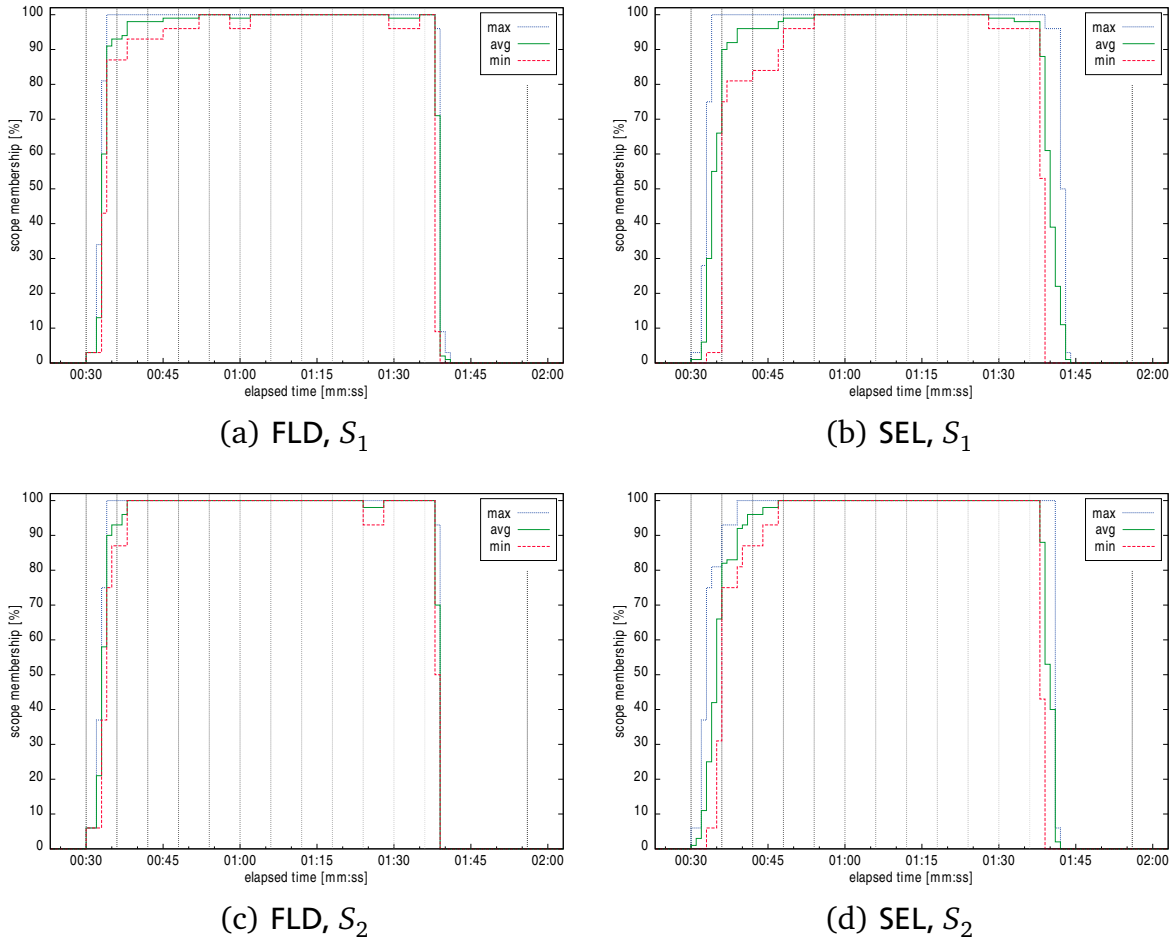
In this section, we use the results of the Contiki version of the Scopes Framework as baseline for our security enhancements (see section 4.2.2.3). The evaluation of SecScopes itself was carried out in the same way as the one for the Scopes Framework and therefore, we only summarize the evaluation parameters.

**Scope Creation, Removal and Maintenance.**

This paragraph looks at the operations of scope *creation*, *maintenance* and *deletion*. To evaluate the reliability we employ the same measurements as with the former Contiki results. We show the average of 5 test runs $\phi_{avg}$, the minimum $\phi_{min}$ and maximum $\phi_{max}$ percentage of memberships achieved over all test runs. Of course, this does not reflect the single experiment run, but shows worst and best case.

In our experiments with the SecScopes Framework we send the scope creation at t = 30 seconds. The scope is then kept alive for 68 seconds and is then deleted by a deletion message at t = 98 seconds. The scope refresh interval is set to 6 seconds like in the Scopes Framework experiments and reflected in the graphs by light grey vertical lines. The time span the scope is alive allows for 11 scope refresh cycles. If a node misses the scope deletion message the scope times out 20 seconds after the last refresh message and is then deleted at t = 116 seconds. This is also shown in the graph by the last vertical line on the right.

In figure 4.25 we show the results for the reliability tests using scopes $S_1$ and $S_2$. First thing to mention is that we again reach almost steady the 100% scope memberships of nodes in the network. Although it takes some time while creating a scope to reach it. Flooding (FLD) achieves a faster startup behavior then our selective flooding (SEL). FLD needs 1 second to reach more than 80% membership, SEL takes 3 seconds. For the




(a) FLD, $S_1$ (b) SEL, $S_1$

(c) FLD, $S_2$ (d) SEL, $S_2$

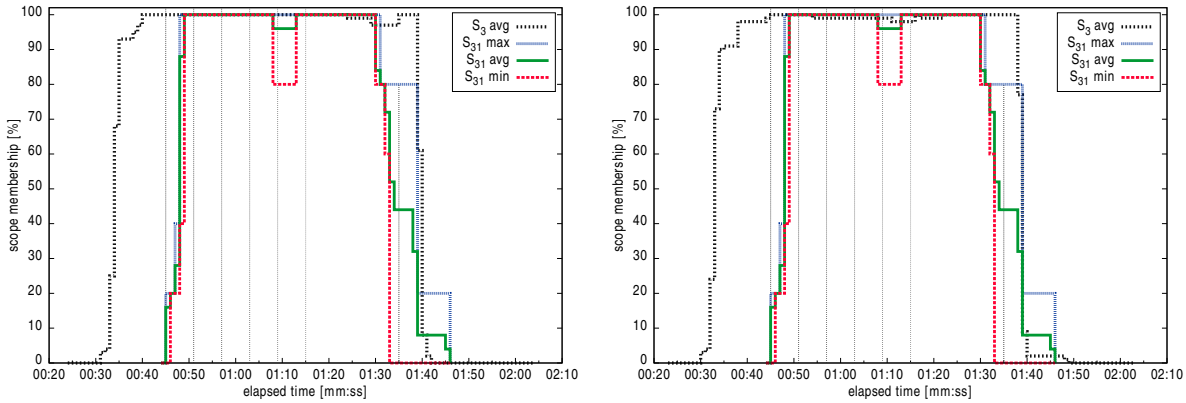
**Figure 4.25:** Scope creation, removal and maintenance with SecScopes

removal of the scope FLD takes 2-3 seconds while SEL takes 3-4 seconds. FLD is a bit faster compared to SEL with our security enhancements, but the results are still similar as SEL also uses a flooding algorithm.

Compared to the results of the Scopes Framework it is obvious that the creation, maintenance and deletion process is less stable, especially for creation. But this is what has to be expected, as the creation process has to transfer a much bigger SCR than without the security enhancements. This can lead to multiple messages sent and these messages have to be received by each node before they can decrypt and evaluate the data. As the decryption of messages adds some additional time to all processing it explains the growth of overall processing time.

#### Nested scopes.

For the evaluation of the reliability of nested scopes with FLD and SEL we create the scope $S_3$ as the parent scope with the same timings as in the setup before. 15 seconds after the creation of $S_3$ the nested scope $S_{31}$ is created. Also 5 seconds before the scope $S_3$ is deleted the nested scope $S_{31}$ is deleted.




(a) FLD, $S_{31}$ (b) SEL, $S_{31}$

**Figure 4.26:** Scope creation, removal and maintenance for nested scopes with SecScopes

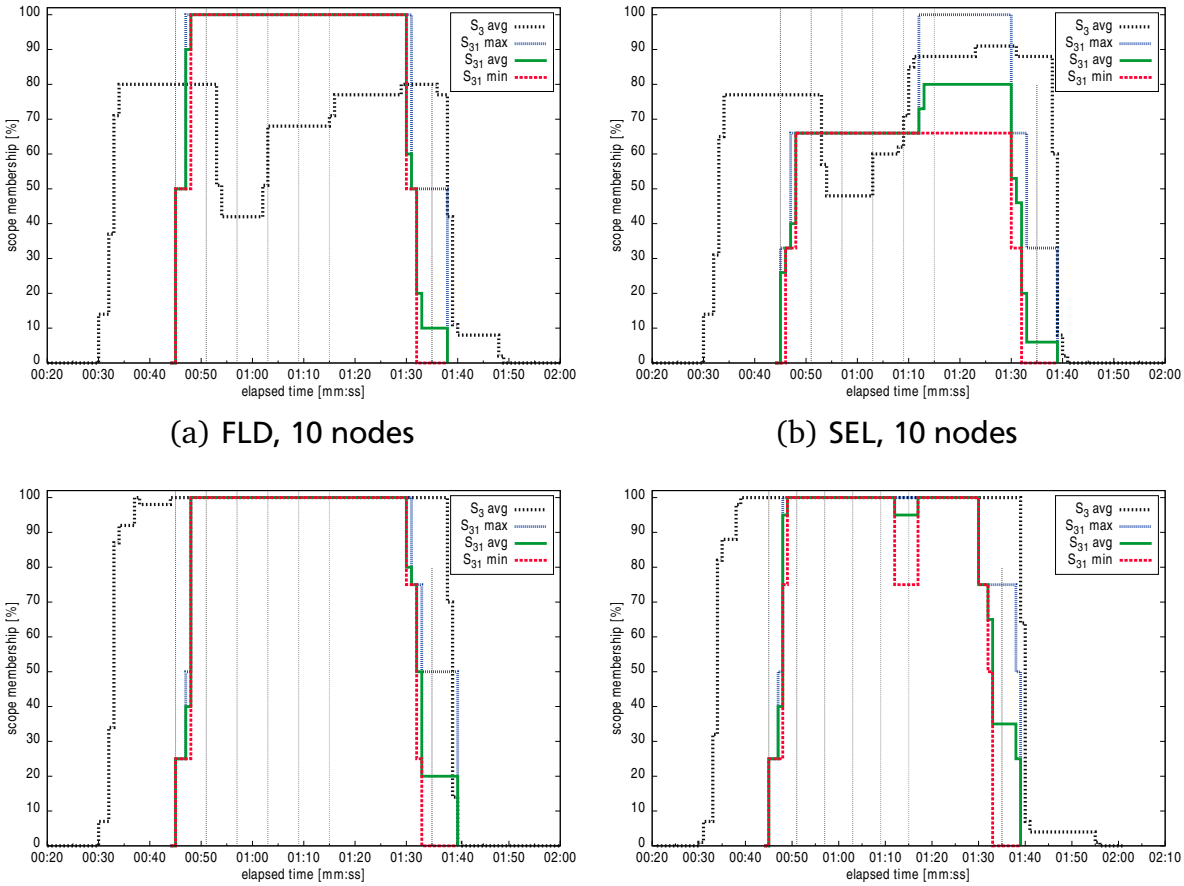
Figure 4.26 supports the observations from the previous paragraph that the process of creation, maintenance and deletion it less stable compared to the Scopes Framework and a single operation takes more time to execute. The used routing algorithm seems to make no difference, as the graphs are very similar.

The stability of the nested scope depends on the stability of the parent scope. The parent scopes here are quite stable, but when a node gets removed from the parent scope the nested scope reacts sensitive to this instability. On the one hand it may loose the nodes removed from the parent scope as members, if they are also member of the nested scope and on the other hand the nested scope may suffer from instabilities in its own refresh mechanism.

In comparison to the Scopes Framework evaluation the average scope reliability is not as constant as the results of these experiments, also ignoring the extended time for creation and deletion. But the minimum curve of scope $S_{31}$ shows that the larger refresh messages lead to a higher probability of being missed by some nodes. As this drop can be seen in both figures for FLD (see figure 4.26(a)) and SEL (see figure 4.26(b)) in the same distance to the creation of the nested scope, it seems that the refreshes for the nested scope do not reach all the members right after creation of the scope. This hypothesis is supported by the time window of 20 seconds between this drop and the creation of the nested scope. A definite cause for this behavior could not be found in our data. Presumably the number of messages after creation of the nested scope leads to collisions and consequently the loss of refresh messages.

#### Network density.

For this evaluation we use the same node densities $\rho_A = \{1, 2, 4\}$ as specified for the Scopes Framework previously in table 4.4. The graphs for scopes $S_1$ and $S_2$ are similar to scope $S_3$, therefore, we omit them and just show the graphs for the nested scopes $S_3$ and $S_{31}$. Figure 4.27 shows the graphs for $\rho_A = 1$ and $\rho_A = 2$, the graphs for $\rho_A = 4$ are the same as shown for the nested scope evaluation in figure 4.26.




(a) FLD, 10 nodes

(b) SEL, 10 nodes

(c) FLD, 20 nodes

(d) SEL, 20 nodes

**Figure 4.27:** Scope density for $S_3$ and $S_{31}$ with SecScopes

As we have seen in the evaluation of the Scopes Framework, here also density $\rho_A = 1$, figures 4.27(a) and 4.27(b), is not suitable for our system. The average curve for scope $S_3$ shows that clearly. It may not be obvious, but the performance of the subscope $S_{31}$ is the same for FLD and SEL. The nodes joined in the FLD test-runs are all the nodes belonging to the scope, whereas the nodes joined in the test-runs for SEL are not all nodes that may belong to the subscope. Therefore, the subscope with SEL cannot reach the maximum percentage of members compared to FLD.

The difference between SEL and FLD for $\rho_A = 2$ and $\rho_A = 3$ is just minor. In the raw numbers there is a visible difference between the two densities. Unfortunately, it is not clearly visible through the graphs. It can be observed that the density of 4 is getting less stable than the density of 2, but compared to the density of 1 these two can be considered stable. At first sight this appears strange, since more nodes in the area seems like a better coverage and a better chance to forward messages. But starting somewhere in between the densities of 2 and 4 there is a tradeoff between this enhanced coverage and the number of collisions of messages. This means a number at the higher end between 20 and 32 nodes in the area of our experiment is the sweet-spot in terms of scope reliability.

Compared to the previous results for the density measures this was not observed, but this can be explained by the increase of messages in the SecScope version. Depending on the scope definition 3 times the number of messages is exchanged during scope creation. This behavior could not be observed with the smaller number of messages in the Scopes Framework version.

**Scope resource costs.**

The resource costs for SecScopes include the scope software modules and routing modules, like in the previous evaluation (see 4.2.2.3) plus the additional code for the security algorithms.

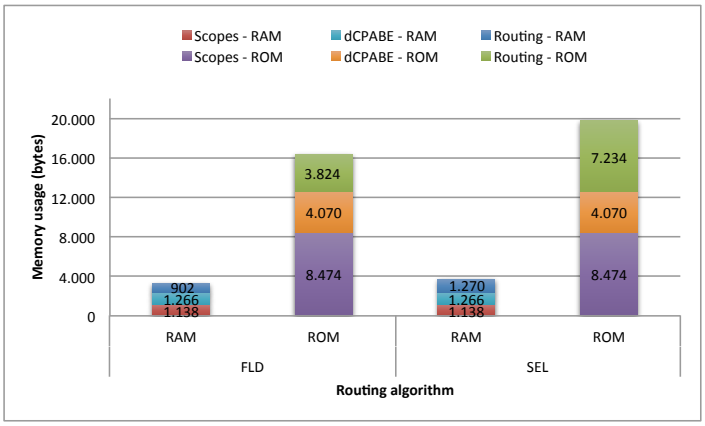



**Figure 4.28:** Resource costs with SecScopes (RAM / ROM)

The RAM and ROM usage for the scopes modules, see 4.28, is quite static, compared to the previous evaluation. This is ok, as the code only changed to call the new security measures and some minor bug fixes. More interesting is the increase of the routing algorithms' ROM usage of about 3.5kB. This increase includes the implementation of fragmentation and CRC checks in the routing algorithms to compensate for the larger messages, that are sent in SecScopes Framework. The additional amount of RAM is about 0.8kB. The security algorithms themselves introduce additional 4kB of ROM and 1.2kB of RAM. This includes the stripped CP-ABE and interface code for the hardware AES-unit. For the full version of CP-ABE the ROM footprint would increase about roughly 25kB for the 192bit curve. Compare results from figure 4.23.

In total the security measures implemented and supporting measures to compensate for the higher network load result in 7.5kB more ROM and 2kB RAM usage in this firmware image.

**Scope traffic.**

The scope traffic tests for SecScopes are carried out the same way it was done for the Scopes Framework tests. It is described in detail in chapter 4.2.1.3. A small difference had to be implemented. The intervals used to distribute the messages and which are used to form the different traffic shapes had to be enlarged to 4 seconds. In the

previous tests the intervals were set to 2 seconds. This was necessary due to reliability issues. With the smaller intervals we were not able to achieve 5 complete test-runs, as the increased SR traffic collided more often with the test patterns for our evaluation. Stretching the time frame spreads the network traffic and lowers collisions. As the general setup stays the same the test runtime is also extended to 160 seconds. Intervals without activity are not shown in the figures. Unfortunately the extended intervals lead to jumps in the graphs, as you will notice below.

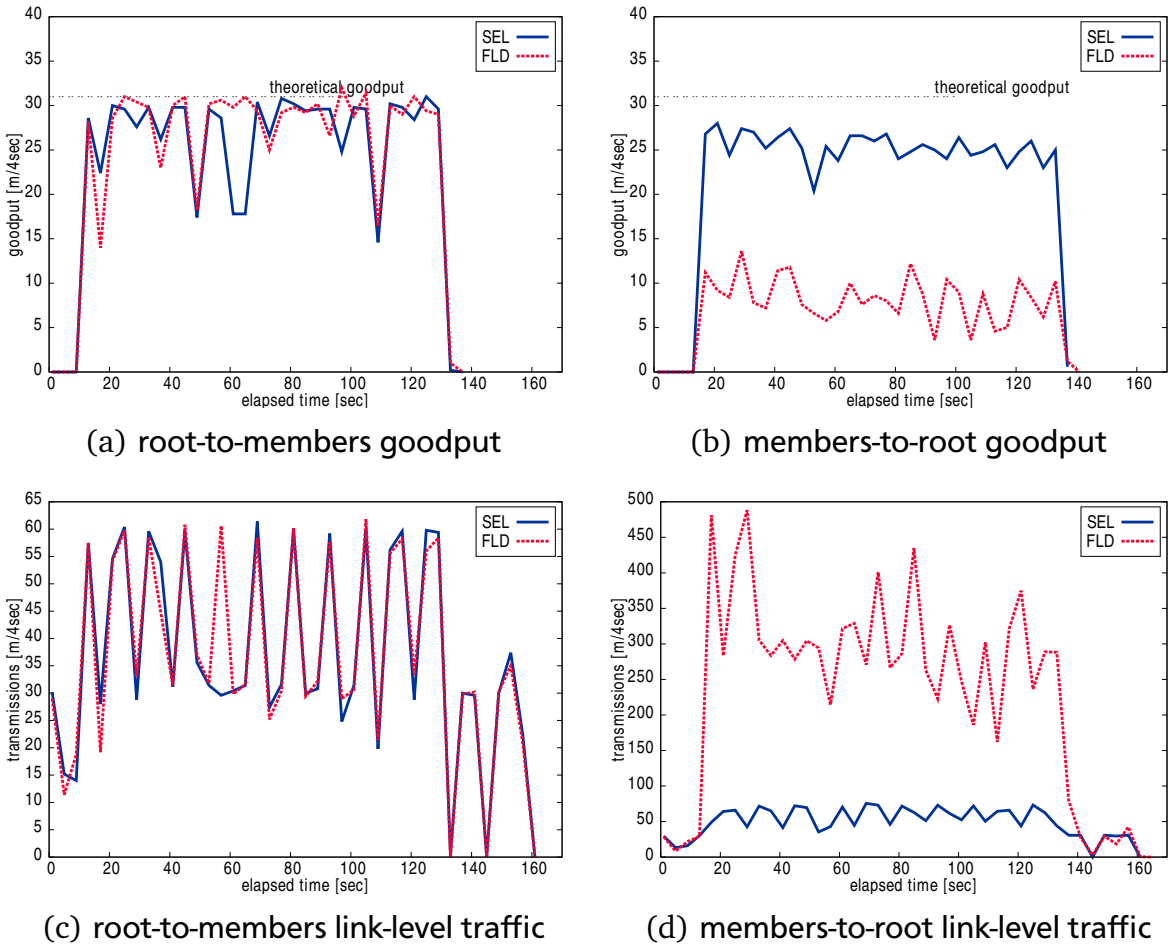
We evaluated the two directions root-to-member (RtM) and member-to-root (MtR) with two common communication patterns found in wireless sensor networks: burst and periodic. After creation of the scope we allow it $\alpha = 15$ seconds to stabilize. After this our traffic tests are carried out. In the end a tear down phase of $\beta = 23$ seconds allow remaining messages to be delivered before the scope is closed.

These tests show the performance of data transfers inside the scope. Compared to the Scopes Framework without security the only change here in SecScopes is the encryption of data. But, the AES encryption does not significantly change the size of the data. In contrast to the added CP-ABE encryption for scope creation, where the size of the SCR was, e.g., tripled depending on the access structure used. Therefore, we expect only minor performance changes due to the added stability features, like CRCs, and the AES encryption.

**Periodic traffic.**

The RtM tests, send $M = 31$ messages to the member nodes, whereas in MtR tests each scope member sends $M = 31$ messages to the root node. As described before, each sending node posts a timer that is triggered with a frequency $\lambda = \frac{t_3 - t_2}{M} = 4$ seconds. To reduce network collisions, message transmission is still delayed randomly by $\varepsilon$ (with $\frac{\lambda}{8} \leq \varepsilon \leq \frac{7\lambda}{8}$) by the sending application if sending direction is MtR. We again use the scope $S_1$ to stress the routing protocols.

The first two figures in 4.29 show the results of the goodput experiments. As already mentioned the messages are shown in a 4 seconds interval. Compared to the previous Contiki version without security the figures look quite similar. Figure 4.29(a) shows the RtM messages and a performance of 96.80% for FLD and 94.18% for SEL. FLD is still slightly better than SEL, but the difference to the previous experiment is -1.90% for FLD and +6.68% for SEL. Figure 4.29(b) shows the goodput results for the RtM direction. Here FLD reaches 28.66% and SEL 87.33%. This is a difference to the previous experiment of +13.68% for FLD and +9.59% for SEL, which is roughly a double of goodput for FLD and a good improvement for SEL. Besides the improvements the results are comparable to the one of the Scopes Framework experiment without security. The next two figures show the link-level traffic in the periodic traffic scenario. Both figures show the same behavior as in the Scopes Framework experiment without security, with a less distinct decrease of messages in figure 4.29(d). The integrity layer introduced to stabilize the scope join rates also enhance transfer rates, as the overall improvements shown by these figures suggest.




(a) root-to-members goodput

(b) members-to-root goodput

(c) root-to-members link-level traffic

(d) members-to-root link-level traffic

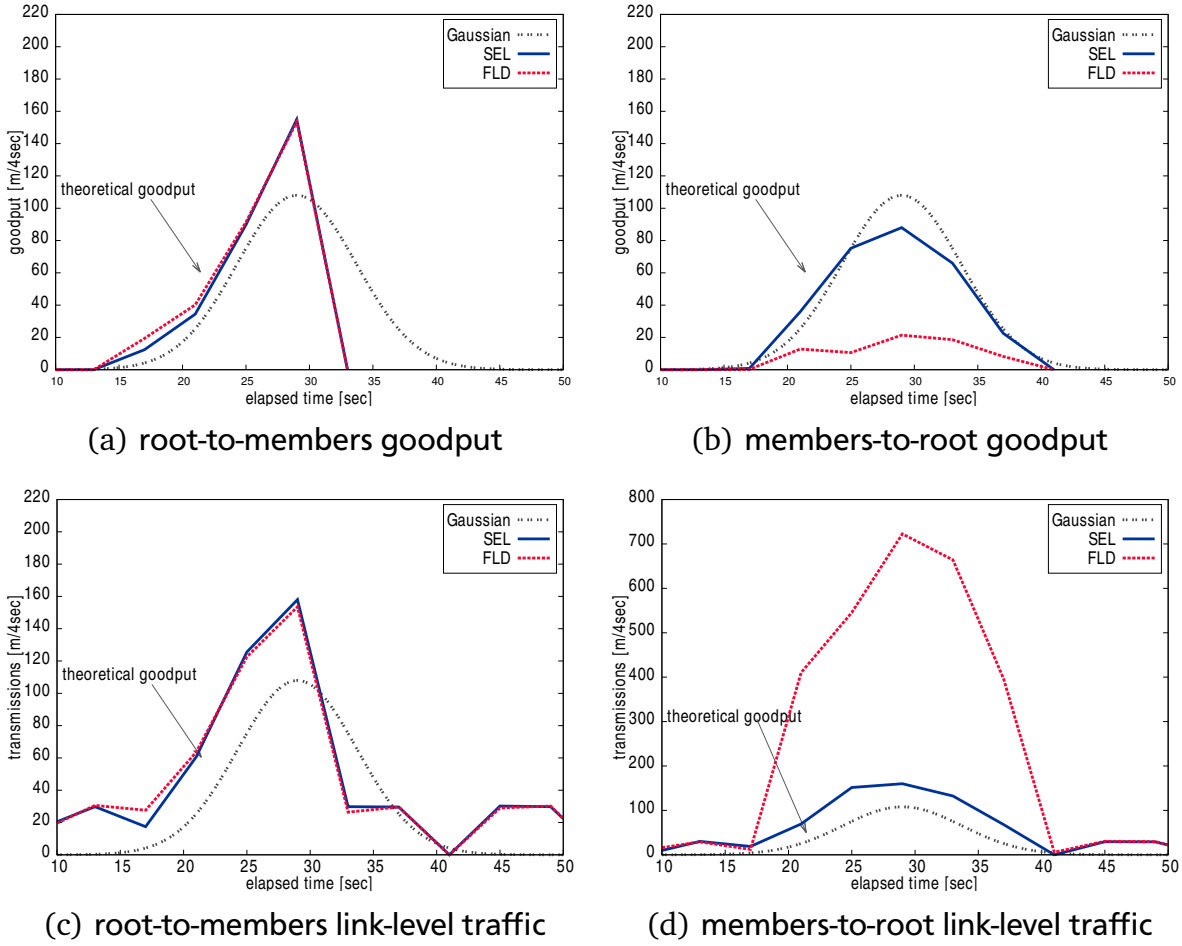
**Figure 4.29:** Goodput and link-level periodic traffic with SecScopes

**Bursty traffic.**

The bursty tests use the same procedure used in the tests of insecure scopes. The message bursts are modeled after a gaussian function with center=5, standard deviation=3.2 and a peak value of 7. This amounted to a total of M=15 messages. As with the SecScopes periodic tests the bursts are ideally sent in intervals of $\lambda = 4$ seconds. Scope $S_1$ is used for the experiments. As seen in the experiments before, in the RtM setup only the scope root sends the bursts, while in the MtR setup, all 31 (member) nodes sent the bursts back to the scope root. The messages in the figures of 4.30 are evaluated in 4 second slots for the same reason as it was done in the periodic tests before.

The RtM figures 4.30(a) and 4.30(c) look a bit awkward, but this is due to the difference in analysis compared to the Scopes Framework tests without security. In this tests we had to use 4 second intervals for the tests due to the large messages in the scope creation process. The larger intervals allowed less collisions in the process and a comparable result. Unfortunately, this increases the impact of timer deviations on the embedded sensor nodes compared to the PC real-time clock. In the two figures the data from the time slot with the burst-peak and the time slot after that fall together into the

same point value of second 29. Therefore, the following entry is roughly 0 and the previous slot exceeds the theoretical maximum.




(a) root-to-members goodput

(b) members-to-root goodput

(c) root-to-members link-level traffic

(d) members-to-root link-level traffic

**Figure 4.30:** Goodput and link-level bursty traffic with SecScopes

The curves in figure 4.30 represent data and scope management traffic in the link-level and data-traffic only for the goodput figures. The theoretical goodput values of the data-traffic are represented by the gaussian curve. Figures 4.30(a) and 4.30(b) show the goodput for RtM and MtR burst traffic.

For the RtM tests both routings could improve their goodput, see 4.30(a). SEL achieved 93.46% and FLD 97.63%. This is an increase of 18.02% and 15.40%, respectively. The figure for the link-level traffic reflects the same results, figure4.30(c). The graphs for SEL and FLD are almost identical, this is possible as for the RtM communication but routings use flooding. With the newly introduced integrity measures collisions and such things are highly reduced and the influence on the result is reduced. FLD is still ahead of SEL, but it is only 4.17% difference.

For the MtR tests with SecScopes SEL could improve its performance compared to the Scope Framework tests by 16.01% to 92.44% goodput, see 4.30(b). This is the result of the integrity measures that were taken in this version of the Framework. In contrast FLD lost 4.51% compared to the previous test and only achieved 23.01%. The

MtR communication of FLD does distribute each message through the whole network and with the additional load of the large SCR messages for SecScopes it is not able to keep its performance. This can also be seen in the link-level traffic, figure 4.30(d), the integrity measures could reduce the number of sent messages for FLD roughly by half, due to decreased collisions and resends, but the huge number of messages in the network still get many messages dropped and decreases the received goodput.

## 4.4 Summary

In this section we first showed the feasibility of scopes with the initial version of the Scopes Framework for the SOS operating system. We evaluated different aspects of Scopes and found it feasible for common low power wireless sensor networks. The mechanisms for our scoping approach worked stably and the resource requirements were acceptable. With the experiences from the SOS-version we built an enhanced version on the Contiki operating system including lessons learned, especially in the area of routing and scope creation. The evaluation showed slightly better results than our initial version. The better performance and stability lead us to use it as basis for our security enhancements.

Evaluating security algorithms matching our requirements showed quickly that they are quite computation intensive. As the sensor nodes we had available in number lack memory space and computation capabilities we had to replace these parts of the algorithms with a stripped version that use the message size, RAM footprint and traffic as the original algorithm for our evaluation. This replacement has no major influence on our further results, as the evaluation of the algorithms on more capable systems showed that they are feasible in low-power systems, like todays ARM Cortex-M Series, especially as most of them have hardware support for extended math operations used in the computations.

Adding the security mechanisms to the Scopes Framework and creating SecScopes, we expected computational limits as mentioned before, but what was unexpected in the first place was the large increase of data to exchange while creating a scope. Besides the ciphertext the access structures for the data are quite large. Therefore, with the size of data we had to introduce fragmentation of packets to the data exchange. And with the larger number of packets we generated more collisions. This resulted in less connection stability and in some early experiments in the inability to establish a scope. To ensure packet integrity, we had to introduce additional CRC checking to the routing process to achieve a better performance than the previous Scopes implementations. Finally, the larger messages and the cryptographic operations introduced also a longer computation time. This led to an increased slot time for the SecScopes tests to achieve comparable results.

The trends observed in our previous evaluations on SOS or Contiki, like the connection stability at different densities of the network, are now more clearly to see in the SecScopes evaluation. E.g., the SEL routing algorithm is less stable than in the non-

secure implementations and performs well at medium density, slightly worse at high density and poorly at low density.

In the creation and deletion phases of scopes FLD routing is more stable than SEL, but this is due to a huge number of messages compared to SEL. SEL still has a better goodput of messages than FLD due to fewer collisions. Therefore, SEL performs better than FLD, but still the density of the network influences the performance of the whole system and has to be taken into account when creating such a system.

Security not only requires more resources in memory and computation, but also in communication which leads to instability due to more data to be exchanged. Countermeasures for the loss of stability in communication have to be taken, like CRC checking or fragmentation of packets. With these measures stability improves beyond the stability of insecure scopes.

From the above experiments we learn that the Scopes Framework works best in an environment of medium density, as shown above. The size of messages sent in the network should be below 100 bytes. We used this maximum size for the SecScope experiments and observed fewer collisions with smaller sizes. The traffic pattern used (we used bursty and periodic traffic) has no direct influence on the performance, unless a high load of messages transferred increases the probability of collisions and message loss in general. Thus, saving bandwidth is mandatory for this kind of applications.

The Scopes Framework does not have special performance requirements, whereas SecScopes demand a capable processor with up to 200MHz for the cryptographic calculation to be finished in a reasonable amount of time. What time is reasonable depends on the scenario. We choose to evaluate the possibilities on low-power sensor nodes. As the major computational work has to be done at scope creation and maintenance it is important to configure the respective scope module parameters to match a given scenario.

To adjust the Scopes Framework or SecScopes to a scenario there are modules that can be customized and a choice of parameters of the scope module. The modules for routing, membership evaluation and scope application can be tailored to a scenario's needs.

To provide a high degree of freedom for users of the Scopes Framework the interfaces to the modules have been kept very simple. For example, the membership evaluation module receives the unprocessed data as byte-string and parses this string. The structure and data in this byte-string is owned and defined with this module. The framework only processes the binary result of the evaluation. Similarly, the scope properties that may be used in conjunction with membership evaluation is kept as a repository of configurable key/value-pairs. The scope applications contain the scenario-specific program logic, also called tasks. The routing can also be adjusted to the scenario's needs and its intended communication behavior. But here the Scopes Framework has some limitations that have to be fulfilled to work properly. For example, for scope creation there has to be a mechanism to send the SCR to all potential scope members. The data communication can then be defined independently, but the routing must manage the needed routing

information by itself. For these three modules interface definitions are provided by the Scopes Framework.

The ScopesManager module has a small set of parameters to optimize its behavior. Especially for SecScopes these parameters are vital for performance. In general these parameters should be set the same on all nodes of the network. First, we have `SCOPES_MAX_SCOPES`, which limits the number of concurrent (sub-)scopes in the network. The default is 5. Then there is the `SCOPES_MAX_SUBSCRIBER`, which limits the number of tasks that subscribe to a scope, the default is currently 3. With `SCOPES_TIMER_DURATION` the time of reevaluation of dynamic properties can be configured, the default is 15 seconds. The parameter `SCOPES_REANNOUNCE_FACTOR` configures the time intervals of scope refreshes, which is calculated by TTL of the scope times the factor in seconds. The default is 0.3. And specifically for SecScopes the `SCOPES_KEY_RENEWAL_INTERVAL` defines after how many scope refreshes the AES keys are renewed, the default is 0, which means every scope refresh is used to renew the keys.

# 5 Related Work

## Contents



In this chapter related work to our Scopes Framework will be presented. First, we will look at closely related approaches of network node grouping algorithms in common networking. After that we summarize important approaches in WSNs related to the Scopes Framework. Next is an overview of approaches that use so called mixed mode systems, these are systems encompassing components of widely varying capabilities in common wireless and non-wireless networks. Section 5.4 shows selected security approaches in common networking, and after that we summarize approaches introducing security to WSNs as librarys or similar to the Scopes Framework.

## 5.1 Structuring Computer Networks

The Idea of structuring a computer network is as old as the Internet. IP addresses are grouped in different classes and the Domain Name System (DNS) represents the Internet in a hierarchical naming scheme. The topology of the Internet with border routers and gateways reflects the physical structure of different organizations interconnecting their networks. But structuring in the sense of creating groups that can be reached by an identifier is not included in the end-to-end nature of IP-based networks.

We selected two important approaches that are related to our Scopes Framework. First the IP multicast that supports dynamic grouping in IP-based networks. And second, the Rebeca publish/subscribe middleware as representative for the general publish/subscribe mechanism, which is also a key building block of our framework.

### 5.1.1 IP Multicast

The idea of *multicast* was proposed by Deering [25, 26] to support sending data to a dynamic group of receivers via the Internet Protocol (IP). Key concepts in IP multicast are IP multicast group addresses, multicast distribution trees and receiver driven tree creation.

IP multicast enables one-to-many or many-to-many communication over an IP network. For multicast special blocks of IP addresses (ranges) are reserved. The sender uses one of these addresses to send its messages, and the receiver joins this multicast group by notifying a network router that it is interested in the multicast group with this specific address. This mechanism scales well without prior knowledge to whom the messages have to be sent or how many members belong to a group. The sender has to send its messages only once, the routers in between will take care of duplicating the data if necessary. The most commonly used transport protocol is UDP, but others are available.

The process of joining a multicast group is managed by the IGMP protocol [14,25,31]. The message routing is controlled by other protocols. To join a group a client has to send a request to its router. The router then tries to connect to the next router until a path from the client to the data source is established. If there are routers in between that do not support IGMP or multicasting, these routers can be tunneled by forwarding messages in a unicast packet to a router that supports multicasting. In newer versions of the protocol besides joining a multicast group also leaving and selection of data source were added.

#### Discussion

The multicast grouping scheme is one of the basic building blocks of many approaches proposed for WSNs. Especially for grouping of network nodes multicasting is essential, as this mechanism allows efficient communication in a group.

Multicasting in WSNs is not the same as in IP networks, as most of the time there is no infrastructure in WSNs like routers or gateways in IP networks. Therefore, the WSN nodes themselves have to take care of the join process. In WSNs you encounter mostly scenarios where many senders (sensing nodes) are sending to a small number of data sinks (e.g. base stations or cluster heads). In IP based networks this is vice versa, most of the time a node sends data to many data sinks, e.g. a streaming server serving several clients.

### 5.1.2 Publish/Subscribe

Over the last one and a half decades there were many publish/subscribe systems in conventional networks and peer-to-peer-networks, such as Siena [16], Scribe [17], Hermes [102] or PADRES [58] in conventional networks. And SpiderCast [20], StAN [83] or Vitis [105] in peer-to-peer-networks. But still, as the Scopes Framework was inspired by Rebeca, we will focus on a discussion of this approach for the pub/sub domain.

#### Rebeca

The *Rebeca* publish/subscribe (pub/sub) middleware was started in 1999 as a PhD project by Gero Mühl and Ludger Fiege [87]. Since then it was used by different researchers and was therefore extended and changed several times. The last version was built by a team from Berlin Institute of Technology [95].

Rebeca provides so called components. These can be publishers or subscribers depending if they act as producers or consumers of information. The communication between components is done by notifications. They consist of name-value pairs and are specified by the producer or derived automatically from the fields of the notification to be published. Components can use an interface to asynchronously publish notifications. The notifications are transmitted by the notification service provided by Rebeca. This service transmits notifications from producing components to the consumers with matching subscription. The notification service is implemented by Rebeca using a set of cooperating brokers distributed over the network and each locally managing a set of components. The brokers are connected among each other via an overlay network and exchange published notifications and information about issued and revoked subscriptions. Additionally each broker keeps track of active subscriptions of connected neighbor brokers. This information is gathered in a content-based routing table. Hence, a published notification can start at the publisher's local broker and then hop-by-hop be transmitted to every broker and their local components that have a subscription for the content of the notification.

#### Discussion

Scopes were defined as part of Rebeca to provide structuring to an otherwise unstructured Pub/Sub environment. Providing the means for managing a distributed event based system and restricting the visibility of events to provide a first level of privacy. In

the Scopes Framework the ideas of Scopes from Rebeca were transferred to WSNs. Here nodes can subscribe to a group by matching the properties of a sensor node to the group-properties. If they match, the node is allowed to join. As WSNs are self-configuring and highly distributed this is a decision taken on the node itself. This also assumes a friendly network where no adversaries are present. For hostile network environments further measures have to be taken as we have seen in the SecScopes extension in section 3.

## 5.2 Structuring Wireless Sensor Networks

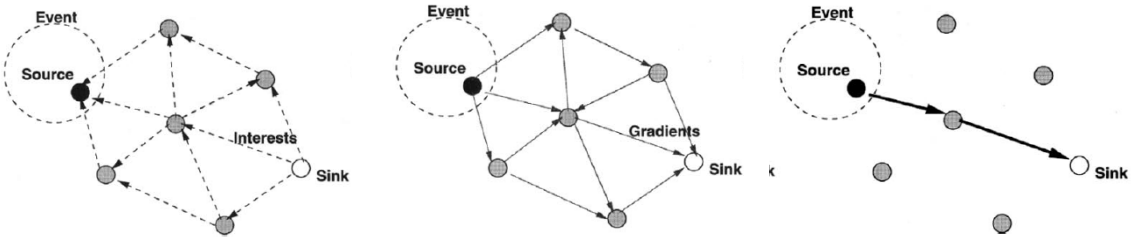
In wireless sensor networks there are different needs to structure a network. Mostly it is to reduce the use of bandwidth or send-time (and thus energy) in a network with scarce resources. Structuring can even be achieved using a single routing algorithm. By creating local clusters that are connected by a backbone network of so called cluster heads, this is a simple approach to reduce the load in the network as local sensor nodes only talk to their cluster head and the cluster heads then talk to the base station. Other approaches, such as the use of location-based routing, are valid alternatives for structuring a network. The structuring achieved by routing algorithms is done mostly for the purpose of the previously mentioned reduction of network usage or optimizing routes to better utilize the wireless medium. Therefore, we will look at different routing schemes used in WSNs that have a structuring component.

From the routing algorithms in WSNs more advanced approaches and frameworks have emerged over time and serviced WSN applications with different kinds of structuring notions. The structuring in these approaches is less dependent on the underlying routing, but provides a more abstract view of the WSN. Structuring here can be defined via node properties, regions of the network or just by node neighborhoods. In section 5.2.2 we will show the development of such approaches related to the Scopes Framework.

### 5.2.1 Routing algorithms

Routing algorithms build a first layer of structure in a network. This structure is important for WSNs with a high number of sometimes randomly placed sensor nodes. In WSNs with several hundred sensor nodes it is not feasible to address every single node with a numeric address, the routing tables would easily exceed the storage capacity of the small nodes. Instead, often the data that is interesting for an application or regions of interest are used as unique but universal addressing scheme. A simple solution to reach all nodes of a network is classic message flooding, but with these numbers of nodes in the network and the scarce resources of a WSN this will lead almost instantly to a drastically reduced network lifetime. Therefore, approaches with, e.g., tree, mesh or other structures are used in WSNs.

There are numerous routing algorithms specifically designed for wireless sensor networks, so we can only introduce a small number of important representatives. First




(a) Interest processing (b) Creating gradients (c) Data transfer with reenforced path

**Figure 5.1:** Scheme of Directed Diffusion [51]

we show algorithms based on trees, meshes or other structures. We have chosen Directed Diffusion and Gradient-based routing as the routing algorithms used in the Scopes Framework were inspired by them. We then show examples of hierarchy- and location-based routing algorithms. These three categories can be seen as the major routing approaches used in WSNs, but of course they are not limited to these.

#### 5.2.1.1 Content-based Routing

A content-based routing scheme uses regular node addresses only to differentiate local neighbors of a node from each other. The global routing task is, for example, expressed in key/value pairs that describe the data that is needed for the current task. This description is then sent to the WSN and nodes that have data matching the description start sending the data back to the sink, the originator of the data description. Well known algorithms in this field are, for example SPIN [47], Directed Diffusion [51], gradient-based routing [114], Cougar [147] and ACQUIRE [111]. In the following we will discuss Directed Diffusion and gradient-based routing, as these two approaches highly influenced the design of our routing algorithms.

**Directed Diffusion**

Directed-Diffusion [51] is a case of content-based routing. It uses different elements: interests, data, gradients and reinforcements. The interest is a request for data that uses the aforementioned key/value pairs to specify the kind of data needed in the current task, but other naming schemes for data are also possible. The data description can include location information, frequency of data sampling, a timespan of interest and other properties. The interest is repeated in intervals and can reflect changes of the data description. Flooding an interest through a WSN, figure 5.1(a), creates gradients, pointing from the receiving node to the sending one. These gradients are used to find the best route from the data source back to the sink, see figure 5.1(b). To enforce the usage of a specific path it can be reinforced, e.g. to use a higher bandwidth. This can also be done with multiple paths to spread the load of a query over several nodes and

paths. Data in a WSN consists mostly of sampled sensor data related to a physical event. After the steps described above, the data source can send data to the sink, see figure 5.1(c). To cope with problems in the network like failing links or loops, a path can also be negatively reinforced. This way a former best path can be deselected and a new path can be reinforced. As the nodes in the network keep a list of neighbors they got an interest from, it is easy to choose a new path if the old one is blocked for some reason. But multiple links contain the risk of loops. These can also be avoided using negative reinforcements.

#### Gradient-based Routing

In [114] Schurgers et al. propose an extended version of directed diffusion where the interest and gradient interaction is changed. The interest includes the number of hops it has traveled. With this information each node in the network knows its distance to the sink. The difference between its own height in the network and the height of its neighboring node forms the gradient. This way the gradient is no abstract value but reflects the shortest path to the sink.

#### Discussion

We chose directed diffusion as base for our own routing implementation as it is closely related to the features we need in our framework. It supports sending of interests and building a routing tree while the nodes subscribe to an interest. But this routing tree is only capable of sending data from the sensor nodes to the base station. Therefore, we had to extend this feature in our own implementation to achieve a bidirectional communication, see [53]. Also we had to change the interest handling as we send our own messages from the scopes layer, see chapter 2.5.1, and do not want two messages of the same kind in the network.

### 5.2.1.2 Hierarchy-based Routing

Hierarchy-based routing algorithms are often based on clustering, see figure 5.2. This means that for each group of local nodes one is defined as cluster head. All other nodes of this group send data only to their cluster head. How the cluster head is chosen depends on the actual algorithm. The cluster heads can form their own subnetwork that sends data to the base station or there can be additional hierarchy levels. The graph of such a clustered WSN is a tree. Examples for these algorithms are LEACH [46], TEEN [78], energy aware routing for cluster-based WSNs [148], or [134,146] as further advances in energy-efficient clustering. Here different approaches are taken to reduce energy consumption or distribute the energy consumption more evenly over the WSN.

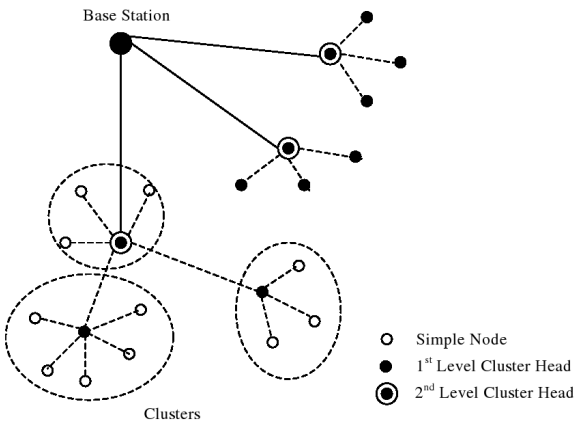



**Figure 5.2:** Schema of hierarchy-based routing [1]

### 5.2.1.3 Location-based Routing

The approaches discussed above build routing structures before they send any data. In location-based routing no fixed routing paths are used. The only assumption is that every node in the network knows its location with a specific precision. This can be achieved quite accurately with GPS-receivers [108] or less accurately with relative distances to other nodes gathered from the received signal strength indication (RSSI) values of received messages [145, 151]. Messages are then addressed to a region or a specific location, see figure 5.3. Depending on a node's location and the location of its neighboring nodes it is known where to send the message next. With this algorithms single nodes and groups are addressable. This approach saves the effort of establishing a routing tree or mesh, but has to spend extra energy to determine the location of a node.

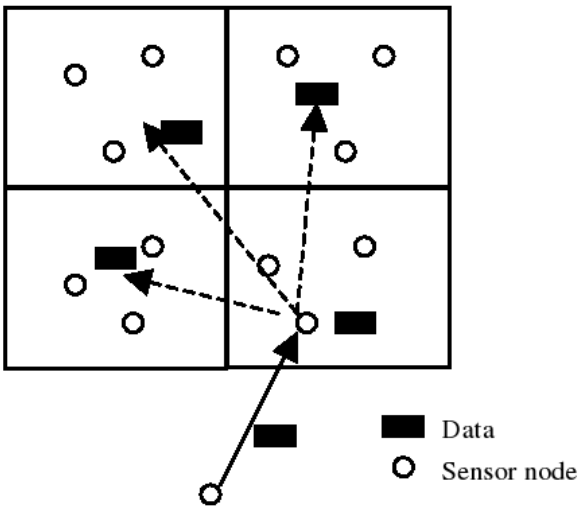



**Figure 5.3:** Schema of location-based routing [145]

### 5.2.2 Frameworks

A framework is an abstraction of generic functionality that can be accessed by a defined API. It is often configurable so that the framework can be reused in different scenarios. The configurability may be static at compile time or dynamic at runtime. WSN applications can be implemented with less effort as they can be written on an abstract level on top of the framework.

In this section approaches with framework characteristics for WSNs are discussed that do not include any system outside the WSN, like laptops, servers or the Internet. The following approaches are trying to solve similar problems as the Scopes Framework. They differ in aspects like a different systematic approach or are used in different areas of WSNs. We also want to show the advances over time in the area of grouping schemes in WSNs. Therefore, we start with some approaches that were presented at the time the work on scopes began and advance through time until the most recent developments.

#### 5.2.2.1 Hood

Hood [142] provides two abstractions that are commonly used in sensor networks. First, the notion of a node's neighborhood, and second, sharing of named variables and its data among neighbors. A key mechanism of Hood is *broadcast/filter*. It is used for both data sharing and neighborhood discovery. When data is shared it is always sent as broadcast. Nodes receiving data filter them to determine valuable data and what data may be cached. The receivers of data can decide what nodes to add to their neighbor list and what data to store. The owner of an attribute has no control over the storage behavior of its neighboring nodes.

Shared attributes define the elements of a node's state offered to its neighbors. These may be sensor readings or its location. After an update of an attribute value it is broadcast, other behaviors are possible depending on the push policy used. Is a co-neighbor receiving an updated attribute value, it is passed through all filters defined on the node. The filters then evaluate if the node is still valuable to be kept on the neighbor list and updates or removes its cached attributes. For a node in the neighbors list a so called *mirror* is instantiated where the node's observed status is stored. This comprises not only the reflected attributes, but also *scribbles*. These are local annotations about the neighbor node, like a quality estimate.

The definition of a neighborhood determines if a node is valuable enough to be added to the neighbor list. A node can define multiple neighborhoods and each of them have their own notion of a valuable neighbor and which attributes to cache. The decission if a node is valuable is taken depending on its shared attributes. This leads to potential asymmetric neighborhoods. Let's suppose node A's neighborhood definition accepts node B's shared attributes as valuable. This does not mean that node B also has a neighborhood where A's attributes would make A a neighbor of B. In this scenario B is called

a *neighbor* of A and A a *co-neighbor* of B, but not vice versa. So A is not a neighbor of B and B not a co-neighbor of A.

The definition of different neighborhoods with different attribute sets needs slightly different algorithms and data structures, which are difficult to parameterize. To enable this the authors defined a new command *generate*, so the code needed for the parameterization can be automatically generated.

**Discussion**

The biggest difference between the Scopes Framework and Hood is the restriction of the grouping scheme. This limits Hood to one-hop neighbors while Scopes provide a grouping scheme that can use multi-hop connections over the whole WSN. Hood is a very early approach and is limited in its functionality. Nonetheless, it enables different neighborhoods on one node and thus neighbors and data for different operations. This is also a feature of the Scopes Framework represented by the ability to define multiple scopes.

#### 5.2.2.2 Abstract Regions

The Abstract Regions programming model [137, 138] is based on the model originally proposed by Hood. Abstract Regions is able to create a neighborhood relationship between a node and others in the same region. This makes neighborhoods (or *regions*) like "the set of nodes in $k$ hops" or "the set of nodes with maximum distance $d$" possible. The major goal of Abstract Regions is to provide programmers with a programming abstraction that hides the complexity of underlying mechanisms, but still be able to optimize the application, and to perform local spatial operations in the sensor network by sharing data and coordinate activities between nodes in the neighborhood. For this purpose it provides a set of operators: neighborhood discovery, enumeration, data sharing and reduction.

Before performing any other operation on a region a neighbor discovery is performed. This process is initiated on every node and is a continuos activity, where every node is informed about changes in the membership set, due to joining, leaving or moving nodes. The process can be terminated by a node at any time to prevent further neighborhood discovery messages. In this case the operation returns a quality metric, for example the percentage of responses of candidate nodes. The enumeration operator returns the list of nodes participating in a region to be able to directly interact with them. With the data sharing operator it is possible to define name/value pairs to be stored and shared within the nodes of a region. Therefore, two functions are supported: *get* and *put*. With these functions, variables can be stored and queried from the local cache and also from nodes in the same region. To reduce a shared variable the reduction operator takes a shared variable key and an associative operator, like *min*, *max* or *sum*, gathers the shared values in the region and makes the result also available in a shared variable.

The authors implemented Abstract Regions on top of different routing paradigms, like location-based routing, a planar mesh or a spanning tree.

As one special feature Abstract Regions provides a *quality measure* to the application. This measure specifies the completeness or accuracy of an operation. Applications can use this feedback through a *tuning interface* to set low-level parameters for the Abstract Regions implementation. The application designer has to study the impact of parameters on application performance, as the parameters and their influence depend on the Abstract Regions implementation and the application behavior.

#### Discussion

Abstract Regions is still a very basic approach, but it is much closer to the Scopes Framework than its ancestor Hood as it supports grouping over nodes in a region. It also provides a static neighborhood definition. However, abstract regions are limited to variable sharing while the Scopes Framework is much more generic and allows even tasks to be shared.

### 5.2.2.3 Generic Role Assignments

Römer et. al created with Generic Role Assignments [110] a *role-based* approach to program sensor networks. The general assumption of the authors is that when programming a node not all parameters are known in advance, so it is hard to decide which node should get the code for which role. This means a modus for *in-situ* configuration of roles has to be found. For this purpose four core elements are defined and every node must support Generic Role Assignment.

The first element is the *property directory*, here the properties of an individual node are stored. This ranges from properties like the available sensors and other hardware features, to static or dynamic properties that can be assigned on an application basis. There is a directory on every node and the different directories are independent of each other. The next element is the *role specification*, it is created by an identifier and optional properties to further refine a role. A set of rules defines the conditions for the assignment of roles. The decision which role is assigned to a single sensor node is typically based on properties of a set of nodes. It is assumed, that all sensor nodes have the same set of roles and same related rules. The *role assignment algorithm* assigns roles to a sensor node, taking role specifications and node properties into account. Depending on the application it is possible to assign multiple roles to one node. Because of the harsh energy restrictions this assignment algorithm is a distributed and localized one that only requires a limited amount of communication within a node's neighborhood. The last element are *basic services*. This stands for the additional functionality that is needed by the application and for role assignment, like localization, time synchronization and so forth.

**Discussion**

This approach is very interesting, as its focus is on the different roles that are present in a WSN. Generally there will be, e.g., a base station, sensing nodes and nodes that are just forwarding data between the other two. So, when deciding at run-time which node gets which role this is a kind of grouping into the different roles. The grouping in this approach is not based on communication or network location but on the roles the nodes chose. This has obviously influence on the network and how to communicate. With this grouping Generic Role Assignments is orthogonal to the Scopes Framework as the network is not structured in the sense of a communication group based on node properties, but by specific functions carried out on the nodes.

#### 5.2.2.4 Region streams

In [91] Newton et al. present *regiment*, a functional macroprograming language for sensor networks, which is based on the *region streams* data model. Macroprograming is here understood as an approach to program a sensor network as a whole and not every sensor node separately. This means that from the high level macroprogram the local behavior for the individual node is decomposed automatically by the regiment compiler.

Region streams represent spatially distributed, time-varying collections of node states. With some geographic, logical or topological relation the interest in a specific group of nodes is expressed. The related region stream represents the sensor data of the included sensor nodes. E.g., one could specify a relation like "$k$-hops from an anchor node $n$". Operations supported on a region stream include *fold* and *map*. Fold represents an aggregation function that fetches sensor values all over the region and aggregates them at an anchor node. Map applies a function to all values of a sensor node in a specified region. Map does not require additional communication, in contrast to fold, where all sensor data has to be gathered at one physical location.

The goal of regiment is to write a complex sensor network application in only a few lines of code. To do this, the regiments compiler transforms the global network-wide macroprogram to a nodal program based on a token machine. A token machine is a simple distributed machine that acts upon arriving named tokens, received via network or generated locally. The tokens initiate local sensing and computation. Depending on its type the related token handler is executed. Token handlers themselves can create new tokens or call other local handlers. These token machines are a simple mechanism of local execution, remote function calls and data storage.

**Discussion**

Region streams are an approach to specify a location or region an application is interested in. To get the sensor data of the nodes located in such a region it is transferred to the origin of the region stream. Transferred data can be pre-processed by using a macroprogramming language. These region steams are statically defined and are distributed to the local set of WSN nodes with a special program-image for every single node. The

basic idea of region streams is similar to the one of the Scopes Framework. It defines the sensor nodes of interest, but this is done statically and the WSN application has no influence at run-time. Also the retrieval of sensor data is one of the core functionalities, but retrieval only, whereas Scopes is built for bi-directional transfers.

#### 5.2.2.5 Logical Neighborhoods

With Logical Neighborhoods [86] Mottola and Picco propose an approach that also defines a notion of scoping. Logical Neighborhoods are quite high level for the application programer and consist of three parts: the Spidey language, the logical neighborhoods abstraction and the underlying routing algorithm. These components are tightly integrated with each other.

The Spidey language abstracts two concepts: logical *nodes* and *neighborhoods*. For the logical node you first specify a template to define what properties, sensor data, etc. can be gathered. And second you instantiate the template and connect the defined properties in the template with the calls and constants used in the code to retrieve these values. The specified attributes in the template can be static or dynamic, but this has to be decided when creating the template. A logical neighborhood is defined the same way as a node. The template defines which attributes a node has to match to be a member of the logical neighborhood. The neighborhood can be parameterized. For example, in the instantiation of a neighborhood, a threshold can be adjusted to the needs of the application. The logical neighborhood is then composed by the nodes in the network that match the specified criteria. With the instantiation of the logical neighborhood a maximum hop distance for the neighborhood can be defined and also a number of credits that can be spent by the cost function of the routing algorithm.

The authors show a very specialized routing algorithm that is structure-less and built upon a notion of local search. Nodes advertise their profile, which is the list of their name/value pairs specified by their template. In doing so the nodes build a distributed state space containing information about the cost of reaching a node with given data. Messages sent to a neighborhood contain its template, which determines the parts considered for matching. So, messages navigate towards members of a neighborhood by following paths along which the cost associated with a given neighborhood template is decreasing.

**Discussion**

In Logical Neighborhoods the specification for a logical node and the neighborhood definitions are defined at compile time. This means that no new attributes can be defined at runtime and the neighborhood cannot be changed. As a state space is distributed in this approach only a small set of attributes is feasible. The more attributes are exchanged the more overhead for routing is introduced. The overhead results from the way the routing works, as for every attribute a new routing request is generated. The tight integration between the different components of the system make it hard to adapt

Logical Neighborhoods to other scenarios, e.g. only very specialized routing algorithms can be used, even if it is in principle possible to exchange them. Another difference to the Scopes Framework is that scopes not only allows grouping of nodes, but also creating a hierarchy of groups that can be nested. Although Scopes and Logical Neighborhoods appear to be similar from an abstract point of view, many differences exist when looking at the details.

#### 5.2.2.6 TinyDB

In TinyDB [76] the sensor network is seen as a database where you can query data from it with a SQL-like language. There are also other approaches like [11], a predecessor of TinyDB. TinyDB is a distributed query processor that runs on each node in the sensor network. A query is parsed, optimized and sent to the sensor network from the base station and then disseminated and processed in the nodes on the network. The results are flowing back up the routing tree that was built as the query was propagated, and possibly get aggregated. As these queries are executed thru a period of time and a stream of data is generated for it, time is divided into *epochs*, where for each sensor participating in a query a value is generated and sent per epoch. Each epoch takes the same amount of time.

The query language proposed in TinyDB is a subset of SQL where the whole network is seen as one table with every node and sensor adding a new entry. The semantics of `SELECT`, `FROM`, `WHERE` and `GROUP BY` are the same as in SQL. Some extensions were added like a `SAMPLE PERIOD` clause to be able to specify the sampling period for queried sensors or a `LIFETIME` clause to set a lifetime for the sensor network. TinyDB then estimates sensor sample rates and transmission rates with the given energy to meet the lifetime goal. Further additions are `ON EVENT` queries and `STORAGE POINTS` that are related to materialized views in common databases.

TinyDB uses *semantic routing trees* (SRT) to disseminate the queries and collect all the data. SRTs are designed to allow each node to locally decide if any node below it will need to participate in a given query over a constant attribute. To do so, an SRT is an index over the attribute values stored in an unidimensional interval representing the range of attribute values beneath each of its children. With this information the decision to further disseminate a query or not is possible. If the attribute of the local node match the query, it is executed. The creation of an SRT is done in a two step process: First the *SRT build request* is flooded through the network. It contains information about the attribute the tree is used for. After forwarding the request, a node waits for answers from its children. If a node has no child it sends a *parent selection message* to its chosen parent including the attribute value of it. When a node receives a parent selection message and the given attribute value changes the stored value range, this node also forwards the change. The tree is further maintained to cope with joining and leaving nodes.

The execution of a query is done with always the same steps for each epoch: Nodes sleep mostly to save energy during epochs. To sleep as long as possible but still to be

able to aggregate values each epoch is slotted in multiple slices and due to time synchronization each node knows when to awake and receive messages from their children. In the awake phase the sensors are also sampled and if needed the values are aggregated. As last step a received message is forwarded or a newly created one is sent. This scheme may change a bit for special demands, e.g., with `ON EVENT` queries.

**Discussion**

TinyDB has a vision of a WSN that each node in the network is a sensing device. And the data of these sensing devices can be transformed into a huge database table. Depending on the query to this table it is possible to query just a subset of nodes, a group. But, even when TinyDB supports actuation, this support is limited to a small and predefined set of methods. In [57] we have shown, that the Scopes Framework can be used to create a very similar WSN application called TikiDB, that also uses a database view on the WSN. In contrast to Scopes, TinyDB cannot be used in scenarios where the scenario logic cannot be fitted to a database approach. Nevertheless, for data retrieval scenarios TinyDB is a good candidate because of the different operations it supports.

#### 5.2.2.7 TinyCubus

The overall goal of TinyCubus [79,80] is a generic reconfigurable framework for wireless sensor networks. It is composed of three components. The Tiny Data Management Framework, Tiny Cross-Layer Framework and Tiny Configuration Engine.

The *Tiny Data Management Framework* provides several system and data management components, like time synchronization or aggregation. Each component has several implementations that are classified in three areas, the dimensions of a cube. First, these are optimization parameters, such as energy consumption; second, application requirements, such as reliability; and third, system parameters, like mobility. Depending on these parameters the Data Management Framework decides which component implementations are the best for the current situation. The evaluation of the best suited choice has to be executed over the whole lifetime of the sensor node.

To be able to retrieve information from other layers of the architecture *Tiny Cross-Layer Framework* provides callbacks and a state repository. Using the state repository the cross-layer framework acts as a mediator between the different components. This way cross-layer data is not accessed directly, but stored in the repository. Components using the cross-layer data are not depending on the implementation of the component providing the data, given a new implementation offers compatible data. To be able to retrieve data from the repository, it has to be known what data is available. For this purpose the authors provide a specification language with which each component can express what kind of data it provides and what data it needs.

In case new components have to be installed or swapping of some functionality is needed the *Tiny Configuration Engine* provides a way to distribute and install new code in the network. It supports the configuration of both system and application components

with assistance of the topology manager. It is responsible for the self-configuration of the network and assigns roles to each node with the help of a role assignment algorithm.

**Discussion**

TinyCubus is not a strucutring approach in the sense of grouping different parts of the network, but it offers structured access to the network and this is where the Scopes Framework and TinyCubus are comparable. Both approaches are designed for application or system module updates while deployed in the field. While Scopes is a communication framework and leaves the data handling to the applications, TinyCubus also provides a datastore where system and application modules can retrieve data in a controlled way. Another difference is the use of roles in TinyCubus, where Scopes just use states to reflect different behavior in the network.

#### 5.2.2.8 Melete

Melete's [152] main goal is to support concurrent applications in a wireless sensor network. Therefore, it extends the Maté [68] virtual machine to support concurrent applications and adds a dynamic grouping technique and code distribution support for spatially uneven and temporally changing deployments. With the extension of the virtual machine to support multiple concurrent applications the authors also introduce protection for the application's execution space. This is achieved by preventing variable sharing between different applications. On the positive side this makes the applications independent of each other, so rebooting one application does not influence others, but it also prevents information sharing between them.

The grouping mechanism is, in the first place responsible for connecting nodes that have to execute one particular application. A node can be member of multiple groups and each of them is connected to an application, referred to as associated groups. The memberships of multiple groups may overlap. A sensor node can also dynamically join or leave a group, as needed by the application or user. The deployment via a group is done in two steps, first grouping the nodes for the application and secondly disseminating code onto the group members.

The code dissemination is done with an altered trickle algorithm [69]. The authors added the so called passive code dissemination policy with active advertisements. Version information of all groups and so all applications is disseminated via advertisements throughout the network and maintained at each sensor node. Applications are only passively transferred on request from a node. In this modified trickle algorithm versions are forwarded right after a node received it and not after transmission of the application code.

**Discussion**

First of all Melete uses a virtual machine (VM) as basis, so the applications for the system are limited to the functionality of the VM. Of course, it has the advantage of isolated

concurrent applications. The Scopes Framework is a native program and applications using scopes are also native and can use the whole functionality the hardware or operating system provides. The grouping used in Melete is directly connected to applications, this means in reverse that every application has its own group. In Scopes the grouping scheme is much more universal. Of course, it can also be used as an application and distribution group like in Melete. But if an application needs an additional group for some special functionality, e.g., an actuation group, the application is free to create as many groups as it needs for its purposes.

#### 5.2.2.9 MQTT for Sensor Networks

MQTT for sensor networks (MQTT-SN) [122] is a classic publish/subscribe protocol adapted to the WSN environment. Data is published to a topic-id towards a MQTT broker. This broker is outside the WSN, but may be talking directly to the WSN via MQTT-SN or via a gateway with MQTT. Therefore, MQTT-SN resembles an advanced connection management with a gateway. It brings no routing with it, but can sit on top of, e.g., 6LoWPAN [118].

As in standard pub/sub-systems the subscriber tells the broker which data topics it subscribes to. This can also be a node in the sensor network. Sensor nodes publish data that they produce to a topic on the broker. To do so, the nodes have to connect themselves to the broker by first finding a suitable gateway. This is done via advertisements from the gateway and search messages from a client. After that they can register for a topic and publish data. All these functions are quite compressed and often ids are used to keep network messages small to cope with the constraints in WSNs.

**Discussion**

MQTT-SN builds groups, like the Scopes Framework, via the topics. But these groups are built solely by the sensing clients, that register for a topic. The subscribers have no influence on them. In contrast in the Scopes Framework the subscriber is the one issuing the definition for a topic/scope and can this way select the specific data and node properties it needs. As the broker is outside the WSN, the energy intensive managing and messaging towards the subscribers is done outside the WSN. In this system it is assumed that the WSN is only used to publish data, subscribing to topics is possible but expensive, as the data traverses out of the network and than back.

MQTT-SN includes only one security measure. That is the authentication while connecting to the broker. If there is further security needed the underlying routing has to provide this. Whereas SecScopes provide integrated and extensive security measures.

#### 5.2.2.10 RushNet

RushNet [70] is an interesting, but somewhat brutal approach. It aims at traffic prioritization in WSNs with a low priority (LP) and a high priority (HP). The LP transmissions

are straight forward. They use a low power CSMA (carrier sense, multiple access) transmission and a token-based routing algorithm. This means that a token is passed to every node and only the node with the token is allowed to transmit. This way there should be no collisions or congestion for LP transmissions on their own. Now, the HP transmissions use instant sending and forwarding with a high transmission power. This means the low power LP transmission is pushed aside by the HP transmission, using the capture effect.

To use this effect reliably the authors proposed the so called preemptive packet train. This means that an HP message gets sent several times in a row, the authors propose 4 times. The first time may collide with an ongoing transmission, depending on the length of the transmission the second one may also be corrupt, but after this the CSMA LP transmission will be stopped and the following retransmissions may be received on the parent node. After this the parent node also instantly retransmits the HP message to its parent, and so on. Is the message sent far enough the normal LP transmission can be restarted with a retransmission of the failed message from the beginning.

The authors also showed evaluations on coexistence with other 802.15.4 and 802.11 networks and the capture effect mechanism works also in these combinations.

#### Discussion

Obviously the RushNet does not use a very kind behavior. And we doubt that two networks of this kind can coexist with each other without interference. This mechanism only works on the kindness of others. RushNet is not structuring the network in different parts, as Scopes do it, but it structures its data with its LP, HP approach. Looking at the characterization of the LP and HP traffic in this paper we see some similarities to our evaluation of Scopes: The LP traffic is characterized as continuous bulk transfer and the HP traffic as single peaks. These are the same traffic patterns that we used for our evaluation. But we evaluated the traffic patterns independent of each other and in this approach they are evaluated simultaneously.

## 5.3 Mixed Mode Systems

Compared to the WSN frameworks of the last section, Mixed Mode Systems support a much wider range of computing devices, while WSN frameworks are focused on sensor nodes as hardware platform. The worst case situation for a WSN may be when it has to cope with a heterogeneous WSN. The different devices in such a WSN are connected via a common network interface, mostly an IEEE 802.15.4 radio. So, all the platforms are more or less scarce in regard of the computing capabilities and the available RAM or ROM space.

Mixed Mode Systems comprise much more diverse computing devices. They can include small sensor nodes, powerful personal computers and also large server-like systems. But not all of them have to be present in one system. As the range of systems can be huge, also the available resources are highly varying. This makes it obvious that not all functionality may be available on all devices of the system. Most of the Mixed Mode

Systems presented here have their roots in ether the PC-world or the WSN-world. To be able to span the missing parts each system has to be scaled up or down depending on their origin.

The scaling up of a former WSN approach brings fewer problems, as the resources available are much larger than on a small sensor node. Scaling a system down to a WSN node introduces several problems. The most obvious is the scarcity of resources on a sensor node. This necessitates a reduction of functionality and the optimization of data exchange formats, e.g., the full spectrum of functionality is basically not fitting into a 24kB flash-ROM, or the XML data exchange format is simply too expensive for the nodes with limited energy and bandwidth. Infrastructural differences may also result in architectural changes. In WSNs mostly distributed algorithms are deployed compared to many client/server architectures on the PC side. Also the change to MCUs as platform may result in a new implementation, when the PC-side software was written, e.g., in Java. A JVM is normally not available on these small scale systems. Additionally, sensor nodes and PCs generally do not use the same communication medium, a wireless medium is used for WSNs (IEEE 802.15.4) and different wireless or wired networks are used for the PC-type devices, so the complexity to communicate with each other is also increasing.

In the following we will see different approaches of these Mixed Mode Systems. Both scaled up and scaled down frameworks, e.g. 5.3.2 and 5.3.4, but also some developed specifically for the mixed nature of these systems discussed in 5.3.3.

### 5.3.1 RUNES

The RUNES Middleware [21, 22] focuses on heterogeneity, scarce ressources and dynamism.

It is based on a kernel that is implemented on multiple systems. All kernel implementations, namely one for Contiki, C and Java, share the same component model, but with slight differences depending on the capabilities of the underlying system. This provides a unified programming model over a wide variety of devices. To communicate the sensor network can send messages through a gateway to other networked hardware, e.g. a PC. The gateway uses the C implementation of the kernel, where PC's can use the Java version. On each device different components can run to fulfill the application's tasks. These components can also be dynamically loaded and unloaded at runtime.

The authors have implemented a Publish/Subscribe component for the scenario presented in [22]. This component is in charge of the following two tasks: creating and maintaining a tree-shaped overlay, and setting up message routes on top of the overlay, and reconfigure them in case of topology changes.

#### Discussion

RUNES has a focus not just on heterogeneity, but also on reconfigurability of the system. Therefore, they introduced a consistent Kernel API that is the same over all the

different platforms and implementations. So called components, similar to tasks in the Scopes Framework, can be created very easily and deployed throughout the system. But RUNES does not provide any grouping mechanisms for the network, it uses a pub/sub approach where the network is connected by a tree-like structure.

### 5.3.2 COSMIC

The COSMIC Middleware [48, 60] was originally designed for fieldbus systems, like the CAN bus. It uses an event based Publish/Subscribe approach, and enables content based addressing of event data. Events are created in two ways. Either a sensor reading triggers a new event, or a periodic timer can do this. An event is composed of three parts. A unique identifier of the content that is globally unique for all subnets used; the data itself, current state information of the node or sensor readings; and additional attributes, e.g. sensor position or quality of data.

COSMIC provides event channels for event transmission in an unidirectional way. The definition includes the unique id of an event, attributes for data dissemination and handlers for an exception or message arrival, the latter is only available to subscribers. Event channels can provide different qualities, like hard, soft or no realtime. The middleware can use this information to reserve resources to meet the requirements. The resource reservation is done transparent for the application. Heterogeneity of the underlying network is also hidden by the middleware.

Different subnetworks are connected with a gateway. COSMIC supports CAN, TCP/IP and ZigBee networks and is available for 8- / 16-bit MCUs, Linux and Windows.

#### Discussion

COSMIC is a straightforward approach. It provides classic pub/sub functionality to establish so called event channels. These channels are used to forward events through the network. Because of the different network types supported, each of them creates its own subnetwork and events that do not have to leave their current network are not forwarded to the other subnetworks. In this sense there is a coarse grouping of devices. Another interesting point is that the event channels can be used to reserve bandwidth in the network for transmitting the events of the channel. So COSMIC is one of the few approaches doing some kind of traffic shaping.

### 5.3.3 PhysicalNet

The idea of PhysicalNet [131, 132] is to enable applications in dynamic environments, where heterogeneous resources from different users coexist. Devices can be mobile inside and between networks. Multiple applications are concurrently running on the same resources, where users can safely share their own resources.

To achieve this, PhysicalNet is based on a centralized design that was developed as a four-tier service oriented architecture. On the first and lowest tier *service provider* pro-

cesses are located, these enable the utilization of the different sensors and actuators of a device. The second tier contains *gateway* processes that are responsible for media conversion, e.g. ZigBee to IP or vice versa, and connects a local device with its negotiator. On sensor network nodes only the service provider tier can be used because of computation constraints. The second tier can be combined with the first one used on PDAs or more powerful devices that provide sensors or actuators and have multiple communication devices. The third tier contains *negotiators*. They are responsible for tracking the registered devices, handle access control and administer the shared resources. The fourth and last tier contains *application* processes that create or cancel requirements for remote sensors and actuators. Multiple applications can run on one negotiator, same as one application can use multiple negotiators to access the addressed sensors.

The *negotiator* is the core component of PhysicalNet. There can be multiple negotiators, each is creating its own administrative domain where it exclusively is responsible for managing resources the registered devices are providing and the applications and their demands in the sense of access restrictions and resources. Therefore, the demands from applications are checked against an access rights specification, but still conflicts can occur when more than one application tries to access the same resource. To handle this, PhysicalNet provides conflict resolution modules.

Besides the negotiator as core component, *Bundles* are the core programming abstraction used in PhysicalNet. It includes two parts, the definition of a group of services and a specification of the functionality of these services. The definition of a group can be arbitrarily complex, as long as it can be expressed in Java code. The membership evaluation is done dynamically, so nodes that do not match the specs at a given point in time, but did before, are removed from the bundle. And bundles can span across different administrative domains, same as applications can require services from different domains. The evaluation of bundles is done on the negotiator, as it knows the requirements of the applications and as services report their state in regular intervals, all needed information is easily available.

#### Discussion

PhysicalNet is a higher level approach that is based on the internet and central servers. It has the notion of administrative domains and can utilize multiple networks of sensor nodes, smart devices or laptops. It has a similar approach with its Bundle abstraction as the Scopes Framework. Bundles are used to select specific devices to gather data from and Bundles can also be created with a set of parameters, like a scope. The evaluation of the parameters and the grouping of the network is then done on a centralized server. The Scopes Framework can make these decisions in-network and on the single node. So, a WSN in PhysicalNet is used just like a data source as no decision taking is done on the nodes. PhysicalNet has the group-building strategy in common with the Scopes Framework, but as it operates from a server-size machine outside the WSN their targeted environment is very different.

### 5.3.4 PEIS-Ecology

The concept of a PEIS-Ecology [112] includes robotic technologies in a smart environment and was extended to also include wireless sensor networks [12]. The basic principles are a uniform notion of a PEIS (Physically Embedded Intelligent System), a uniform communication model for communication between the devices that copes with the dynamic joining and leaving of them, and a uniform cooperation model, which is provided by a publish/subscribe mechanism. The PEIS-Ecology is built on the PEIS-kernel and the PEIS-middleware it is part of.

The PEIS-Ecology middleware consists of the PEIS-kernel and some additional components. The kernel provides a distributed tuple-space where every PEIS in the network can read from and write information to. This information is accompanied by meta data, e.g. a timestamp or the creator. Under the tuple-space a peer-to-peer (P2P) network is maintained among all detected PEIS. The network allows every PEIS to access tuples from other PEIS, even if there is no direct route between the two and it is updated dynamically to allow leaving and joining the ecology. The additional components of the middleware are performing advanced meta-level action on the ecology as a whole, like planning, self-configuration and monitoring.

To support WSN type sensor nodes the PEIS kernel was down sized, this version is known as *tiny* PEIS kernel and has in principle the same functionality. The most changes were done on the sizes of fields, identifiers and timers, but also on shrinking the network messages to a size IEEE 802.15.4 can handle. This change and the switching of network media made it necessary to implement a bridge-like component called tiny bridge to convert messages back and forth.

#### Discussion

PEIS was developed to interoperate with robots via a network connection. It was then scaled down to match the capability of sensor nodes. The data that is exchanged are events that are composed of key/value pairs. The PEIS-Ecology does not provide any notion of a grouping scheme or applications, it allows to exchange basic data. And this way it has nothing directly in common with the Scopes Framework.

### 5.3.5 RoboFrame

RoboFrame [101] is a framework to control a single or a group of autonomous lightweight robots. It is written in ANSI C++ and targets platforms running Windows or Linux. It is based on two extendable components, RoboApp and RoboGui. One as base for any higher level functionality, the other for graphical user interfaces. Some other tools and packages are available, like the behavior specification language XABSL [74] or modules for laser-range finders and other hardware.

*RoboApp* uses a platform abstraction layer to handle system specific calls, threading, synchronization and so on. An application that extends RoboApp introduces modules to execute different tasks. These modules are independent of each other, but define their demands and what they provide in a descriptive manner. Messages between the modules can be exchanged directly with defined types, that have to have the ability to serialize and deserialize themselves, and a blackboard approach for large data structures is available. Data providers and receivers have unique identifiers, so the router process can dynamically route the messages between modules not only on the local robot. The framework allows to extend the router, so that messages can be exchanged, e.g., via a network socket. The execution of a module can be done event-based or in a sequential way.

*RoboGui* uses the Qt toolkit to provide an easy way for debugging, monitoring and displaying. It uses the same messaging subsystem and so can easily access all information available in RoboApp.

**Discussion**

RoboFrame is very successful in the robotics area, but it is more focused on high end equipment and robots compared to a sensor node. As mentioned above it runs on a Windows or Linux machine and its focus is on the robot and its control. It has the ability to communicate with other robots but this is used as a point to point communication and not for communication with groups or perform grouping in a network.

#### 5.3.5.1 Constrained Application Protocol

The Constrained Application Protocol (CoAP) [13, 119] was designed with WSNs as target platform in mind. CoAP provides a REST-like interface and duplicates features of HTTP in WSNs with respect to the constraints in this environment. It is designed for easy conversion to HTTP and thereby an easy connection to the Internet. This brings URIs and HTTP's GET, PUT, POST and DELETE commands to resource constrained networks. CoAP builds on top of IPv6 (6LoWPAN [118]), but uses UDP in contrast to HTTP's TCP connections. UDP matches better as TCP with it's low overhead for transmitting data.

Besides this, there are further options to cope with the peculiarities of sensor networks. We will show only two of them in this summary: Block and Observer. The Block-option allows block transfers for large amounts of data, like firmware updates. Regular CoAP messages are suitable for small message sizes, but UDP limits these to a maximum of 64 kB. The Observer-option optimizes the client-initiated transfer pattern known from HTTP. If an HTTP-client wants to keep up to date information about the status of a resource it has to constantly poll the resource. With the Observer-option the client has to specify only one request and gets constant updates on changes of the requested resource, if it is modified. This results not only in the client always being up to date on that particular resource, but also in savings of network bandwidth since update messages are

only sent when a change occurred. Security is also an option in CoAP, here a lightweight version of DTLS is used.

Besides the CoAP specifications there is also potential for optimization in the layers below CoAP, as shown in [65]. And there are further refinements, e.g. for group communication [106].

**Discussion**

CoAP is a widely adopted approach and highly optimized for the WSN environment, but with all the features above it also adopts the point-to-point philosophy from HTTP. This means, for example, with the introduction of URIs it gets easy to address a resource in the network, but also you address mostly single resources. Still, CoAP is able to use IP multicast, but only with a restricted feature set, e.g. no security. The Scopes Framework always addresses groups or types of resources. In this way standard CoAP and Scopes are designed on orthogonal principles.

But there are efforts to enhance group communication in CoAP, until now this is an experimental RFC created after the Scopes Framework.

### 5.3.6 DepSys

DepSys [89] is an application level system that aims at resolving dependencies or collisions when using Smart Home network actors or sensors. DepSys defines apps for the different tasks in a smart home and also an app-store, like those available in the mobile phone domain. These apps have meta-data attached to the firmware itself that describes, for example, what sensors/actors are used, and how they are used. The meta-data is then used to check for different dependencies, like requirements on resources, control dependencies for sensors and actors, or missing dependencies. The dependencies are than checked at installation time of the app and at execution time.

The meta data contains besides regular parameters, like types of sensors/actors and usage frequency, data on *effect, emphasis and condition*. These three parameters are used to define the effect of an app on its environment, the emphasis describes the importance of a control operation, and condition allows to define the conditions when the app is active, e.g. in the morning or at sunset.

**Discussion**

Even though DepSys has been developed for a specific domain, the smart home, the problems mentioned arise in every multi-application sensor/actor network, wired or not. The Scopes Framework manages the life-cycle of in-network tasks, but does not check the actions that the different tasks try to execute on actors or sensors. Therefore, the Scopes Framework and DepSys complement each other, one enabling the access to the underlying network and the other by checking the dependencies of WSN applications.

## 5.4 Security in Computer Networks

Security is a hot topic in computer networks. We will concentrate on approaches from areas such as publish/subscribe and mobile networks. Approaches proposed for these domains must cope with similar security problems and utilize similar security algorithms as our SecScopes Framework.

### 5.4.1 Capability Based Access Control

In [99, 100] Pesonen et al. propose a multi-domain publish/subscribe system with capability based access control and event data encryption. In this decentralized pub/-sub system a broker network of several broker nodes provides an event service which is responsible for publishing events, managing subscriptions and routing of events. A domain is composed of a broker network, clients and an access control service. The access control service grants access rights to broker nodes and clients according to a domain internal policy. To form the shared multi domain system, one domain is the so called coordinating domain. It coordinates the formation of the shared pub/sub system and invites other domains to join. It can be seen as the owner of the shared system who manages access to it. The main idea is to decentralize access control decision making and policy and credential management over all participants of the network. Therefore a simple public-key infrastructure is used and it results in a scalable system. The certificates are used to authorize a subject for specific actions. These actions are: Connection to the broker network; introducing new event types to the network; extending existing event types; and access to the pub/sub API, that consists of publishing and subscribing to events of a given event type. To be able to restrict the access to events, secure event types are introduced. These are events of predefined types that provide integrity and authenticity of type via digital signatures. They also have attributes that may contain confidential data. Therefore, each attribute is encrypted with its own key. This way only broker or clients with access rights for a specific attribute are able to decrypt. As for the encryption of attributes symmetric keys are used. Clients do not access keys directly, the access rights are enforced by the brokers according to their own access rights. Events are transferred to clients via an encrypted connection and attributes for which the clients have no access rights are set to a default value by the local broker. The clients have to connect to a broker with sufficient access rights for their published or subscribed events.

#### Discussion

This approach is similar to the Scopes Framework. In Scopes we also distribute the access control decision making to the network, but we cannot limit this mechanism to a set of trusted nodes, as for a sensor network every node is a potential publisher, subscriber and/or broker node. Additionally introducing even a simple public-key infrastructure results in much higher load for network and computational resources. Especially the small

network bandwidth and the lossy medium are huge bottlenecks in WSNs for central infrastructures. In contrast to WSNs the computer systems assumed here are server-sized, this means that there are no resource problems to be expected from the hardware or the network bandwidth.

### 5.4.2 On-Demand Multicast Groups

Yu et al. use a network of PC-sized nodes to form multicast groups with membership anonymity [149]. To achieve anonymity they choose CP-ABE with a replacement access structure. It uses unique IDs for each node. From these IDs single bits are used to encode attributes. First, an ID is assigned to all nodes in the network, after that the group controller (GC) can establish a new multicast network by creating a group with a specific set of IDs, which relates to a set of attributes. To achieve anonymity the group key is encrypted via the CP-ABE algorithm and distributed to all possible group members (GM). If and only if a node id matches the group definition the GM can access the group key to transceive messages. But the GM is not able to determine which attributes gave him access. This is one key point to member anonymity, as only the GC knows the definition of the group and thus no further information about the kind of group members is known to others. Also the number of members of a group is kept anonymous.

**Discussion**

The general design is closely related to the Scopes Framework with the attribute-based dynamic grouping of network nodes. But the way that attributes are implemented would not work for a scope, as in the Scopes Framework a large number of attributes are assumed due to the shared character of our design and its usage patterns. Here the number of attributes is limited by the size of the id and it is not possible to enlarge this number after deployment, unless every id in the network is replaced by a new, larger one. Also the usage in a shared multi-domain scenario is restricted by the centralized management of the attribute-space. In contrast our SecScope approach uses a hash value of the attribute's name to represent the attribute and the name can be chosen by a party without interfering with other users of the network.

### 5.4.3 MundoMessage

The MundoMessage middleware [136] builds on top of the open TETRA standard, which provides a communication system with end-to-end encryption. The middleware supports one-to-many communication in an emergency scenario, and provides a technique to re-identify selected attributes of pseudonymous receivers and expressive logical policies to address them.

MundoMessage uses an adapted CP-ABE algorithm to establish attribute-based messaging (ABM). The attributes are used to address specific groups or single entities of emergency response organizations that do not have to be known in advance, but can be

identified by their role. The algorithm was extended by a location component, so that a message may be decrypted only by personnel working in the specified area. With the re-identification technique it is possible to, e.g., look for a specialist in a geographic area. This is possible as location updates are sent regularly by each user of the system to the headquarter. These updates are tagged by a pseudonym in a way that it is not possible for unauthorized personnel to retrieve the identity. The pseudonym is calculated from an unique ID with an associated set of properties, like the area of a specialization. With the proposed mechanism one or some of the associated properties can be retrieved and it can be checked, e.g. in an emergency scenario, if a required specialist maybe already at the incident site or if one has to be requested by the headquarter.

**Discussion**

MundoMessage is proposed for a similar, but still resource richer scenario than Scopes in WSNs. The here mentioned TETRA network is also a distributed network using wireless technology, but it is more related to a mobile phone network than a WSN. Also the resources of the devices used in the network are on the scale of mobile phones. Beside the scenario the approaches have the usage of the CP-ABE algorithm in common, but the provided functionalities are different. Where for the Scopes Framework the attribute-based grouping of unattended nodes is the core functionality, for MundoMessage the anonymous, location-based addressing of groups or personnel for voice transmission and the re-identification are the core functionalities.

## 5.5 Security in Wireless Sensor Networks

In this section we introduce different cryptographic libraries used in WSNs and why we have chosen a specific one. Afterwards some frameworks related to the Scopes Framework are discussed.

### 5.5.1 Cryptographic libraries

There is a wide variety of cryptographic libraries for WSNs available. From libraries with just single algorithms to suites of basic cryptography and advanced algorithms. In the beginning of cryptographic libraries for WSNs there were several different libraries available and it was not quite clear, if RSA or ECC is the best choice for the restricted hardware in WSNs. As it had to be expected ECC was the better choice and with the publication of TinyECC [72] in 2008 there was proof and an efficient and portable solution was available. TinyECC was written for the TinyOS environment, but as most of it is written in ANSI C and only some parts using nesC (a C extension used for TinyOS) it is portable, as we have also shown in the scope of this thesis (we ported the library to Contiki OS).

After the publication of TinyECC most of the newer libraries are based on TinyECC, as long as the algorithms use elliptic curve cryptography. In this section we will first give

an overview of TinyECC and after that show more advanced libraries that were possible candidates to satisfy our needs in the Scopes Framework.

#### 5.5.1.1 TinyECC

In [72] Liu et. al. publish an efficient and extensive library for elliptic curve cryptography in WSNs. It includes several algorithms and provides encryption and decryption, signature and verification, and a Diffie-Hellman key agreement protocol based on ECC. The algorithms are ECIES, ECDSA and ECDH.

TinyECC provides an implementation that is suited for 8-, 16- and 32-bit processor architectures and can be configured to be used with different elliptic curve sizes. It uses elliptic curves recommended by NIST [121] and allows special optimizations to speed up processing. Additionally some parts of the library can use optimized assembler implementations to additionally increase performance. The elliptic curve algorithms are separated from the underlying math operations, which supports the reuse of these operations in other algorithms.

**Discussion**

Possibly the first re-use of the TinyECC library was done by Kampanakis in [61] as he implemented a pairing algorithm on top of TinyECC. The algorithm is still provided with the TinyECC distribution, but only computes a compressed pairing [115]. As we ported TinyECC for our use, we also extended the Tate-pairing to compute the full pairing.

The combined algorithms of the ECC library and the Tate-Pairing made it the optimal choice for our Scopes Framework, as we do not need to include other libraries that may duplicate some parts of the functionality. As the implementation of TinyECC is modular, it is easy to exclude algorithms not used in our implementation from the framework's code-image.

#### 5.5.1.2 TinyPairing

The TinyPairing Library [143, 144] claims to be the first library with bilinear pairing functions available out-of-the-box. It not just provides the computation of a bilinear pairing, but also algorithms for encryption/decryption and different signature schemes based on pairings.

The provided performance is comparable to the performance provided by TinyECC with better RAM utilization, as the space used is drastically smaller for the encryption and signature algorithms. The pairing computation is highly optimized compared to the Tate computation based on TinyECC. This shows that pairing computation is possible in an efficient way, even on resource constrained systems like WSN nodes.

**Discussion**

Unfortunately the library is written in nesC that is only used in TinyOS, so the library is not easily portable to other operating systems used in WSNs and made it unavailable as choice for our framework.

#### 5.5.1.3 TinyTate

TinyTate [92] is a very early approach to compute a Tate pairing on WSN nodes. It is also based on TinyECC as the library for elliptic curve operations. With its 30 seconds per pairing computation on an 8-bit microcontroller it is slower than our implementation which is also also based on TinyECC. Therefore, this approach is not an option for our framework.

#### 5.5.1.4 TinyPBC

The TinyPBC library [93] evolved from the TinyTate project and is based on RELIC [2]. RELIC provides the basic mathematical functions used by TinyPBC.

TinyPBC optimized the pairing computation to 5.5 seconds. This was achieved by using binary fields and super singular curves with an optimized binary field multiplication.

**Discussion**

This library is also implemented for TinyOS and has some dependencies that are hard to fulfill. In our tests as a candidate for our framework we had a hard time to even get the library running on our nodes. At the time of our evaluation this library was not in a state that it could serve our purposes of a lean and efficient library for elliptic curve and pairing calculations.

#### 5.5.1.5 MoTE-ECC

MoTE-ECC [73] provides (ephemeral) ECDH on two specific elliptic curve types. These are Montgomery and twisted Edwards curves. The library is highly optimized for 8-bit AVR MCUs and allows four different key sizes, between 160 and 256 bit. Additionally the library was hardened against simple power analysis attacks.

The optimized calculations on the selected curves allow the ECDH computation to complete in 1.22 seconds, compared to 4.35 seconds of TinyECC in the same setting. Additionally, the memory footprint of RAM and ROM is drastically reduced compared to TinyECC.

**Discussion**

This library is also based on TinyOS and implemented in assembler for AVR MCUs. As we are using MSP430 MCUs this library was not compatible with our sensor nodes. Nonetheless, the results achieved are quite impressive.

#### 5.5.1.6 Cooperative Ciphertext-policy Attribute-based Encryption (C-CP-ABE)

The approach described in [128], cooperative CP-ABE, is not a complete security library. But, it evaluates an approach to allow the usage of CP-ABE even in the smallest sensor nodes, that we were also thinking of, but we could not use it since it did not match with the requirements of our Scopes Framework.

The idea of cooperative CP-ABE is to offload the expensive exponentiations of the regular CP-AB from resource constrained nodes to assistant nodes that have no resource constraint in the sense of limited power supply. The exponentiations are used in the key generation and encryption steps. Where key generation is done at the attribute authority, which uses a server outside the WSN to generate keys. Encryption is the costly operation in the network that is optimized in this approach. When encrypting, the calculations are spread over some assistant nodes, the authors propose 5 of them in the neighborhood of the calculating node. The results are then incorporated into the encrypted message to be sent.

The evaluation shows that the offloading works, the constrained node has to compute only some multiplications and the assistant node computes the exponentiations offloaded. But, the receiver of the encrypted message has to compute a high number of additional multiplications, which may cause additional load on other resource constrained nodes. This depends on the messaging scheme in the network.

**Discussion**

As mentioned above the offloading scheme is a neat idea, but it would not be easy to integrate it into the Scopes Framework, as the communication with the assistant node is just one-to-one communication and the Scopes Framework is designed to use one-to-many communication. An additional drawback for the use in Scopes are the additional multiplications on the decrypting side, as these are always the child nodes in Scopes, the encryption is done on the root node, which will mostly be a node with more capabilities and resources, compared to the child nodes. So, this scheme is not feasible for the Scopes Framework.

### 5.5.2 Security Frameworks

In this section we show frameworks that include security in their designs. These are all approaches for WSNs and related to our Scopes Framework. We show a broad range of frameworks, from peer-to-peer approaches to computationally demanding ABE approaches.

#### 5.5.2.1 SMEPP Light

SMEPP Light [5, 129] is based on SMEPP [15] for embedded peer-to-peer networks. The embedded components used in these networks are much more capable as the ones

used in WSNs. That is why SMEPP Light is only a subset of SMEPP and provides basic functionality including a replacement of XML self description with a bit vector technique. The approach explained here is valid for SMEPP Light and also SMEPP, therefore we will only speak of SMEPP from now on.

In SMEPP peers organize themselves into groups. Inside these groups nodes interact via a publish/subscribe mechanism. A node can subscribe for events of another node of the group. Relevant events are then automatically received and processed. Groups can be open or closed. To join a closed group a node has to authenticate itself, a public group can be joined by any node. The routing used is implemented using the directed diffusion paradigm.

SMEPP Light provides security on two levels. First on network level and second on group level. Both are secured via symmetric encryption. On network-level two keys are used, one for encryption (confidentiality) and one to compute a MAC (integrity). On group-level there are three keys. One is the so called *masterKey*, the other two are session keys to again encrypt and compute a MAC for a message. The masterKey is used as shared secret to join a closed group. When a node has joined a group it retrieves the two session keys and can then send encrypted messages to its peers. The masterKey is set at compile time and the session keys can be changed at runtime.

#### Discussion

Sharing the same principles with the Scopes Framework in its WSN section (group-based communication and routing based on directed diffusion) SMEPP Light is different from Scopes. Groups have to be planned at compile time, as secure, closed groups have to get their shared secret at this stage. Furthermore, the communication paradigm inside a group is quite different. SMEPP Light seems to allow node-to-node connections, where scopes introduce the scope root to scope member communication. Single point-to-point connections are possible, but expensive. As both approaches are based on the same paradigms SMEPP Light may be expected to be less efficient because of the overhead of the point-to-point connections.

Also if a node gets compromised the security measures of the Scopes Framework make it possible to exclude single nodes from a group and the whole network. With the static group keys this is not possible in SMEPP Light. But SMEPP allows to update the group's session keys, this at least enables backward secrecy.

### 5.5.2.2 Secure Group-based WSN Architecture

Garcia et. al. propose a secure group-based communication architecture in [35]. It is self-organizing and creates distinct groups in a WSN. Node IDs are unique in a group and contain the node position, that is acquired using GPS or similar. The node creating a group selects the group ID which has to be network wide unique. A group has a center node and border nodes. The center node manages the group key and has no other special

function. The border nodes are at the border of a group and can communicate with other groups. An event that occurs at a node is forwarded to all other group members.

In this approach, also two distinct security levels are defined: Network- and group-level (or Inter- and Intra-group security). The network-based security is based on symmetric encryption with a shared key. This key is obtained from another node in the network after authenticating. Each group has its own group key. It is managed by the center node. If a node is moved it may change its group membership, this includes replacing the old group key with the one from the new group. The group key is not distributed to the group members but calculated by each node. The center node stores a logical tree structure of the group and based on the position in this tree and a correction factor each node can compute its group key and each node knows the keys towards the center node.

**Discussion**

The grouping approach proposed here has a huge difference to the scopes approach. In the Scopes Framework a core element is the description of the group. In this approach there is no description what node may join which group. It is only based on location of the single node. Using GPS as source of location achieves a quite accurate localization, but puts a big burden in terms of energy consumption on the sensor nodes. Also the more or less random groups may not be an optimal choice depending on processing of an event. From the security perspective the shared key for the inter-group communication is a big drawback, as only one node has to be compromised and an adversary can join his own malicious nodes to the network. Of course a compromised node in the Scopes Framework allows cloning of a node, but after detection this node can be excluded from the network. In this approach it is not possible to replace the shared key without additional measures.

#### 5.5.2.3 SM-Sens

SM-Sens [34] is a cluster-based WSN security scheme with so called Guardians as observers of system security. Therefore, nodes in the WSN can fulfill four different roles: regular, cluster-head, gateway or guardian node. Regular nodes execute the application and transmit the results to their cluster-heads. Cluster-heads are responsible to acquire data from their sensing/regular nodes, aggregate data and communicate with the base station. They also distribute keys among cluster nodes. Gateway nodes are responsible for building a network backbone and have at least two cluster-heads or the base station in communication range. Guardians are responsible for observing the cluster-heads and forward from time to time data they received from cluster/regular nodes around them to the base station. The base station compares the data from the cluster-heads and the guardian nodes and decides if a node is compromised and may be excluded from the network.

To secure the approach four different kinds of keys are available. Individual keys that are used by regular cluster nodes. Cluster keys that are shared between a cluster-head and its cluster-nodes. Cluster-head keys that are shared between all cluster-heads and are used to communicate with the base station and region keys that are shared by cluster-heads and the guardians of a region of the network. For example, the communication between the cluster-heads or guardians and the base station is secured by ECC encryption. Another example is the distribution of a cluster key, where the cluster-head uses one of the individual keys of the cluster nodes and sends the encrypted cluster key to its nodes, one by one.

**Discussion**

SM-Sens is not providing any kind of grouping. It uses clustering to structure the network, but only for the purpose of routing. Nevertheless SM-Sens provides a wide variety of security measures to ensure different security goals. The system introduces four roles, where the guardian role is used to provide a second data-route to the base-station. With these two data-flows the base-station can ensure that the data is not altered on its way towards it. Additionally there are a bunch of different cryptographic keys each node has to store to be able to communicate with its peers. Depending on a node's role and what roles it is communicating to, different keys are used. This ensures that, when a key is compromised, there is another channel to replace the compromised key only on authorized nodes. On the other hand the large number of keys and the data that must be stored consume a big part of the available memory. Also the guardian node forwards a smaller number of messages compared to the real data stream. So, to compare the real and the guardian data stream there have to be variations allowed in a defined range. If the attacker knows these parameters he can alter data in an amount that may not be detected because the predefined ranges are not violated. Nonetheless the guardian role is a very interesting concept.

#### 5.5.2.4 Fine-grained Distributed Data Access Control (FDAC)

Yu et. al. propose in [150] a fine-grained distributed access-control scheme for data in WSNs, called FDAC. It uses Key policy - attribute-based encryption (KP-ABE) [38] to secure the data or more exactly the access to data inside the network. To achieve this KP-ABE uses a reverse approach compared to CP-ABE. The private keys distributed to the users of the network contain an access-structure where it is described what kind of data they are allowed to decrypt. After generation of data at the nodes the data is encrypted using the system's public-key and a set of properties is connected to the data. As in CP-ABE the access-structure is based on defined properties and defines via boolean expressions what combinations of properties are needed to decrypt data with a given private key. An extension to KP-ABE is a revocation mechanism, where all the keys of a system can be updated to revoke a compromised key. The compromised key has to be excluded from the update mechanism.

**Discussion**

Following its name, FDAC does not provide a structured network or some kind of grouping scheme. It uses KP-ABE and therefore works in the opposite way the Scopes Framework does. Here the data is tagged with properties and the users have an access structure in their private keys that limits their ability to decrypt data. In Scopes the sensor nodes have a set of properties and the access-structure is embedded in the scope creation message. In this sense the data (the scope) authenticates the user (the sensor node). An additional assumption made in the FDAC approach is the use of time-synchronization, which is not necessary for using scopes.

#### 5.5.2.5 Ring-based Secure Group Communication Scheme (RiSeG)

In [18] Cheikhrouhou et. al. propose a secure grouping scheme, RiSeG, for WSNs that are based on a ring topology for each group. In this approach three different kinds of roles are present: the base station, a group controller and end devices. An end device is just a group member. It has to store the node addresses that are located before and after it in the ring topology and a set of cryptographic keys to authenticate itself or encrypt/decrypt messages. A group member sends in regular intervals a HELLO message to confirm its presence to the group controller. The group controller has knowledge about all members of a group, it manages the group key and handles the members' HELLO messages. The base station is used in grouping related operations and handles authentication of a sensor node, blacklisting of (potential) compromised nodes, which get excluded from the network, and allows creating new groups or joining of a node.

The ring topology was chosen because of scalability reasons. In this scheme the group controller and members have only to send one message each for a message that has to be sent to the whole group. In the pre-deployment phase all the nodes get preloaded with the necessary keys for the different cryptographic operations, like pair-wise key establishment, computing a message authentication code (MAC) of a message and computing an elliptic curve signature (ECDSA). The communication between two nodes is always encrypted by symmetric encryption and the pair-wise keys. Therefore, if a message has to be forwarded through different nodes it has to be re-encrypted for every connection.

To create a group a node sends a join request for this non-existing group to the base station. Base station and node authenticate themselves to each other and the node receives a group-creation-invitation from the base station, if the authentication succeeded. If the node accepts the invitation it becomes the group's group controller. To join an existing group the same join-request is sent to the base station. If authentication went well a join-inform is sent from the base station to the group controller and the further steps to join are handed over to the group controller. The group controller sends then a join-key message to the requesting node and includes it to the ring topology, by sending its new previous node a ring-update. A new node is always appended at the end of the ring (the group controller is assumed the head of the ring). After joining or leaving of a node the group key is renewed to ensure forward- and backward-secrecy. A group controller

can leave or switch. To do that the group-management-information is forwarded to the next node in the ring and if it accepts it becomes the new group controller. If it does not accept the information is forwarded to the next node in the ring. In case no node accepts to be group controller the group is destroyed. To leave a group a node sends a leave-request to the group controller which then updates the ring topology and the group key.

**Discussion**

The basic idea of a ring topology for a group in WSNs seems easy, but gets quite expensive in deployments with more than one-hop communication range. Especially as in this approach new nodes are just appended to the end of the ring and a message has to traverse many hops if the neighboring nodes in the ring are located at opposite positions in the real sensor network. From a security point of view each step in the group creation or join processes is secured, but with the base station included in this process there is a single point of failure to prevent the network from even joining existing groups. In contrast the Scopes Framework is still fully operable in networks that are partitioned at runtime as long as the root of a scope is reachable. Also the HELLO messages can introduce a considerable amount of load in a WSN, depending on their frequency and network size. Scopes uses a similar approach with its refresh messages, but as the messages are sent to the member node's the scope root (group controller) is unburdened from processing a large number of messages. The behavior observed in RiSeG drains much energy from the group controller's neighborhood, same as for the base station. Beside the data messages also the HELLO messages are sent toward the group controller, where a high number of messages in medium to large sensor networks are accumulated very fast.

The RiSeG uses some assumptions that the Scopes Framework also uses, like overlapping groups, a group controller or security to ensure only valid nodes can join or create a group. But it uses a different notion of a group and bases the group parameters only on the sensed data of a node. Scopes does not restrict the grouping parameters to the type of sensed data, but leaves it to the user of the network and the scenario to define a set of matching criteria. The security in RiSeG is centered around the base station and the group controller, where in scopes the security is only based on the group controller, still in both approaches the needed cryptographic keys are deployed in a bootstrap phase before physical deployment, but can be managed after deployment in Scopes.

#### 5.5.2.6 Di-Sec

With Di-Sec [130] the authors propose a novel distributed intrusion detection system for WSNs. The system is designed to monitor the single sensor nodes in the background, so that it is transparent to WSN applications or tasks.

It consists of a monitoring core (M-Core), a communication module (COMM), the sensing module (Sense) and the detection and defense modules (DDMs). The M-Core

provides all the functionality needed by the DDMs an also scans messages traversing through the Sensing and COMM modules. COMM provides the communication interface that has to be used in order to use Di-Sec in a system. All network messages, in or out, are forwarded through this module to analyze, mark and send the (encrypted) data. The Sense module is built analogously to the COMM module and scans all the data coming from sensors and going to actuators to detect any irregularities. The DDMs provide attack detection or defense mechanisms against specific threats. Therefore, there can be several DDMs in the system and they can dynamically be exchanged.

To allow not only WSN experts to configure the system the M-Core Control Language (MCL) is used to program the DDMs.

**Discussion**

The Scopes Framework only copes with its immediate threats and it protects the data inside the scopes. But there are much more possible attacks and it can not be known up-front which attacks an adversary will execute. Di-Sec may be a valuable addition to many WSN-Frameworks, not only Scopes, to allow threat detection and countermeasures with the ability to retrofit new functions on a network-wide scope. Di-Sec aims to be a general solution to security threats. It provides proxy like modules for every nodes input/output and specific detection and countermeasures functionality for different threats. As a general approach it can be used to secure a WSN against common attacks, whereas the framework or protocol specific threats, like the one we cope within SecScopes, will be covered by them. This approach is a valuable complement for WSNs in general. Nevertheless, Di-Sec copes with security threats only, it is the WSN version of a malware and intrusion detection program. It does not manage a WSN, like Scopes does it with the support of partitioning of the network or the support of multiple concurrent tasks.

#### 5.5.2.7 Object Security Architecture (OSCAR)

In [133] Vucinic et. al. propose a new security architecture for WSNs using CoAP, as the end-to-end paradigm of the standardized DTLS algorithm does not match the many-to-many paradigm of CoAP itself. The new security architecture OSCAR introduces Authorization Servers that authorize data producers and consumers to work with each other. Therefore, DTLS is still in use for exchanging an access secret between authorization server and producer or consumer. The data exchange between the latter two is done through ECDSA and a symmetric encryption, e.g. AES128.

When a sensor node (producer) wants to provide data with its sensors, it subscribes with the Authorization Servers and receives a resource access secret for the kind of data it can provide. This is done through a DTLS channel. When a consumer wants to retrieve data from a producer, it requests the kind of data at the Authorization Servers and receives an access secret, if it is authorized. This is also done through a DTLS channel. The request for the actual data is done by the consumer directly or via a cloud or proxy-

server to one or more producer nodes. This request and the following data exchange is encrypted using a session key that is derived from the matching access structures. This way the data can only be decrypted, if the consumer has a matching access structure. The producer also signs the data with its device certificate and the consumer can check the signature to ensure, that the message originates from the correct producer.

#### Discussion

OSACR is a very new approach, but with a similar access control scheme as attribute-based encryption. But, this is achieved using classic symmetric and asymmetric cryptography and a central infrastructure, that has to be available at all times to keep the network operational. With the Scopes Framework we intentionally designed our system to work without fixed infrastructure, as this may not be easily possible, for example, in our container harbor scenario, where producer nodes had to authorize their sensors on every ship and in every harbor to the local authorization servers. This would introduce high management effort before the journey of a container. As sensor node has to have the allowance on every authorization server on its way to the destination to connect to the different networks.

## 5.6 Summary

Wireless sensor networks and network security are two active research areas. This is clearly illustrated by the variety of approaches we presented and discussed. The main contributions of this thesis are a dynamic grouping mechanism for WSNs that is both flexible and economical, the Scopes Framework, and a security mechanism based on attribute-based encryption that works in WSNs and effectively supports the dynamic grouping and structuring of the sensor network, SecScopes. It is this combination of features that sets the research presented in this thesis apart from the many related approaches. The Scopes Framework developed as part of this thesis is a grouping mechanism that is flexible and can be tailored for many different application scenarios. The secure extension of the Scopes Framework, SecScopes, incorporates security mechanisms that were previously only known to work in a PC environment, to support access control to a scope and enable continued security.

# 6 Conclusions

## Contents



With the Internet of Things (IoT) at our doors the problem of efficient and data-oriented communication of hundreds of devices has reached every one of us. The step towards interconnected WSNs used by multiple parties and for different purposes is getting closer and makes the consideration of secure multi-purpose WSNs even more important. With the Scopes Framework and SecScopes we provide a viable solution for structuring multi-purpose WSNs and securing them in a way that supports their data-oriented nature.

### 6.1 What we achieved

The major contribution of this work is the design and evaluation of the Scopes Framework. It leverages an efficient grouping mechanism, called scopes, that can restrict communication in a WSN to groups of nodes that satisfy a set of configurable properties. This relieves unrelated areas of the network from the load and saves network resources. The framework also supports the management of concurrent tasks, where concurrent means multiple installed and active tasks in a WSN. The Scopes Framework also provides a high configurability to enable the use in many scenarios. Therefore, it keeps scenario-related aspects configurable to archive this high degree of reusability. Example scenarios we evaluated throughout this work are the container harbour scenario [55] and TikiDB [57], a WSN wide database system similar to TinyDB [76]. But, with a bigger focus on sensing and acting compared to the mostly sensing capabilities of TinyDB.

Security was recognized as an important aspect from the beginning of this work. The second important contribution of this work is the proposed security concept of the Scopes Framework, called SecScopes. It utilizes attribute-based encryption to secure the scoping process and allow access control and enable secure key exchange for the light weight, symmetric transport encryption. It provides scope wide group-encryption of messages, including secure key-exchange, and takes scenarios of a breach in the security chain into account, e.g., the exclusion of a malicious node in the WSN.

The two major properties that were achieved with the proposed framework are a multi-purpose and secure WSN. To enable the multi-purpose property we enabled the

framework to be highly configurable. Therefore, we modularized our implementation and introduced defined interfaces to the modules surrounding scopes. This allows the substitution of these modules with specialized versions that support a given scenario. Additionally, we introduced scopes to enable efficient group communication that can be easily configured to an application's communication needs.

To enable a secure system we introduced not just symmetric encryption for encrypted communication, but also a concept for access control to the system, secure key exchange and rekeying in the running network. To achieve this, we introduced attribute-based encryption in WSNs and enabled a scaled deployment of the attribute-based algorithms depending on the role and resources of a sensor node. This makes this kind of encryption possible also on very low-power nodes.

To ensure proper performance we did extensive evaluations with the different evolutions of the Scopes Framework, as seen in the previous chapter. The Scopes Framework proved efficient for low-power WSNs and results showed that it performs better in more dense networks for reliable operation. SecScopes proved possible in low-power WSNs, but can only be used efficiently in specific scenarios. In WSNs where the sensor nodes are more powerful it is efficiently usable. The evaluations provide performance indicators on the sensor nodes thenselves and in the network. We show computation times for the different algorithms used and memory usage for RAM and ROM. To compare the performance in the WSN we defined the performance indicators of nodes that join in a scope over its lifetime and the goodput of periodic and bulk data traffic. Goodput means percentage of possible messages delivered in a given evaluation setup. We showed three evolutions of the Scopes Framework in our evaluation. Our first prototype was based on the SOS operating system and was used for the first feasibility study. The first evolution was then based on Contiki and contains the full feature-set of the Scopes Framework. The last evolution is SecScopes, the security enabled incarnation of the Scopes Framework.

The detailed evaluation showed the general feasibility of the scopes approach in wireless sensor networks. In sparsely populated networks the results show less stable scope creation and, therefore, fewer delivered messages. Medium to dense networks are better suited to our approach, as more alternative routing path are available to ensure proper delivery of messages. The introduction of security in SecScopes led first to a drastic drop in performance, because the additional data to be transferred resulted in many more message collisions and consequently to message loss. We could improve the performance to levels similar to insecure scopes only by introducing additional measures to detect and treat these situations by using CRC for integrity and fragmentation management.

## 6.2 How this work can be enhanced in the future

The results obtained in this thesis can be improved in various areas.

A further analysis of the SecScopes architecture showed a possible flaw in scope deletion. Currently it cannot be checked if a scope deletion command is issued by the root node. This may be possible by having the root node sign the deletion message. But this would impose an additional signature algorithm and a scope public/private key pair. The influence on the performance is equal to the one of scope creation; scope deletion would be getting an expensive operation, but it would be executed only once. The impact of this flaw is increasing the management overhead and a short interruption of message transfer on the specific scope. As the root node does not accept a false deletion command, the next SR that is send for the scope will recreate the scope. Additionally, a false deletion command can only affect the routing tree from the receiving scope member node away from the root node, because of the routing direction.

The evaluation of the security enhancements showed clearly that more powerful nodes are needed to use these algorithms to their full extent. Today's ARM Cortex-M Family [3] provides a multiple of performance, compared to, e.g., Tmotes. Of course this costs more energy, but still a lifetime of one year and more for a sensor node is achievable on batteries. This would allow a much faster processing of the crypto-algorithms used and more memory space for scenario dependent adjustments and tasks. In-network the performance and stability could be enhanced by introducing compression to reduce the size of messages that was a major issue introducing security. Further optimization, for example, of the CP-ABE access structure could lead to improved behavior. Aggregation in a scope could be investigated for message transmission. However, this may cause problems in the integration with the security mechanisms.

Another area of further research may be the support of the cryptographic algorithms by dedicated co-processors, e.g., FPGAs with a specific implementation of the proposed algorithms. Another way may be the use of the ARM Cortex-M's DSP-library to accelerate processing without additional hardware.

The introduction of further security measures is another area of enhancements. Currently the code inside a sensor node can potentially access all memory areas due to the lack of the hardware to restrict access to certain areas. To achieve this hardware support is mandatory and can be found in, e.g. ARMs TrustZone technology [4], this will be available in the near future to smaller ARM-cores like the Cortex-M-family.

# Bibliography


[1] Kemal Akkaya and Mohamed Younis. A survey on routing protocols for wireless sensor networks. *Ad Hoc Networks*, 3:325–349, 2005.

[2] D. F. Aranha and C. P. L. Gouvêa. RELIC is an Efficient Library for Cryptography. `http://code.google.com/p/relic-toolkit/`, August 2012.

[3] ARM Ltd. Cortex-M series. Website, September 2015. http://www.arm.com/products/processors/cortex-m.

[4] ARM Ltd. TrustZone. Website, September 2015. http://www.arm.com/products/processors/technologies/trustzone.

[5] Javier Barbarán, José A. Dianes, Manuel Díaz, Daniel Garrido, Luis Llopis, Ana Reyna, and Bartolomé Rubio. Programming wireless sensor networks applications using smepp: a case study. In *Proceedings of 3rd ERCIM Workshop on eMobility*, Enschede, Netherlands, May 2009.

[6] Eugen Berlin, Pablo Guerrero, Arthur Herzog, Daniel Jacobi, Kristof van Laerhoven, Alejandro Buchmann, and Bernt Schiele. Demo abstract: Whac-A-Bee - a sensor network game. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, SenSys '09, pages 333–334, New York, USA, November 2009. ACM Press.

[7] John Bethencourt. Intro to bilinear maps. Slides, March 2006. http://www.cs.berkeley.edu/~bethenco/bilinear_maps.pdf.

[8] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *IEEE Symposium on Security and Privacy*, SP 2007, pages 321–334, Berkeley, CA, June 2007.

[9] BlueKrypt. Cryptographic key length recommendation. Website, December 2015. http://www.keylength.com.

[10] Dan Boneh and Matt Franklin. Identity-based encryption from the weil pairing. In Joe Kilian, editor, *Advances in Cryptology (CRYPTO 2001)*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229. Springer Berlin / Heidelberg, 2001.

[11] Philippe Bonnet, Johannes Gehrke, and Praveen Seshadri. Towards sensor database systems. In *Proceedings of the Second International Conference on Mobile Data Management*, January 2001.

[12] Mirko Bordignon, Jayedur Rashid, Mathias Broxvall, and Alessandro Saffiotti. Seamless integration of robots and tiny embedded devices in a peis-ecology. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, IROS 2007, San Diego, CA, 2007.

[13] Carsten Bormann, Angelo P. Castellani, and Zach Shelby. CoAP: An application protocol for billions of tiny internet nodes. *Internet Computing, IEEE*, 16(2):62–67, March 2012.

[14] B. Cain, S. Deering, I. Kouvelas, B. Fenner, and A. Thyagarajan. *RFC 3376 – Internet Group Management Protocol, Version 3*. Network Working Group, October 2002.

[15] Rafael J. Caro, David Garrido, Pierre Plaza, Rodrigo Roman, Nuria Sanz, and Jose L. Serrano. Smepp: A secure middleware for embedded p2p. In *ICT Mobile and Wireless Communications Summit*, ICT-MobileSummit 2009, Santander (Spain), June 2009.

[16] Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems (TOCS)*, 19(3):332–383, 2001.

[17] Miguel Castro, P. Druschel, Anne-Marie Kermarrec, and Antony I.T. Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications*, 20(8):1489–1499, September 2006.

[18] Omar Cheikhrouhou, Anis Koubâa, Gianluca Dini, and Mohamed Abid. Riseg: a ring based secure group communication protocol for resource-constrained wireless sensor networks. *Personal and Ubiquitous Computing*, 15:783–797, 2011. 10.1007/s00779-011-0365-5.

[19] Xiangqian Chen, K. Makki, Kang Yen, and N. Pissinou. Sensor network security: a survey. *Communications Surveys Tutorials, IEEE*, 11(2):52 –73, April 2009.

[20] Gregory Chockler, Roie Melamed, Yoav Tock, , and Roman Vitenberg. Spidercast: a scalable interest-aware overlay for topic-based pub/sub communication. In *In Proceedings of the international conference on distributed event-based systems*, pages 14–25. ACM, 2007.

[21] Paolo Costa, Geoff Coulson, , Cecilia Mascolo, Gian Pietro Picco, and Stefanos Zachariadis. The RUNES middleware: A reconfigurable component-based approach to network embedded systems. In *Proceedings of 16th International Symposium on Personal Indoor and Mobile Radio Communication*, PIMRCO 05, pages 806 – 810. IEEE Press, September 2005.

[22] Paolo Costa, Geoff Coulson, Richard Gold, Manish Lad, Cecilia Mascolo, Luca Mottola, Gian Pietro Picco, Thirunavukkarasu Sivaharan, Nirmal Weerasinghe, and Stefanos Zachariadis. The runes middleware for networked embedded systems and its application in a disaster management scenario. In *Proceedings of the Fifth Annual IEEE International Conference on Pervasive Computing and Communications*, PerCom '07. IEEE Computer Society, 2007.

[23] Crossbow Technology Inc. *Mica2 Datasheet*. Crossbow Technology Inc., San Jose, California, 2002.

[24] David E. Culler and Hans Mulder. Smart sensors to network the world. In *Scientific American*, pages 84–91. Scientific American, Inc., June 2004.

[25] Steve Deering. *RFC 1112 – Host Extensions for IP Multicasting*. Stanford University, August 1989.

[26] Steve Deering. *Multicast Routing in a Datagram Network*. Ph.d. thesis, Stanford University, Palo Alto, California, December 1997.

[27] Adam Dunkels, Björn Gronvall, and Thiemo Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Proceedings of 29th Annual IEEE International Conference on Local Computer Networks*, pages 455–462, Nov. 2004.

[28] Adam Dunkels, Fredrik Österlind, Nicolas Tsiftes, and Zhitao He. Software-based on-line energy estimation for sensor nodes. In *Proceedings of the Fourth Workshop on Embedded Networked Sensors*, Emnets IV, Cork, Ireland, June 2007.

[29] Adam Dunkels, Oliver Schmidt, Thiemo Voigt, and Muneeb Ali. Protothreads: Simplifying event-driven programming of memory-constrained embedded systems. In *In Proceedings of the ACM Conference on Embedded Networked Sensor Systems*, SenSys '06, pages 29–42. ACM Press, 2006.

[30] Morris Dworkin. *Recommendation for Block Cipher Modes of Operation*. NIST, December 2001.

[31] W. Fenner. *RFC 2236 – Internet Group Management Protocol, Version 2*. Network Working Group, Xerox PARC, November 1997.

[32] Ludger Fiege. *Visibility in Event-Based Systems*. PhD thesis, Technische Universität Darmstadt, Fachbereich Informatik, Darmstadt, Germany, April 2005.

[33] Ludger Fiege, Mira Mezini, Gero Mühl, and Alejandro P. Buchmann. Engineering event-based systems with scopes. In B. Magnusson, editor, *Proceedings of the European Conference on Object-Oriented Programming (ECOOP'02)*, volume 2374 of *Lecture Notes in Computer Science*, pages 309–333, Malaga, Spain, June 2002. Springer-Verlag.

[34] Luiz H. Freitas, Kalil A. Bispo, Nelson S. Rosa, and Paulo Cunha. Sm-sens: Security middleware for wireless sensor networks. In *Global Information Infrastructure Symposium*, GIIS '09, pages 1–7, June 2009.

[35] Miguel Garcia, Jaime Lloret, Sandra Sendra, and Raquel Lacuesta Gilaberte. Secure communications in group-based wireless sensor networks. *IJCNIS*, 2(1):8–14, April 2010.

[36] Joao Girao, Dirk Westhoff, and Markus Schneider. CDA: Concealed data aggregation for reverse multicast traffic in wireless sensor networks. In *Proceedings of IEEE International Conference on Communications*, volume 5 of *ICC 2005*, pages 3044 – 3049. IEEE, May 2005.

[37] D. Gordon. Discrete logarithms in ff(p) using the number field sieve. *SIAM Journal on Discrete Mathematics*, 6:124–138, 1993.

[38] V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute based encryption for fine-grained access control of encrypted data. In *ACM conference on Computer and Communications Security*, CCS. ACM, 2006.

[39] Tomislav Greguric. Gezielte softwareverteilung in drahtlosen sensornetzwerken. Diploma thesis, Technische Universität Darmstadt, 2011.

[40] Pablo Guerrero, Daniel Jacobi, and Alejandro Buchmann. Workflow support for wireless sensor and actor networks. In *4th International Workshop on Data Management for Sensor Networks*, September 2007.

[41] Pablo Guerrero, Daniel Jacobi, Ilia Petrov, and Alejandro Buchmann. Scopes declarative language. Technical report, Technische Universität Darmstadt, January 2010. http://www.dvs.tu-darmstadt.de/research/scopes/ScopesSyntax.pdf.

[42] Pablo E. Guerrero, Alejandro P. Buchmann, Abdelmajid Khelil, and Kristof Van Laerhoven. TUD$\mu$Net, a metropolitan-scale federation of wireless sensor network testbeds. In *9th European Conference on Wireless Sensor Networks*, EWSN, Trento, February 2012.

[43] Nils Gura, Arun Patel, Arvinderpal Wander, Hans Eberle, and Sheueling Chang Shantz. Comparing elliptic curve cryptography and rsa on 8-bit cpus. In Marc Joye and Jean-Jacques Quisquater, editors, *Proceedings of 6th International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2004)*, volume 3156 of *Lecture Notes in Computer Science*, pages 119–132, Cambridge, MA, USA, August 2004. Springer.

[44] Chih-Chieh Han, Ram Kumar, Roy Shea, Eddie Kohler, and Mani Srivastava. A dynamic operating system for sensor nodes. In *Proceedings of the 3rd international conference on Mobile systems, applications, and services*, MobiSys '05, pages 163–176, New York, NY, USA, 2005. ACM Press.

[45] Darrel Hankerson, Alfred Menezes, and Scott Vanstone. *Guide to Elliptic Curve Cryptography*. Springer-Verlag, New York, 2004.

[46] Wendi Rabiner Heinzelman, Anantha Chandrakasan, and Hari Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences*, January 2000.

[47] Wendi Rabiner Heinzelman, Joanna Kulik, and Hari Balakrishnan. Adaptive protocols for information dissemination in wireless sensor networks. In *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking*, MobiCom '99, pages 174–185, New York, NY, USA, 1999. ACM.

[48] André Herms and Michael Schulze. Publish/subscribe middleware für selbstorganisierende drahtlose multi-hop-netzwerke. In *Proceedings of Fachgespräch Selbstorganisierende, Adaptive, Kontextsensitive verteilte Systeme*, SAKS '08, Wiesbaden, Germany, March 2008.

[49] Arthur Herzog, Daniel Jacobi, and Alejandro Buchmann. A3ME - an Agent-Based middleware approach for mixed mode environments. In *The Second International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies*, (UBICOMM 2008), pages 191–196, Valencia, Spain, October 2008. IEEE Computer Society Press.

[50] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister. System architecture directions for networked sensors. *ACM SIGPLAN Notices*, 35(11):93–104, November 2000.

[51] Chalermek Intanagonwiwat, Ramesh Govindan, Deborah Estrin, John Heidemann, and Fabio Silva. Directed diffusion for wireless sensor networking. *IEEE/ACM transactions on networking*, 11(1):2–16, February 2003.

[52] Intel Corporation. *Intel PXA255 Processor: Electrical, Mechanical, and Thermal Specification*, February 2004.

[53] Daniel Jacobi. *Scopes in drahtlosen Sensornetzwerken: Grundlagen und Verfahren*. Vdm Verlag Dr. Müller, January 2007. Diplomathesis. Technische Universität Darmstadt. Advisors: Alejandro Buchmann and Jan Steffan.

[54] Daniel Jacobi, Marc Fischlin, and Alejandro Buchmann. *Security for Multihop Wireless Networks*, chapter Secure Multi-Purpose Wireless Sensor Networks, pages 247 – 272. CRC Press, 1st edition, April 2014.

[55] Daniel Jacobi, Pablo E. Guerrero, Ilia Petrov, and Alejandro Buchmann. Structuring Sensor Networks with Scopes. In *3rd IEEE European Conference on Smart Sensing and Context*, EuroSSC 2008, Zurich, Switzerland, October 2008. IEEE Communications Society.

[56] Daniel Jacobi, Pablo E. Guerrero, Ilia Petrov, and Alejandro Buchmann. Distributed network structuring with scopes. Technical Report 2741, Technische Universität Darmstadt, Darmstadt, Germany, October 2009.

[57] Daniel Jacobi, Pablo Ezequiel Guerrero, Khalid Nawaz, Christian Seeger, Arthur Herzog, Kristof Van Laerhoven, and Ilia Petrov. Towards declarative query scoping in sensor networks. In Kai Sachs, Ilia Petrov, and Pablo Guerrero, editors, *From Active Data Management to Event-Based Systems and More*, volume 6462 of *Lecture Notes in Computer Science*, pages 281–292. Springer, November 2010.

[58] Hans-Arno Jacobsen, Alex Cheung, Guoli Li, Balasubramaneyam Maniymaran, Vinod Muthusamy, and Reza Sherafat Kazemzadeh. The PADRES Publish/Subscribe System. In *Principles and Applications of Distributed Event-Based Systems*, pages 164–205. IGI Global, 2010.

[59] Antoine Joux. A one round protocol for tripartite diffie-hellman. In Wieb Bosma, editor, *Algorithmic Number Theory*, volume 1838 of *Lecture Notes in Computer Science*, pages 385–393. Springer Berlin / Heidelberg, 2000.

[60] Jörg Kaiser, Carlos Mitidieri, Cristiano Brudna, and Carlos Eduardo Pereira. Cosmic: A middleware for event-based interaction on can. In *Proceedings of 9th IEEE International Conference on Emerging Technologies and Factory Automation*, ETFA 2003, 2003.

[61] Panagiotis T. Kampanakis. Identity-based cryptography: Feasibility & applications in next generation sensor networks. Mastersthesis, North Carolina State University, Raleigh, NC, 2007.

[62] Chris Karlof and David Wagner. Secure routing in wireless sensor networks: attacks and countermeasures. *Ad Hoc Networks*, 1:293–315, 2003.

[63] Stamatis Karnouskos and Patrik Spiess. Towards enterprise applications using wireless sensor networks. In *9th International Conference on Enterprise Information Systems*, June 2007.

[64] Steffen Kilb. Design und implementierung von scopes für Contiki. Bachelorthesis, Technische Universität Darmstadt, September 2009. Advisors: Alejandro Buchmann and Daniel Jacobi.

[65] Matthias Kovatsch, Simon Duquennoy, and Adam Dunkels. A low-power coap for contiki. In *IEEE 8th International Conference on Mobile Adhoc and Sensor Systems*, MASS, pages 855–860, October 2011.

[66] Matthias Kropff, Christian Reinl, Kim Listmann, Karen Petersen, Katayon Radkhah, Faisal Karim Shaikh, Arthur Herzog, Armin Strobel, Daniel Jacobi, and Oskar von Stryk. Mm-ulator: Towards a common evaluation platform for mixed

mode environments. In *Proceedings of Int. Conference on Simulation, Modeling, and Programming*, SIMPAR. Springer, 2008.

[67] Mauro Leoncini, Giovanni Resta, and Paolo Santi. Partially controlled deployment strategies for wireless sensors. *Ad Hoc Netw.*, 7(1):1–23, January 2009.

[68] Philip Levis and David E. Culler. Maté: A Tiny Virtual Machine for Sensor Networks. In *10th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 85–95, New York, NY, USA, October 2002. ACM Press.

[69] Philip Levis, Neil Patel, David E. Culler, and Scott Shenker. Trickle: a self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In *Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation*, NSDI '04, Berkeley, CA, USA, 2004. USENIX Association.

[70] Chieh-Jan Mike Liang, Kaifei Chen, Nissanka Bodhi Priyantha, Jie Liu, and Feng Zhao. RushNet: Practical traffic prioritization for saturated wireless sensor networks. In *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems*, SenSys '14, pages 105–118, New York, NY, USA, 2014. ACM.

[71] Helger Lipmaa, Phillip Rogaway, and David Wagner. Counter-mode encryption. Contribution to NIST on CTR., September 2000.

[72] An Liu and Peng Ning. TinyECC: A configurable library for elliptic curve cryptography in wireless sensor networks. In *Proceedings of the 7th International Conference on Information Processing in Sensor Networks, SPOTS Track*, IPSN 2008, pages 245–256, April 2008.

[73] Zhe Liu, Erich Wenger, and Johann Großschädl. MoTE-ECC: Energy-scalable elliptic curve cryptography for wireless sensor networks. In Ioana Boureanu, Philippe Owesarski, and Serge Vaudenay, editors, *Proceedings of 12th International Conference on Applied Cryptography and Network Security*, ACNS, pages 361–379, Lausanne, Switzerland, June 2014. Springer International Publishing.

[74] Martin Lötzsch, Max Risler, and Matthias Jüngel. XABSL - a pragmatic approach to behavior engineering. In *Proceedings of IEEE/RSJ International Conference of Intelligent Robots and Systems*, IROS, pages 5124–5129, Beijing, China, 2006.

[75] Ben Lynn. Pairing-Based Cryptography (PBC) library. Website, January 2016. http://crypto.stanford.edu/pbc/.

[76] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. TinyDB: an Acquisitional Query Processing System for Sensor Networks. *ACM Trans. Database Syst.*, 30(1):122–173, March 2005.

[77] Alan Mainwaring, David Culler, Joseph Polastre, Robert Szewczyk, and John Anderson. Wireless sensor networks for habitat monitoring. In *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, WSNA '02, pages 88–97, New York, NY, USA, 2002. ACM.

[78] Arati Manjeshwar and Dharma P. Agrawal. Teen: a routing protocol for enhanced efficiency in wireless sensor networks. In *Proceedings of 15th International Parallel and Distributed Processing Symposium*, pages 2009 – 2015, April 2001.

[79] Pedro José Marrón, Andreas Lachenmann, Daniel Minder, Jörg Hähner, Kurt Rothermel, and Christian Becker. Adaptation and cross-layer issues in sensor networks. In *Proceedings of the International Conference on Intelligent Sensors, Sensor Networks & Information Processing*, pages 623–628. IEEE, December 2004.

[80] Pedro José Marrón, Daniel Minder, Andreas Lachenmann, and Kurt Rothermel. Tinycubus: An adaptive cross-layer framework for sensor networks. *it - Information Technology*, 47(2):87–97, 2005.

[81] K. Martinez, R. Ong, and J. Hart. Glacsweb: a sensor network for hostile environments. In *First Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks*, SECON 2004, pages 81–87. IEEE, 2004.

[82] K. Martinez, P. Padhy, A. Riddoch, R. Ong, and J. Hart. Glacial environment monitoring using sensor networks. In *Proceedings of the 1st Workshop on Real-World Wireless Sensor Networks*, REALWSN, Stockholm, Sweden, 2005.

[83] Miguel Matos, Ana Nunes, Rui Oliveira, and José Pereira. Stan: exploiting shared interests without disclosing them in gossip-based publish/subscribe. In *In proceedings of 9th international workshop on peer-to-peer systems*, April 2010.

[84] G. Montenegro, N. Kushalnagar, Jonathan Hui, and David E. Culler. *RFC 4944 – Transmission of IPv6 Packets over IEEE 802.15.4 Networks*. Internet Engineering Task Force (IETF), September 2007.

[85] Moteiv Corporation. *Tmote Sky Datasheet*. Moteiv Corporation, San Francisco, California, 1.0.2 edition, June 2006.

[86] Luca Mottola and Gian Pietro Picco. Logical neighborhoods: A programming abstraction for wireless sensor networks. In Phillip B. Gibbons, Tarek F. Abdelzaher, James Aspnes, and Ramesh Rao, editors, *Proceedings of 2nd IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS '06)*, volume 4026 of *Lecture Notes in Computer Science*, pages 150–168, San Francisco, CA, USA, June 2006. Springer.

[87] Gero Mühl. *Large-Scale Content-Based Publish/Subscribe Systems*. PhD thesis, Technische Universität Darmstadt, Fachbereich Informatik, Darmstadt, Germany, September 2002.

[88] Sascha Müller, Stefan Katzenbeisser, and Claudia Eckert. On multi-authority ciphertext-policy attribute-based encryption. In *International Conference on Information Security and Cryptology*, ICISC 2008, 2008.

[89] Sirajum Munir and John A. Stankovic. DepSys: Dependency aware integration of cyber-physical systems for smart homes. In *Proceedings of ACM/IEEE International Conference on Cyber-Physical Systems*, ICCPS, pages 127–138, April 2014.

[90] James Newsome, Elaine Shi, Dawn Song, and Adrian Perrig. The sybil attack in sensor networks: Analysis & defenses. In *Proceedings of The Third International Symposium on Information Processing in Sensor Networks*, IPSN '04, Berkeley, California, USA., April 2004. ACM Press.

[91] Ryan Newton and Matt Welsh. Region streams: Functional macroprogramming for sensor networks. In *Proceedings of the First Workshop on Data Management for Sensor Networks*, DMSN 2004, Toronto, Canada, August 2004.

[92] L. B. Oliveira, D. F. Aranha, E. Morais, F. Daguano, J. López, and R. Dahab. TinyTate: Computing the Tate Pairing in Resource-Constrained Sensor Nodes. In *IEEE International Symposium on Network Computing and Applications*, NCA 2007, pages 318–323. IEEE, 2007.

[93] Leonardo B. Oliveira, Diego F. Aranha, Conrado P. L. Gouvêa, Michael Scott, Danilo F. Címara, Julio López, and Ricardo Dahab. Tinypbc: Pairings for authenticated identity-based non-interactive key distribution in sensor networks. *Comput. Commun.*, 34(3):485–493, March 2011.

[94] Oracle. Sun SPOT World. Website, January 2016. http://www.sunspotdev.org.

[95] Helge Parzyjegla, Daniel Graff, Arnd Schröter, Jan Richling, and Gero Mühl. Design and implementation of the rebeca publish/subscribe middleware. In *From Active Data Management to Event-Based Systems and More*, volume 6462 of *Lecture Notes in Computer Science*, pages 124 – 140. Springer Berlin / Heidelberg, 2010.

[96] Al-Sakib Khan Pathan, Hyung-Woo Lee, and Choong Seon Hong. Security in wireless sensor networks: issues and challenges. In *The 8th International Conference of Advanced Communication Technology*, volume 2 of *ICACT*, page 6 pp., Phoenix Park, February 2006. IEEE.

[97] Adrian Perrig, John Stankovic, and David Wagner. Security in wireless sensor networks. *Commun. ACM*, 47:53–57, June 2004.

[98] Adrian Perrig, Robert Szewczyk, Victor Wen, David Culler, and J. D. Tygar. SPINS: Security protocols for sensor networks. In *Proceedings of Mobile Computing and Networking*, Rome, Italy, 2001. ACM Press.

[99] Lauri I.W. Pesonen. A capability-based access control architecture for multi-domain publish/subscribe systems. Techreport of Dissertation UCAM-CL-TR-720, University of Cambridge, Cambridge, United Kingdom, June 2007.

[100] L.I.W. Pesonen, D.M. Eyers, and J. Bacon. A capability-based access control architecture for multi-domain publish/subscribe systems. In *International Symposium on Applications and the Internet*, SAINT 2006, pages 222–228, January 2006.

[101] Sebastian Petters, Dirk Thomas, and Oskar von Stryk. Roboframe - a modular software framework for lightweight autonomous robots. In *Proceedings of Workshop on Measures and Procedures for the Evaluation of Robot Architectures and Middleware of the 2007 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, San Diego, CA, USA, October 2007.

[102] Peter R. Pietzuch and Jean M. Bacon. Hermes: a distributed event-based middleware architecture. In *In Proceedings of 22nd International Conference on Distributed Computing Systems*, pages 611–618, 2002.

[103] J. Pollard. Monte carlo methodes for index computation (mod p). *Mathematics of Computation*, 32:918–924, 1978.

[104] J. Pollard. Factoring with cubic integers. In A. Lenstra and H. Lenstra, editors, *The Development of the Number Field Sieve*, pages 4–10. Springer-Verlag, 1993.

[105] Fatemeh Rahimian, Sarunas Girdzijauskas, Amir H. Payberah, and Seif Haridi. Vitis: A gossip-based hybrid overlay for internet-scale publish/subscribe enabling rendezvous routing in unstructured overlay networks. In *Proceedings of the 2011 IEEE International Parallel & Distributed Processing Symposium*, IPDPS '11, pages 746–757, Washington, DC, USA, 2011. IEEE Computer Society.

[106] A. Rahman and E. Dijk. *RFC 7390 – Group Communication for the Constrained Application Protocol (CoAP)*. Internet Engineering Task Force, 1 edition, October 2014. Experimantal Draft.

[107] Redwire LLC. Econotag documentation. Website, January 2016. https://github.com/malvira/econotag.

[108] Volkan Rodoplu and Teresa H. Meng. Minimum energy mobile wireless networks. *IEEE Journal on Selected Areas in Communications*, 17(8):1333–1344, 1999.

[109] Kay Römer. *Handbook of Sensor Networks: Algorithms and Architectures*, chapter 1, page 552. Parallel and Distributed Computing. Wiley & Sons, 1 edition, October 2005.

[110] Kay Römer, Christian Frank, Pedro Jose Marron, and Christian Becker. Generic role assignment for wireless sensor networks. In *Proceedings of the 11th ACM SIGOPS European Workshop*, pages 7–12, Leuven, Belgium, September 2004.

[111] Narayanan Sadagopan, Bhaskar Krishnamachari, and Ahmed Helmy. The acquire mechanism for efficient querying in sensor networks. In *In Proceedings of IEEE International Workshop on Sensor Network Protocols and Applications*, SNPA'03, pages 149–155, 2003.

[112] Alessandro Saffiotti, Mathias Broxvall, Marco Gritti, Kevin LeBlanc, Robert Lundh, Jayedur Rashid, Beom-Su Seo, and Young-Jo Cho. The PEIS-ecology project: vision and results. In *Proc of the IEEE/RSJ Int Conf on Intelligent Robots and Systems*, IROS, pages 2329–2335, Nice, France, 2008.

[113] A. Sahai and B. Waters. Fuzzy identity based encryption. In *Advances in Cryptology – Eurocrypt*, volume 3494 of *Lecture Notes in Computer Science*, pages 457–473. Springer, 2005.

[114] Curt Schurgers and Mani B. Srivastava. Energy efficient routing in wireless sensor networks. In *Proceedings on Communications for Network-Centric Operations: Creating the Information Force*, MILCOM, pages 357–361, McLean, VA, 2001.

[115] Michael Scott and Paulo S. L. M. Barreto. Compressed pairings. In *In Advances in cryptology - Crypto 2004*, pages 140–156. Springer-Verlag, 2004.

[116] Winston K. G. Seah, Zhi Ang Eu, and Hwee pink Tan. Wireless sensor networks powered by ambient energy harvesting (wsn-heap) – survey and challenges. In *1st International Conference on Wireless Communication, Vehicular Technology, Information Theory and Aerospace & Electronic Systems Technology*, Wireless VITAE 2009, pages 1 – 5, Aalborg, May 2009.

[117] A. Shamir. Identity based cryptosystems and signature schemes. In *Advances in Cryptology – CRYPTO*, volume 196 of *Lecture Notes in Computer Science*, pages 37–53. Springer, 1984.

[118] Zach Shelby and Carsten Bormann. *6LoWPAN: The Wireless Embedded Internet*. Wiley Series in Communications Technology. John Wiley & Sons, 1. edition, November 2009.

[119] Zach Shelby, Klaus Hartke, and Carsten Bormann. *RFC 7252 – The Constrained Application Protocol (CoAP)*. Internet Engineering Task Force (IETF), June 2014.

[120] Libelium Comunicaciones Distribuidas S.L. *Waspmote Datasheet*. Libelium Comunicaciones Distribuidas S.L., v4.2 edition, April 2013.

[121] Standards for Efficient Cryptography Group (SECG). Recommended elliptic curve domain parameters. PDF, January 2010. http://www.secg.org/download/aid-784/sec2-v2.pdf.

[122] Andy Stanford-Clark and Hong Linh Truong. *MQTT For Sensor Networks (MQTT-SN) Protocol Specification*. International Business Machines Corporation (IBM), November 2013. Version 1.2.

[123] Jan Steffan, Ludger Fiege, Mariano Cilia, and Alejandro P. Buchmann. Scoping in wireless sensor networks: A position paper. In *Proceedings of the 2nd Workshop on Middleware for Pervasive and Ad-hoc Computing*, pages 167–171. ACM Press, October 2004.

[124] Jan Steffan, Ludger Fiege, Mariano Cilia, and Alejandro P. Buchmann. Towards multi-purpose wireless sensor networks. In *Systems Communications*, ICW / ICHSN / ICMCS / SENET 2005, pages 336–341, Montreal, Canada, August 2005. IEEE Computer Society.

[125] Texas Instruments Inc. *MSP430F241x, MSP430F261x mixed signal microcontroller*, June 2007.

[126] John Thelen and Daan Goense. Radio wave propagation in potato fields. In *In 1st Workshop on Wireless Network Measurements*, 2005.

[127] Gilman Tolle, Joseph Polastre, Robert Szewczyk, David Culler, Neil Turner, Kevin Tu, Stephen Burgess, Todd Dawson, Phil Buonadonna, David Gay, and Wei Hong. A macroscope in the redwoods. In *Proceedings of the 3rd international conference on Embedded networked sensor systems*, SenSys '05, pages 51–63, New York, NY, USA, 2005. ACM.

[128] Lyes Touati, Yacine Challal, and Abdelmadjid Bouabdallah. C-CP-ABE: Cooperative ciphertext policy attribute-based encryption for the internet of things. In *Proceedings of International Conference on Advanced Networking Distributed Systems and Applications*, INDS, pages 64–69. IEEE, June 2014.

[129] Claudio Vairo, Michele Albano, and Stefano Chessa. A secure middleware for wireless sensor networks. In *Proceedings of the 5th Annual International Conference on Mobile and Ubiquitous Systems: Computing, Networking, and Services*, Mobiquitous '08, pages 59:1–59:6, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[130] M. Valero, Sang Shin Jung, A.S. Uluagac, Yingshu Li, and R. Beyah. Di-sec: A distributed security framework for heterogeneous wireless sensor networks. In *Proceedings IEEE INFOCOM*, pages 585–593, Orlando, FL, USA, March 2012. IEEE.

[131] Pascal A. Vicaire. *Physicalnet: A Framework for Interoperability and Application Concurrency in Wireless Networks of Sensors and Actuators*. Phd thesis, School of Engineering and Applied Science, University of Virginia, Charlottesville, USA, December 2008.

[132] Pascal A. Vicaire, Zhiheng Xie, Enamul Hoque, and John A. Stankovic. Physicalnet: A generic framework for managing and programming across pervasive computing networks. In *Proceedings of 16th Real-Time and Embedded Technology and Applications Symposium*, RTAS, pages 269–278. IEEE, April 2010.

[133] Malisa Vucinic, Bernard Tourancheau, Franck Rousseau, Andrzej Duda, Laurent Damon, and Roberto Guizzetti. Oscar: Object security architecture for the internet of things. *Ad Hoc Networks*, 32:3–16, September 2015.

[134] S. Vural, P. Navaratnam, Ning Wang, and R. Tafazolli. Asynchronous clustering of multihop wireless sensor networks. In *Proceedings of International Conference on Communications*, ICC, pages 472–477. IEEE, June 2014.

[135] John Paul Walters, Zhengqiang Liang, Weisong Shi, and Vipin Chaudhary. Wireless sensor networks security: A survey. In Yang Xiao, editor, *Security in Distributed, Grid, and Pervasive Computing*. Auerbach Publications, CRC Press, 2006.

[136] Stefan Georg Weber, Yulian Kalev, Sebastian Ries, and Max Mühlhäuser. Mundomessage: Enabling trustworthy ubiquitous emergency communication. In *Proceedings of ACM International Conference on Ubiquitous Information Management and Communication*, ICUIMC 2011. ACM Press, 2011.

[137] Matt Welsh. Exposing resource tradeoffs in region-based communication abstractions for sensor networks. In *Proceedings of the 2nd ACM Workshop on Hot Topics in Networks*, HotNets-II, November 2003.

[138] Matt Welsh and Geoff Mainland. Programming sensor networks using abstract regions. In *Proceedings of the first USENIX/ACM Symposium on Networked Systems Design and Implementation*, NSDI '04, pages 29–42, March 2004.

[139] Erich Wenger and Paul Wolfger. Harder, better, faster, stronger - elliptic curve discrete logarithm computations on fpgas. Cryptology ePrint Archive, Report 2015/143, 2015. `http://eprint.iacr.org/`.

[140] G. Werner-Allen, K. Lorincz, M. Ruiz, O. Marcillo, J. Johnson, J. Lees, and M. Welsh. Deploying a wireless sensor network on an active volcano. *Internet Computing, IEEE*, 10(2):18 – 25, march-april 2006.

[141] Geoffrey Werner-Allen, Patrick Swieskowski, and Matt Welsh. Motelab: a wireless sensor network testbed. In *Proceedings of the 4th international symposium on Information processing in sensor networks*, IPSN '05, Piscataway, NJ, USA, 2005. IEEE Press.

[142] Kamin Whitehouse, Cory Sharp, Eric Brewer, and David Culler. Hood: A neighborhood abstraction for sensor networks. In *Proceedings of the 2nd international conference on Mobile systems, applications and services*, MobiSYS '04, pages 99–100, Boston, Massachusetts, USA, June 2004. ACM Press.

[143] Xiaokang Xiong, Duncan S. Wong, and Xiaotie Deng. Tinypairing: Computing tate pairing on sensor nodes with higher speed and less memory. In *Proceedings*

*of the 2009 Eighth IEEE International Symposium on Network Computing and Applications*, NCA '09, pages 187–194, Washington, DC, USA, 2009. IEEE Computer Society.

[144] Xiaokang Xiong, Duncan S. Wong, and Xiaotie Deng. Tinypairing: A fast and lightweight pairing-based cryptographic library for wireless sensor networks. In *IEEE Wireless Communications & Networking Conference*, IEEE WCNC10, Sydney, Australia, April 2010.

[145] Ya Xu, John Heidemann, and Deborah Estrin. Geography-informed energy conservation for ad hoc routing. In *Proceedings of the 7th annual international conference on Mobile computing and networking*, MobiCom '01, pages 70–84, New York, NY, USA, 2001. ACM.

[146] Li Ya, Wang Pengjun, Luo Rong, Yang Huazhong, and Liu Wei. Reliable energy-aware routing protocol for heterogeneous wsn based on beaconing. In *Proceedings of 16th International Conference on Advanced Communication Technology*, ICACT, pages 109–112. IEEE, February 2014.

[147] Yong Yao and Johannes Gehrke. The cougar approach to in-network query processing in sensor networks. *ACM SIGMOD Record*, 31(3):9–18, September 2002.

[148] M. Younis, M. Youssef, and K. Arisha. Energy-aware routing in cluster-based sensor networks. In *Proceedings of the 10th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems*, MASCOTS '02, Washington, DC, USA, 2002. IEEE Computer Society.

[149] Shucheng Yu, Kui Ren, and Wenjing Lou. Attribute-based on-demand multicast group setup with membership anonymity. In *Proceedings of the 4th international conference on Security and privacy in communication netowrks*, SecureComm '08, pages 18:1–18:6, New York, NY, USA, 2008. ACM.

[150] Shucheng Yu, Kui Ren, and Wenjing Lou. FDAC: Toward fine-grained distributed data access control in wireless sensor networks. *IEEE Transactions on Parallel and Distributed Systems*, 22:673–686, 2011.

[151] Yan Yu, Ramesh Govindan, and Deborah Estrin. Geographical and energy aware routing: A recursive data dissemination protocol for wireless sensor networks. Technical Report UCLA-CSD TR-01-0023, UCLA Computer Science Department, May 2001.

[152] Yang Yu, Loren J. Rittle, Vartika Bhandari, and Jason B. LeBrun. Supporting concurrent applications in wireless sensor networks. In *Proceedings of the 4th international Conference on Embedded Networked Sensor Systems*, pages 139–152, New York, NY, USA, November 2006. ACM.

[153] Sencun Zhu, Sanjeev Setia, and Sushil Jajodia. LEAP: Efficient security mechanisms for large-scale distributed sensor networks. Techreport, Center for Secure Information Systems, George Mason University, Fairfax, VA 22030, August 2004.

[154] Zolertia. *Z1 Datasheet*. Zolertia, Barcelona, Spain, 1.1 edition, March 2010.