Deliverable D5.3

Laboratory Evaluation Results

Editor:	Georgios Papadopoulos, UNIVBRIS George Oikonomou, UNIVBRIS
Deliverable nature:	Report (R)
Dissemination level: (Confidentiality)	Public (PU)
Contractual delivery date:	29 Feb 2016
Actual delivery date:	6 March 2016
Suggested readers:	Researchers, IERC, application developers, system administrators
Version:	1.0
Total number of pages:	104
Keywords:	Internet of Things, Reliability, Availability, Compressive Sensing, Heterogeneous Networks, Matrix Completion, Spectrum Sensing, Datagram Transport Layer Security, Elliptic Curve Cryptography, Leakage-Resilient Message Authentication Codes, 6LoWPAN Multicast Forwarding, Cognitive Radio, Traffic Monitoring

Abstract

This report presents the results of the work undertaken as part of Task 5.3 "Proof-of-Concept Laboratory Experiments". The aim of the in-lab experiments is to conduct *proof-of-concept* controlled tests to assess both qualitatively and quantitatively the performance gains of the protocols and algorithms, as well as the individual system modules developed in work packages WP2-WP4. The results of these lab tests will be used to identify potential issues and try to resolve them by revising the software or the hardware modules. This feedback cycle will result to the improvement of the components tested, and these will be applied in the first phase of the live trials in the pilot cities. For the performance evaluation activities, we followed the evaluation methodology defined in Deliverable D5.1. Moreover, this deliverable received the developed software and hardware components from T5.2. The results from the in-lab experiments will enhance the components that will be integrated in task 5.2 for the trials performed in two cities, in task 5.4 for Heraklion and in in task 5.5 for Tarragona.

Disclaimer

The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 609094.

Impressum

Full project title	Reliable, resilient and secure IoT for smart city applications
Short project title	RERUM
Number and title of work-package	WP5 - Application Development, Experiments and Trials
Number and title of task	T5.3 – Proof-of-Concept Laboratory Experiments
Document title	Laboratory Evaluation Results
Editor: Name, company	Georgios Papadopoulos, UNIVBRIS George Oikonomou, UNIVBRIS
Work-package leader: Name, company	Ricard Munné, Atos
Estimation of person months (PMs) spent on the Deliverable	

Executive summary

This deliverable presents the results of the indoor experiments that were undertaken in order to evaluate the separate mechanisms developed within the RERUM project. The goal of these mechanisms is to improve the security, privacy, network reliability and energy efficiency of IoT systems. RERUM had put a significant emphasis on the hardware devices that are fundamental elements of any IoT system and aimed to develop mechanisms that would be able to run flawlessly on those constrained devices. In the technical workpackages of the RERUM project, the developed mechanisms were evaluated either using simulations or analytically. However, the road from simulations to real-world trials is very long and none can be sure that the developed mechanisms will be able to perform perfectly in a real environment. For this reason, within RERUM we defined an intermediate step for testing the mechanisms and the hardware in controlled environments within laboratories. That way, we could identify early misbehaviours, malfunctions, bugs and issues that would be very easily fixed in such environments. Only after the mechanisms are evaluated positively in the experiments they will get the approval to be tested in the real-world trials.

Thus, in the experiments, the objective is to qualitatively and quantitatively assess the performance gains of the protocols, algorithms and hardware developed within WP2-WP4. Considering the nature of each mechanism to be tested, several experiments were identified in RERUM Deliverable D5.1 [RD5.1] and these were split into three main categories: (i) security and privacy on sensor platforms, (ii) network reliability and (ii) privacy on android smartphones. In all experiments, the tested mechanisms were evaluated not only for their performance, but also for their energy consumption and resources (CPU/memory) consumption to ensure a flawless operation in real world trials.

As it can be seen form the experiments, all developed mechanisms give very promising results regarding both their performance and their lightweight operation. The experiments basically showed that, contrary to previous impressions, it is not infeasible to apply advanced security and privacy preserving mechanisms on constrained IoT devices. If the mechanisms are designed to be lightweight, they can have a great performance, while at the same time consume very little resources and energy. The stability of the developed software during all tests was also noticed in the experiments. Furthermore, the excellent performance of the developed hardware (the Zolertia RE-Mote platform) was a very interesting result, showing that it can be a key platform for advanced IoT applications.

Summarizing, the experimental results presented in this deliverable are very promising for the real-world trials that have already started and will run until the end of the project. The experiments helped to identify minor flaws and bugs in both software and hardware, which have already been fixed and thus no similar issues are expected to be found in the trials.

List of Authors

Company	Author	Contribution
UNIVBRIS	Georgios Papadopoulos George Oikonomou Marcin Wójcik	6LoWPAN Multicast Lightweight DTLS MS NUMS LR-MAC
LiU	David Gundlegård Vangelis Angelakis Sesanka Katuri	Android app power drain Android app CPU Server side scalability Android app stability SNR measurement precision
UNI PASSAU	Johannes Bauer Benedikt Petschkuhn Henrich C. Pöhls Ralf C. Staudemeyer	Overheads of Signing and Verifying with ECC Energy Efficiency of ECC-based Payload Signatures Overheads of Signing and Verifying with Malleable Signatures Energy Efficiency of Malleable Signatures
FORTH	Alexandros Fragkiadakis Elias Tragos Pavlos Charalampidis George Stamatakis Manolis Surligas Antonis Makrogiannakis	RSSI-based CS encryption keys Adaptive CS-based data gathering Sensor self-monitoring Lightweight spectrum sensing CR-based gateway

Table of Contents

List of Figures

List of Tables

Abbreviations

6LoWPAN	IPv6 over Low power Wireless Personal Area Networks
ABER	Average Bit Error Rate
ACE	Authentication and Authorization for Constrained Environments (IETF WG)
AES	Advanced Encryption Standard
ANPE	Average Number of Path Estimations
ASNR	Average Path Signal-to-Noise Ratio
AWGN	Additive White Gaussian Noise
API	Application Programming Interface
BMFA	Bi-Directional Multicast Forwarding Algorithm
CADC	Congestion-Aware Duty Cycling
CBR	Constant Bit-Rate
CCI	Channel Check Interval
CCR	Channel Check Rate
CDF	Cumulative Distribution Function
CoAP	Constrained Application Protocol
CPM	Change Point Method
CR	Compression Rate
CS	Compressive Sensing (or Compressed Sensing)
CSI	Channel State Information
CSMA	Carrier Sense Multiple Access
DAG	Directed Acyclic Graph
DBPSK	Differential Binary Phase Shift Keying
DES	Data Encryption Standard
DODAG	Destination-Oriented Directed Acyclic Graph
DSA	Digital Signature Algorithm
DSS	Digital Signature Standard
DTLS	Datagram TLS
DTMC	Discrete Time Markov Chain
ECC	Elliptic Curve Cryptography
ECDSA	Elliptic Curve Digital Signature Algorithm
EtED	End-to-End Delay (EtED)
ETX	Expected Transmission Count
FWHT	Fast Walsh-Hadamard Transform
GPIO	General-Purpose Input Output
HBHO	Hop-By-Hop Option
I2C	Inter-Integrated Circuit
ICMPv6	Internet Control Message Protocol version 6
IEEE	Institute of Electrical and Electronics Engineers
IERC	Internet of Things European Research Cluster
IETF	Internet Engineering Task Force
IoT	Internet of Things
ISM	Industrial Scientific Medical
JSON	JavaScript Object Notation
JSS	JSON Signature Scheme

KS	Kolmogorov-Smirnov
LiPo	Lithium Polymer Batteries
LPM	Low-Power Mode
MAC	Medium Access Control
MC	Matrix Completion
MGF	Moment Generating Function
MOFSET	Metal-Oxide Transistor
MPL	Multicast Protocol for Low power and Lossy Networks
MPR	Multiple Packet Reception
ND	Network Density
OF	Objective Function
OMP	Orthogonal Matching Pursuit
OP	Outage Probability
OS	Operating System
OSC	Oscillator
PDF	Probability Density Function
PDR	Packet Delivery Ratio
PKCS	Public Key Cryptographie Standard
PLL	Phase-Locked Loop
PM	Power Mode
QoS	Quality of Service
RD	RERUM Device
RDC	Radio Duty Cycling
RFC	Request For Comments
RIP	Restricted Isometry Property
RPL	IPv6 Routing Protocol for Low Power and Lossy Networks
RX	Reception
SD	Selection Diversity
SEC	Switch and Examine Combining
SHA	Secure Hash Algorithm
SINR	Signal to Interference-plus-Noise Ratio
SMRF	Stateless Multicast RPL Forwarding
SNR	Signal to Noise Ratio
SP	Switching Probability
SPI	Serial Peripheral Interface
SRM	Structurally Random Matrix
SSC	Switch and Stay Combining
TLS	Transport Layer Security
TM	Trickle Multicast
TX	Transmission
UART	Universal Asynchronous Receiver Transmitter
UDGM	Unit Disk Graph Medium
VBR	Variable Bit Rate
WDT	Watchdog Timer
WFI	Wait For Interrupt
WG	Work Group

WSN	Wireless Sensor Network
XOSC	Crystal Oscillator

Definitions

Term	Definition	Source
Acting element	An (embedded) device that has the capability to affect the condition of a Physical Entity, (like changing its state or moving it) by acting upon an electrical signal	RERUM/ IoT-A part of actuator [IOTA]
Actuator	A smart device that includes one or several acting elements and receives (IT-based) commands translating them to electrical signals for the acting elements. An actuator can also include a sensor so that there is knowledge on the Physical Entity it acts upon, in order to translate correctly the command into the electrical signal.	RERUM/ IoT-A
Application server	The point responsible for the end-user services (e.g., automation services, energy management, etc.). The Application server may reside either in the internet or in the RERUM domain and is responsible for accepting dynamic resource requests, executing the appropriate actions, and returning the results to the user.	RERUM/ IoT-A
Context	Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.	[AG99]
Cluster	A group of wireless (mainly sensor) nodes that work together for a more efficient and scalable organisation and management of the network.	RERUM, based on [AY07]
Cluster Head (CH)	The RERUM Device that plays the role of the Head of a Cluster within the RERUM network. The CH is responsible for routing the data from the members of the cluster to the rest of the network, as well as to take centralized networking decisions. The CH is either pre-assigned or can be selected by the RERUM Devices.	RERUM, based on [AY07]
Clustering	The process of splitting the network in clusters and electing CHs.	RERUM, based on [AY07]
Consent	Within RERUM the user consent is used for privacy purposes, when the system will ask the user if he allows to send his data to an application that requests them.	RERUM
Device	It can be a single or a combination of the following elements: • Sensors, which provide information about a Physical Entity • Tags, which are used to identify Physical Entities	IoT-A

	• Actuators, which can modify the physical state of a Physical Entity	
Federation Head (FH)	A functional component that executes the process of the Federation of VRDs. It can be assigned to any powerful RD, the GW or a centralized server.	RERUM
Federation of Virtual RERUM Devices	Several Virtual RERUM Devices are forming a Federation if they cooperate to offer a joint service for a Virtual Entity (VE). The logic necessary to orchestrate the service is associated to the Virtual Entity that offers the service.	RERUM
Gateway (GW)	Network node equipped for interfacing with another network that uses different protocols.	Federal Standard 1037C [SF96]
Generic Virtual RERUM Object (GVO)	This is a software artefact that groups both virtualisations found in RERUM, namely the Virtual Entities and Virtual RERUM Devices that share properties like, that • they allow to be discovered, • they allow to be addressed, and they allow to be interacted with in a standardized manner.	RERUM
Internet Resources	• These are sources of data/measurements that originate from outside of the RERUM domain and can be used as input for the applications.	RERUM
RERUM Middleware (MW)	• Within RERUM, the Middleware is assumed to be a software layer or a group of functionalities that allows heterogeneous devices to be discovered, addressed and accessed by the applications in a seamless and unified way. The Middleware includes the virtualisation of devices to hide their heterogeneity.	RERUM
Physical Entity (PE)	• A discrete, identifiable part of the physical environment which is of interest to the user for the completion of his goal. Physical Entities can be almost any object or environment.	Merriam-Webster dictionary[1]/ IOT-A
RERUM Aggregator	A RERUM Device can play the role of an Aggregator, when it collects, processes (aggregates, encrypts, filters, etc.) data/measurements from many other RERUM Devices and forwards them to the GW/Middleware/Application Server. A RERUM aggregator can be considered as an RD playing the role of a Federation Head and could be very helpful in terms of privacy, because this aggregation will avoid the leaking of	RERUM

[1] Merriam-Webster Online: Dictionary and Thesaurus www.merriam-webster.com

	personal information that may be contained in the data that are aggregated.	
RERUM Device (RD) or RERUM Smart Object	A RERUM Device (RD) is a piece of hardware and software (incl. the Operating System) that is equipped with intelligence. It has one or more Resources that the RERUM Device is able to either fill with interpreted and pre-processed sensory data or able to read and interpret the commands that are given. The RERUM Device has some Sensing, Tag or Acting elements directly attached to it.	RERUM
RERUM Deployment	The specific topology of software components on the physical layer, as well as the physical connections between these components.	IoT-A / RERUM
RERUM Gateway	A RERUM Gateway is a physical device that plays the role of a network gateway interconnecting different RERUM networks. Furthermore, the RERUM Gateway is responsible for managing the RDs that are connected to it. In this respect it can also include various Middleware functionalities.	RERUM
Resources	Resources are software components that provide some functionality. When associated with a Physical Entity, they either provide some information about or allow changing some aspects in the digital or physical world pertaining to one or more Physical Entities. In general, they are typically sensor Resources that provide sensing data or actuator Resources, e.g. a machine controller that effects some actuation in the physical world.	IOT-A On-device Resources
Sensing element	An (embedded) device that perceives certain characteristics of the real-world environment (Physical Entities), translating a change into an electrical signal.	RERUM
Sensor	• A smart device that includes one or several sensing elements and is able to translate the electrical signal of the sensing elements to some type of information (digital representation) with specific value and semantic.	IoT-A
(IoT/RERUM) Service	• Software component enabling interaction with resources through a well-defined interface, often via the Internet.	IoT-A, RERUM
Smart Object	See RERUM Device	RERUM
Virtual Entity (VE)	The digital synchronized representation of a Physical Entity.	IoT-A
Virtual RERUM	A Virtual RERUM Device (RD) is a digital representation of a RERUM Device. The same one physical RERUM Device at one time is represented by one Virtual RERUM Device. This is a	RERUM

Device (VRD)	software artefact, like a Virtual Entity (VE), but represents a RERUM Device (RD).	
User	A Human or a software that interacts with a system for transferring information.	Based on IoT-A

1 Introduction

1.1 Objectives of this document

This document presents the indoor experimental results of the RERUM project [RERUM] for evaluating the security, networking and scalability of the system in controlled laboratory environments running small experiments.

More specifically, this deliverable qualitatively and quantitatively assesses the performance gains of the protocols and algorithms that are developed within WP2-WP4. To this aim, the evaluation results on conducted proof-of-concept controlled lab-experiments are presented. The results of these in-lab tests will be used to improve the components tested, and the conclusions will be applied in the first phase of the live trials in the pilot cities. In parallel to the first phase, the lab experiments will continue to improve those components with some performance issues, and improvements will be provided in M30 just before the start of the second phase of the live trials.

1.2 Intended audience

This document is intended primarily for the RERUM partners, namely the researchers and developers that are involved in the technical work packages to conduct the proof-of-concept experiments and trials, and the pilot cities that will execute the use case trials. However, we believe that the framework can be of interest for researchers and smart city services developers, and the outcomes from the lab experiments and the trials will certainly be of interest for a wider audience, as they will demonstrate the feasibility of the RERUM architecture applied to a live smart city environment.

1.3 Position within the project

1.3.1 Relation with other tasks and WPs

This deliverable (D5.3) presents experimental performance evaluation of the protocols, algorithms and individual system modules that are developed within WP2-WP4 (see Figure 1). The evaluation methodology defined in Task 5.1 is used for the performance evaluation activities. Moreover, it received the developed software and hardware components from T5.2.

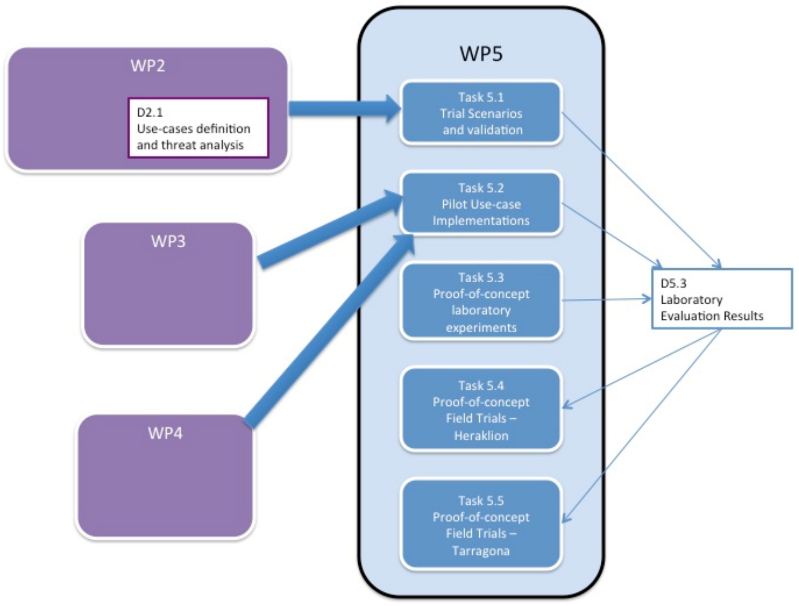

WP2
D2.1
Use-cases definition and threat analysis
WP3
WP4
WP5
Task 5.1
Trial Scenarios and validation
Task 5.2
Pilot Use-case Implementations
Task 5.3
Proof-of-concept laboratory experiments
Task 5.4
Proof-of-concept Field Trials – Heraklion
Task 5.5
Proof-of-concept Field Trials – Tarragona
D5.3
Laboratory Evaluation Results

Figure 1: Position of T5.3/D5.3 within RERUM.

The outputs of the experimental campaign of D5.3 aim at evaluating the performance of the individual system modules in order to identify any potential issues, and to prepare them for the real-world trials in tasks T5.4 and T5.5.

1.4 Document structure

This deliverable is structured as follows:

- Section 2 demonstrates the security lab trials on RERUM Devices (RDs). More precisely, Subsection 2.1 presents the runtime-, memory- and communication-overhead of signing and verifying the message payload with ECC standard signatures. Subsequently, subsection 2.2 evaluates the energy efficiency of ECC based payload signatures on RDs. Subsection 2.3 reports on, the runtime-, memory- and communication-overhead of signing, verifying and messages with malleable signatures, while subsection 2.4, evaluates the energy efficiency of malleable signatures. Subsection 2.5 demonstrates the handshake time for Lightweight Datagram Transport Layer Security (DTLS). Furthermore, Subsection 2.6 evaluates the computation time of the MS NUMS elliptic curves, while in Subsection 2.7 the performance evaluation of a practical Leakage-Resilient Message Authentication Code (MAC) scheme is shown. Finally, in Subsection 2.8 the evaluation of the RSSI-based Compressed Sensing (CS) encryption keys is presented.
- Section 3 shifts focus on the evaluation of networking lab trials. In particular, the section presents results on the performance of: i) the adaptive CS-based data gathering (Sec. 3.1), ii) the sensor self-monitoring component (Sec. 3.2), iii) the lightweight spectrum sensing component (Sec. 3.3), iv) the Cognitive Radio (CR)-based gateway (Sec. 3.4) and v) the BMFA multicast forwarding algorithm for 6LoWPANs (Sec. 3.5).
- Section 4 discusses lab trial testing of the traffic monitoring Use-Case. More specifically, in 4.1 the results of the experimental evaluation of the battery drain of the participatory sensing android app are presented, while in 4.2 the evaluation of the participatory sensing app in terms of CPU usage is demonstrated. Subsection 4.3 evaluates the scalability of the server side implementation of the smart transportation use case. Furthermore, in 4.4 demonstrates the experimental evaluation how the stability of the android app is affected by the sampling period. Finally, Subsection 4.5 explores the usage of the SNR measurement precision for the RERUM android application.
- Section 5 concludes the deliverable with an overview of the key results.

2 Security Lab Trials

This section documents experiments that were undertaken in order to assess the cryptographic capability of RERUM devices and their suitability for RERUM-devised security mechanisms.

2.1 Overheads of signing and verifying with ECC

In this section we investigate the feasibility of the RERUM approach to bring cryptographically strong digital signature capabilities into smart devices. Encouraging results indicate that we can have end-to-end authenticity and integrity protection even at sensor level in the IoT.

We use the term “standard signatures” to refer to classic signature schemes based on Public Key Cryptography Standards (PKCS) [RSA12] or the Digital Signature Algorithm (DSA) [DSS13]. These involve a standard cryptographic hash function (SHA256 [SHA2], or better) and operations of an asymmetric cryptographic primitive, like Elliptic Curve Cryptography (ECC) as those are better suited for constrained devices than RSA-based algorithms. Therefore, this section focuses on experiments that target the evaluation of Runtime-, Memory-, Communication-overhead of Signing and Verifying Message Payloads with ECC signatures.

To generate ECC based signatures requires, as with all asymmetric based digital signature schemes, two distinct keys, one called secret key for signature generation and another one, called public key, for signature verification. The secret key must be generated, stored, and used, such that the confidentiality is not violated at any time. RERUM assumes (see [RD3.1]) that this is a per device key, so that key extraction key, would give the attacker only the key to impersonate that single device. For the tests conducted in the laboratory experiments at UNI PASSAU, we did not address the key distribution problem. For now we assumed that the key-pair was generated on the RE-Mote and the corresponding public key was transported to the communication partner via a pre-established authenticated / integrity protected channel.

For public key transport it is important that the device receiving the RE-Mote's new public key needs to have gained assurance that the public key belongs to a certain authentic RE-Mote. This can be addressed at some earlier stage or during the communication. We assume this being done as part of the secure credential bootstrapping [RD3.1].

After a successful key set-up, we assume the following:

- RE-Mote *A* possesses its own secret signature generation key.
- RE-Mote *A* possesses both public signature verification keys: Its own and the verification key of B.
- Device *B* possesses its own secret signature generation key.
- Device *B* possesses both public signature verification keys: Its own and the verification key of RE-Mote A.

2.1.1 Implementation details

An initial manual search for existing crypto libraries for Contiki and suitable to run on an ARM Cortex-M3 platform was performed in July 2015. The following libraries were selected as candidates for use on the RE-Mote and further investigation:

1. TweetNaCl library (Curve25519 and Ed25519) [BGJLSS15]
2. Implementation by Piñol Oriol (Secp256r1) [P14]
3. MicroECC library (Secp160r1, Secp192r1, Secp224r1, Secp256k1, Secp256r1) [mECC]

We ported these libraries to Contiki to run on the RE-Mote [M16]. The ports still lack any RE-Mote specific acceleration, like use of TI's crypto co-processor. Therefore, there is potential for improvement. The results obtained measuring the performance and overhead of those libraries are presented in the next section.

The more common curves suggested by the National Institute of Standards and Technology (NIST) have been severely criticised [BCCHLN14]. As it might be undesired the use insecure curves, we suggest to use curves like Curve25519 and the signature algorithms Ed25559 [BGJLSS15] at later stages. Just recently those curves have started being considered for standardisation in RFC7748 [RFC7748] and this may lead to their more widespread adoption. In RERUM, we therefore ported and benchmarked the Curve25519 on the Zolertia Re-Mote device [M16], which regarding the general cryptographic design is close to the ones suggested in RFC7748.

Therefore, the chosen implementations that have been ported to Contiki and tested on the RE-Mote are Curve25519 and Ed25519 provided by the TweetNaCl-library, Secp256r1 by Piñol Oriol (Secp256r1) and ECDSA on five standard curves provided by the MicroECC library. Each implementation provides a number of different parameters which can be adjusted to improve performance. The specific adjustments for each implementation, also the different options and their adjustments are as follows:

- **Curve25519 and Ed25519 – TweetNaCl**: TweetNaCl was implemented by Daniel J. Bernstein et.al.. It is a crypto library that supports ECDSA based on Curve25519 and Ed25519 [BGJLSS15]. The library is generic, but some of the design choices target 32bit machines. The library does not need any dynamic memory allocation nor has other library dependencies. Another aspect is that the implementation is side channel attack resistant [BGJLSS15, L15b].
- **ecp256r1 Piñol Oriol**: Piñol's implementation permits the use of different coordinate systems, curve parameters and architectures. The following options are available for coordinate representation:
    - affine coordinates,
    - homogeneous coordinates, and
    - jacobian coordinates.

  The reason behind using different coordinate systems is that a costly operation in one coordinate system may be a low-cost operation in another. Piñol gained best performance by using a homogeneous coordination system. This implementation supports the use of a sliding window, instead of a fixed window size, for point doubling computation. Another feature of this implementation is the use of a different word size for point representation; therefore, it is possible to configure the library for both 16-bit and 32-bit word lengths [P14].
- **MicroECC**: MicroECC implements ECDH and ECDSA on five standard curves. It has a great flexibility and it can be adjusted to use inline assembly (ASM) for ARM platforms. The library is implemented in C and supports the following five standard curves: Secp160r1, Secp192r1, Secp224r1, Secp256k1 and Secp256r1. The internal computation is safe against timing attacks. The MicroECC library can optionally be optimized for either speed or code size.

2.1.2 Experimental setup

The implementation was realized on the Zolertia RE-Mote device with an integrated microcontroller from Texas Instruments. Our experiments run on top of Contiki OS configured with a disabled network stack. This was necessary to eliminate interrupts and other side effects that might interfere with time overhead measurements. Each implementation used its own configuration with implementation-specific arguments.

2.1.2.1 Curve specific test variants and challenges overcome

- **Curve25519 and Ed25519 – TweetNaCl**: The implementation of Ed25519 uses Sha512 to generate message hashes. The TweetNaCl implementation created wrong keys and signatures due to stack overflows. To fix this issue, Contiki's stack size was increased from 2048 to 4096 bytes. To meet the correct calculation behaviour, the type definitions used by TweetNaCl were adjusted to unsigned integers (8-, 32-, and 64-bit).
- **Secp256r1 - Piñol Oriol** was originally designed for Contiki. We tested with the all three different coordinate systems [P14]. The implementation uses Sha256 to hash messages. In addition, differences between 16-bit and 32-bit calculations were evaluated.
- **MicroECC** was adapted to Contiki by us. It excludes hashing, which is therefore not part of the measurement (we applied Sha256). For each of the curves there are three ARM configurations, that use different inline assembly code optimizations:
    - no assembler code,
    - assembler code optimized for small size, and
    - assembler code optimized for fast execution.

2.1.2.2 ECDSA adjustments

For ECDSA, the several processing steps are grouped into the following three main tasks:

- key generation (key-gen)
- signature generation (sign)
- signature verification (verify)

In the key generation and signing process, the pseudo number generator (PRNG) is replaced with an empty function to avoid influences in the test. For all curves, the used hash function is included into the measurement, but its timing is explicitly reported for each task. For reasons of comparison we set the length of the message to sign to 15 bytes, although the influence of the hash function is minimal (in the region of a few ms).

2.1.2.3 Memory-overhead computation

To compute the static code size of the project, all the dependencies to the crypto libraries were first removed. Afterwards the Executable and Linkable Format (elf) file was stripped and the resulting size measured. For measuring, we used the overhead tools from the existing gcc-arm-none-eabi toolchain. Finally we compared the measured size to the original project size. The elf file represents a compiled and already linked executable, which can be flashed on the RE-Mote device.

2.1.2.4 Runtime computation with timelib

We developed our own tool called timelib for measuring the timing. The timelib-tool consists of multiple useful functions to automate the measurement process and standardize the output for other tools. In timing mode, the RE-Motes internal timer prints the time spent for each task to standard output (STDOUT). In the power-trace mode, the start/stop points are marked by activating the LEDs for the corresponding task and glow for 35ms at the beginning and end of each task. This is recognizable on the power-trace. It is configurable at compile time.

For runtime measurements, we used the internal real-time clock of the CC2538 chip, which runs at 32.768 kHz [CC2538]. The developed timelib-tool analyses the output and computes the average over several iterations to the corresponding diagram as shown in Figure 2:

```
---------------------------- TimeLibTasks ----------------------------
occurrence: 24 time 74508 ms avg: 3104 ms |hash| |crypto_sign_keypair|
occurrence: 24 time 84447498 ms avg: 3518645 ms |crypto_sign_keypair|
occurrence: 24 time 74599 ms avg: 3108 ms |hash| |crypto_sign|
occurrence: 24 time 84782398 ms avg: 3532599 ms |crypto_sign|
occurrence: 24 time 75823 ms avg: 3159 ms |hash| |crypto_sign_open|
occurrence: 23 time 162098438 ms avg: 7047758 ms |crypto_sign_open|
occurrence: 23 time 47779466 ms avg: 2077368 ms |alice key generation|
occurrence: 23 time 47778094 ms avg: 2077308 ms |bob shared key
    generation|
occurrence: 23 time 47779562 ms avg: 2077372 ms |bob key generation|
occurrence: 23 time 47780167 ms avg: 2077398 ms |alice shared key
    generation|
---------------------------- ------------ ----------------------------
```

Figure 2: Timelib aggregation output.

We observed each implementation over a period of 10 minutes performing the measurements multiple times for each step of the signing process. The time for each task is aggregated and the average is taken.

2.1.3 KPIs

For ECDSA with TweetNaCl, with Secp256r1 by Piñol, with MicroECC, and as well with MicroECC on the RE-Mote with Contiki, we consider the following evaluation criteria defined in Deliverable D5.1 [RD5.1]:

- Crypto-Runtime-Overhead (AL.PE.3), and
- Crypto-Memory-Consumption-Overhead (AL.EF.3)

For MicroECC on Contiki we also evaluate criterion Crypto-Communication-Overhead (AL.EF.4).

2.1.4 Performance evaluation

Here, we illustrate and compare the measured results for the implementations TweetNaCl, sep256r1 and MircoECC, in time and size.

2.1.4.1 Runtime and memory overhead of ECDSA with TweetNaCl

TweetNaCl accomplished the key generation in 3.518s, signature generation in 3.532s, and verification in 7.047s, as shown in Table 1.

Table 1: TweetNaCl ECDSA timing results.

implementation	keygen	sign	verify
TweetNaCl	3.518 s	3.532 s	7.047 s

Figure 3 shows the graphical presentation of the timing values of different operations measured.

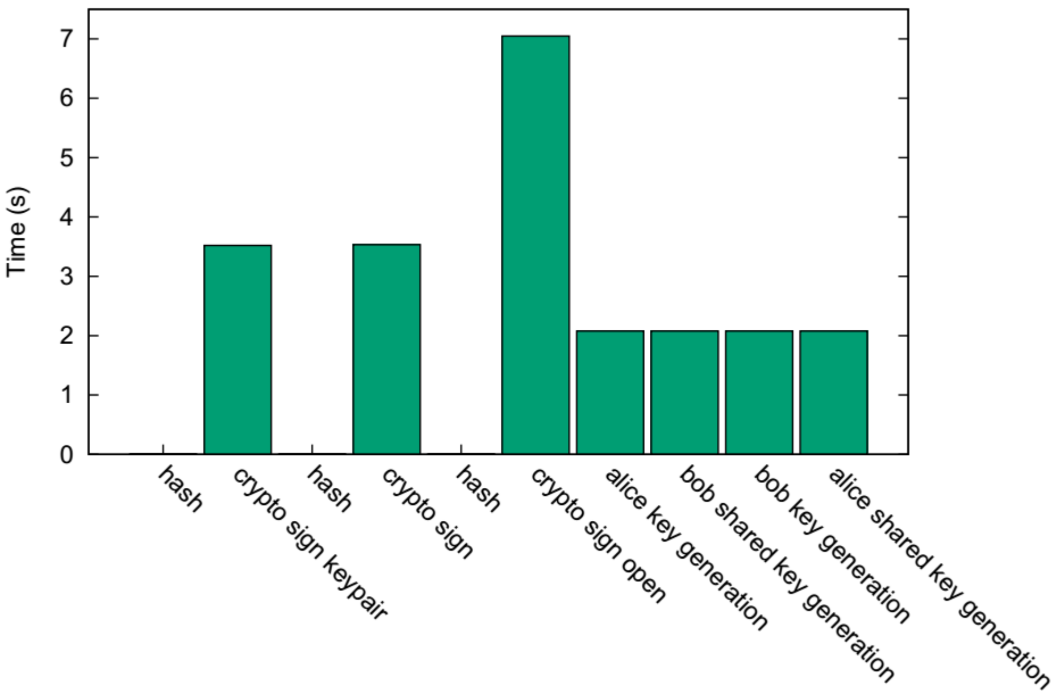


Figure 3: Average time for TweetNaCl over 10 minutes [M16].

The size of the TweetNaCl implementation is shown in Table 2. TweetNaCl supports ECDH and ECDSA.

Table 2: TweetNaCl static code size results.

Configuration	static code size
TweetNaCl	6776 bytes

2.1.4.2 Runtime and memory overhead of ECDSA with Secp256r1 by Piñol

The results of Oriol Piñol documented in [P14] could be confirmed. The best performing configuration was with the Jacobian coordinate system without sliding window in 32-bit mode. This implementation was tested with twelve configuration permutations (three possible coordination systems, with or without sliding windows, using 16- or 32bit calculations). In all variants the differences in the configuration with and without sliding window are either negligible or a disadvantage of 100ms in average. Also on the RE-Mote the 32-bit variant should be preferred because it achieves the best results in terms of runtime with ECDSA. Noteable is the difference in runtime between shared key generation and public key generation, which is also visible in the power trace.

Time measurements indicate that for ECDSA the configuration with Jacobian coordinates, 32-bit words, and without sliding window is also the fastest with this implementation. For further details, see Table 3.

There is no difference in the static code size of the implementations between with and without a sliding window, but the smallest size could be accomplished by configuring the affine coordinate system, on 32-bit at the expense of speed. The fastest configuration has the largest static code size with a difference of 1418 bytes in comparison to the smallest. All static code sizes measured are presented in Table 4.

Table 3: Piñol ECDSA timing comparison.

configuration	keygen	sign	verify
affine 16 bit	45.871 s	47.098 s	93.647 s
affine 16 bit sliding window	46.287 s	47.086 s	90.900 s
affine 32 bit	26.644 s	26.420 s	52.977 s
affine 32 bit sliding window	26.712 s	26.503 s	52.832 s
homogenous 16 bit	29.797 s	30.813 s	60.203 s
homogenous 16 bit sliding window	29.586 s	29.804 s	57.981 s
homogenous 32 bit	10.829 s	10.820 s	21.449 s
homogenous 32 bit sliding window	10.842 s	10.858 s	21.798 s
jacobian 16 bit	18.294 s	18.270 s	36.476 s
jacobian 16 bit sliding window	18.224 s	18.184 s	36.333 s
jacobian 32 bit	6.425 s	6.439 s	13.075 s
jacobian 32 bit sliding window	6.521 s	6.588 s	13.175 s

Table 4: Piñol size comparison.

configuration	static code size
affine 16 bit	6899 byte
affine 16 bit sliding window	6899 byte
affine 32 bit	6915 byte
affine 32 bit sliding window	6915 byte
homogenous 16 bit	8153 byte
homogenous 16 bit sliding window	8153 byte
homogenous 32 bit	8153 byte
homogenous 32 bit sliding window	8153 byte
jacobian 16 bit	8333 byte
jacobian 16 bit sliding window	8333 byte
jacobian 32 bit	8317 byte
jacobian 32 bit sliding window	8317 byte

2.1.4.3 Runtime and memory overhead of ECDSA with microECC

We present the measured runtime and static code size of this library in comparison to ECDSA. There are three ARM configurations with different assembly (ASM) code optimizations for each of the curves. Optimisations can be either focused on small code size or fast execution. Optimisations can also be disabled altogether.

MicroECC shows how fast the differences between a generic implementation and a platform-dependent assembler-optimised version can be. For example, Secp224r1 without any optimisation runs in 0.593ms, while the platform-dependent fast version only took 0.262ms in average for these results. This is a notable advantage of 0.331ms, which amounts to a reduction of more than 50%.

In case of MicroECC with ECDSA, the results of the various curves and optimisations for the tasks of key generation, signing and verification show significant differences. Moreover, although the verification process was time-consuming, the difference between sign and verify rests on 9.7%. In comparison to TweetNaCl the performance improvement is here at approximately 50%. The high security key length Secp256k1 is about 20% faster than Secp256r1.

Table 5 shows the experimental results of the investigated curves. Applied ASM code optimisation (labelled fast, small and none) is documented in the “config” column. Investigated tasks were key generation, signing and signature verification (labelled key gen, sign and verify).

Table 5: MicroECC ECDSA timing comparison.

curve	config	key gen	sign	verify
secp160r1	fast	0.177 s	0.211 s	0.225 s
secp160r1	none	0.318 s	0.356 s	0.386 s
secp160r1	small	0.260 s	0.296 s	0.320 s
secp192r1	fast	0.181 s	0.206 s	0.224 s
secp192r1	none	0.409 s	0.437 s	0.485 s
secp192r1	small	0.305 s	0.333 s	0.368 s
secp224r1	fast	0.262 s	0.297 s	0.324 s
secp224r1	none	0.593 s	0.630 s	0.700 s
secp224r1	small	0.438 s	0.475 s	0.525 s
secp256k1	fast	0.420 s	0.467 s	0.476 s
secp256k1	none	0.925 s	0.972 s	1.010 s
secp256k1	small	0.665 s	0.714 s	0.737 s
secp256r1	fast	0.489 s	0.537 s	0.595 s
secp256r1	none	1.129 s	1.177 s	1.320 s
secp256r1	small	0.805 s	0.855 s	0.957 s

The general difference of the code size between a curve with a small key size and one with a more secure key size is smaller than expected. Between the smallest curve (Secp160r1) and the largest (Secp256r1) are only 168bytes. In addition, code size savings between the versions optimised for code size compared to those without are static an improvement of around 200 bytes. Table 6 shows MicroECC size comparison, focusing on the static code size of the different configurations.

Table 6: MicroECC size comparison.

curve	config	static code size
secp160r1	none	4679 byte
secp160r1	small	4522 byte
secp160r1	fast	5301 byte
secp192r1	none	4223 byte
secp192r1	small	4042 byte
secp192r1	fast	5225 byte
secp224r1	none	4311 byte
secp224r1	small	4130 byte
secp224r1	fast	5825 byte
secp256k1	none	4227 byte
secp256k1	small	4050 byte
secp256k1	fast	6301 byte
secp256r1	none	4531 byte
secp256r1	small	4354 byte
secp256r1	fast	6605 byte

2.1.4.4 Runtime and memory overhead of MicroECC on the RE-Mote with Contiki

Here, we focused on ECC signatures on the Secp192r1 curve using the MicroECC library running on a RERUM device with Contiki. Before the message was signed, it was hashed with the built-in, hardware-accelerated SHA256 function of the RE-Mote's CC2538 chip. Afterwards the generated signature was encoded into the BASE64URI format.

The runtime overhead of the signing process was determined by using the integrated timer of the CC2538 chip. These measurements were performed multiple times for each step of the signing process. It turned out that the SHA256 hashing and the BASE64URI encoding had a minimal impact, that did not significantly affect the runtime: The total duration of hashing and encoding was less than 1ms. The average overhead was 216ms for the Secp192r1 signature utilizing MicroECC as shown in Table 7.

Table 7: Runtime of average execution time.

Execution time overhead average	Execution time [ms]
With MicroECC signatures (curve Secp192r1)	216

The measurement of execution overheads for sending messages with or without signatures, was performed by additionally measuring the transmission time for the respective messages and further adding these results to the specific execution time of the crypto algorithms. The results are shown in Table 8.

Table 8: Execution time overhead (in ms) for sending messages involving signatures.

Time of Tasks in ms	Message transmission	Signing	Total interaction	Overhead
Without signatures	102	-	102	-
With signatures	186	216	402	300

The memory-overhead for ECC signatures on the RE-Mote results from the MicroECC library, the BASE64URl encoding algorithm and additional variables like the signing buffer. To measure the overhead we used tools from the gcc-arm-none-eabi toolchain. The compiled firmware image for the RE-Mote was first stripped, then the size was measured using the gcc-arm-none-eabi-size tool.

Comparing the measured sizes of the full firmware image, enabling the signatures leads to a total memory overhead of 5.71 kilobytes. The biggest part of the overhead comes from the .text section stored in the device's ROM which contains the MicroECC and base64 libraries. The .data section is almost not affected and the increased size of the .bss section results from the additional, uninitialized buffers, such as the signature buffer. This more detailed investigation resulted in the values shown in Table 9.

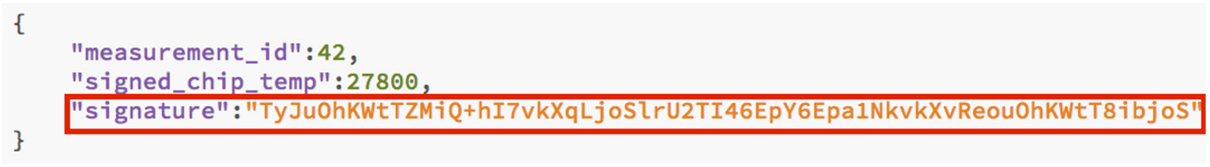
Table 9: Memory Overhead of standard signatures.

Memory consumption	Memory [bytes]			
	.text section	.data section	.bss section	total
With signatures	58895	1706	12765	73366
Without signatures	53315	1702	12497	67514
Overhead	5580	4	268	5852

A more detailed analysis of the runtime and memory overhead including a comparison with Malleable Signatures is found in Section 2.3.5.

2.1.4.5 Communication overhead of ECDSA with MicroECC on the RE-Mote with Contiki

The actual payload sent by the RERUM Device is a JavaScript Object Notation (JSON) message containing the measurement ID, the measured value, and the signature. An example of such a message is depicted in Figure 4. The length of this message is 125 characters, but can vary, depending on the size of the measurement ID and the BASE64URI encoding. The same message without integrity protection contains no signature (see the red marked area) with a resulting length of 46 characters. Hence, the overhead of sending a signed message is 79 characters (bytes).



```
{
    "measurement_id":42,
    "signed_chip_temp":27800,
    "signature":"TyJuOhKWtTZMiQ+hI7vkXqLjoSlrU2TI46EpY6Epa1NkvkXvReouOhKWtT8ibjoS"
}
```

Figure 4: Standard ECC signed message.

2.2 Energy efficiency of ECC-based payload signatures

This section continues the evaluation of ECC-based signatures that was presented in Section 2.1. We now shift our focus to the evaluation of the energy efficiency of the same signature schemes.

2.2.1 Implementation details

We used the same libraries as those used for performance measurements:

1. TweetNaCl library (Curve25519 and Ed25519)
2. Implementation by Piñol Oriol (Secp256r1)
3. MicroECC library (Secp160r1, Secp192r1, Secp224r1, Secp256k1, Secp256r1)
4. MicroECC in the RERUM Device for Contiki

2.2.2 Experimental setup

The power-trace is generated by placing a circuit between the USB-power-supply and the RE-Mote. An analog-digital-converter (AD-converter) of type MCP3008 [B12] was used to do the power measurements. The converter was triggered by a Raspberry Pi over the Serial Peripheral Interface (SPI). The MCP3008 is supplied by 3.25V and has a resolution of 1024bits, which results in a maximal resolution of 3.25V 1024bit = 0.0031738 bit V = 3.17mV bit [MT05].

The AD converter needs between 2.5 milliseconds (ms) and 10 milliseconds per measurement. Another limitation is that the script that reads those values, prints the current system time, which has a resolution of 10 milliseconds. The output of the Raspberry Pi below illustrates this problem. For this reason, the minimum size of a task cycle is limited to 10 ms.

We compute the voltage, current, and power of the RE-Mote as follows:

$$U_{Re-MOTE} = U_{R1} + U_{R2} = 2 \cdot U_{R2} = 2 \cdot U_{AD1}$$

$$I_{Re-MOTE} = I_{R_{34}} = \frac{U_{R3}}{R_{34}} = \frac{U_{AD0}}{R_{34}}$$

$$P_{R3} = U_{AD0} \cdot \frac{U_{AD0}}{R_{34}} = \frac{U_{AD0}^2}{R_{34}}$$

$$P_{Re-MOTE} = U_{Re-MOTE} \cdot I_{Re-MOTE} - P_{R_{34}} = 2 \cdot U_{ad1} \cdot \frac{U_{ad0}}{R_{34}} - \frac{U_{ad0}^2}{R_{34}}$$

As mentioned previously, the start/stop points are marked by activating the LEDs. The high power drain of LED is used as a marker. To calculate the energy drain for a specific function the power-trace is checked for the trigger of the LED. Then for each nearby measuring point the average over 10ms is computed and multiplied by 10ms to get the demand in Wms. Only measuring points within the same time span with an accuracy of 10ms are taken into account. Of course we always check if the next measuring point has the value of the led-marker, if so it will not be taken into account for the average calculation. Afterwards the sum of all average measuring points between the two markers will be used to calculate the total consumption of a given task. For each task the tests are running several times until a certain time limit is reached. We used the average value of several runs as the following Equations show:

$$W_{run\_task} = \sum_{crypto\_sign\_keypair\_end}^{crypto\_sign\_keypair\_begin} W_{Re-MOTE}$$

and

$$W_{total\_task} = \frac{\sum_n^1 W_{n\_run}}{n}$$

Figure 5 shows the summarised power values and detailed information like voltage, current, power, time, and the difference between the time measured with our timelib-tool and the time calculated out of a power trace for each task.

```
task: runs 2 crypto_sign_keypair timelib: 3.524066 s powertrace:
    3.525000 s diff 0.000934 s
     U: 4.809504 V I: 0.016809 A P: 0.080842 W W: 0.285370 Ws
task: runs 2 crypto_sign timelib: 3.538031 s powertrace: 3.530000 s
    diff 0.008031 s
     U: 4.809903 V I: 0.016734 A P: 0.080488 W W: 0.284524 Ws
task: runs 2 crypto_sign_open timelib: 7.058435 s powertrace: 7.055000
     s diff 0.003435 s
     U: 4.809081 V I: 0.016840 A P: 0.080970 W W: 0.571646 Ws
task: runs 1 alice key generation timelib: 2.080548 s powertrace:
    2.090000 s diff 0.009452 s
     U: 4.809073 V I: 0.016876 A P: 0.081144 W W: 0.169591 Ws
task: runs 1 bob shared key generation timelib: 2.080554 s powertrace:
     2.080000 s diff 0.000554 s
     U: 4.809495 V I: 0.016828 A P: 0.080931 W W: 0.169146 Ws
task: runs 1 bob key generation timelib: 2.080755 s powertrace:
    2.070000 s diff 0.010755 s
     U: 4.810039 V I: 0.016749 A P: 0.080561 W W: 0.167567 Ws
task: runs 1 alice shared key generation timelib: 2.080316 s
    powertrace: 2.080000 s diff 0.000316 s
     U: 4.809295 V I: 0.016836 A P: 0.080967 W W: 0.169220 Ws
```

Figure 5: Power analyzer sample output.

2.2.2.1 Power measurement

The power measurement is done with a AD converter MCP3008 [MT05]. The python scripts and wiring diagrams, are based on the work of Erik Bartmann [B12]. The current voltage was measured at AD0 and AD1. The Raspberry Pi communicates with the MCP3008 over Serial Peripheral Interface (SPI). The current voltage from AD0 or AD1 is read by a python script and directly written into a file on the Raspberry Pi.

The MCP3008 has a resolution of 3.17mV/bit in a range from 0V to 3.25V. A notable limitation is that the RE-Mote shuts down if the current is too high. In addition, a high current shortens the lifespan of the resistors. In the actual measurement, two resistors with a value of 10Ω in parallel delivered the best resolution for the low current consumption of the RE-Mote:

$$\frac{1}{R_{AD0}} = \frac{1}{R_3} + \frac{1}{R_4} = \frac{1}{10\Omega} + \frac{1}{10\Omega} = 5\Omega$$

The output of the script starts, by printing the arguments for the resistor (R3, R4) value and the sleeping time, afterwards the actual measurement lines are printed. A measurement line consists of the current timestamp, and the second time value (that was initially planned to measure the time for the process of reading over the SPI, but showed not trustworthy enough). The third value is the current voltage at the measure point AD1 and the last value at AD0.

Figure 6 shows a simplified construction of the circuit. Through the resistors (R1, R2) the current voltage is divided by half and can be measured at AD1. Through the resistors R3 and R4 the voltage between GND and AD0 can be measured and the current drain of the RE-Mote is calculated.

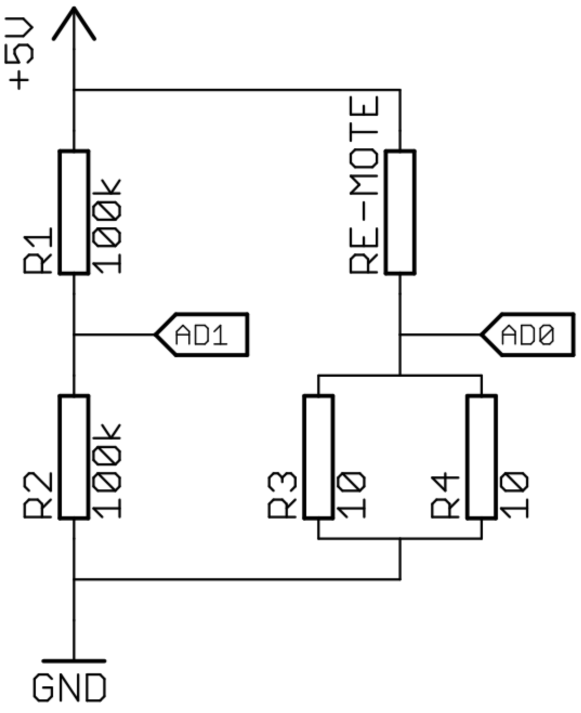


Figure 6: Wiring diagram [M16].

Figure 7 shows the finished measurement utility as a USB stick connected to a Raspberry Pi model B.

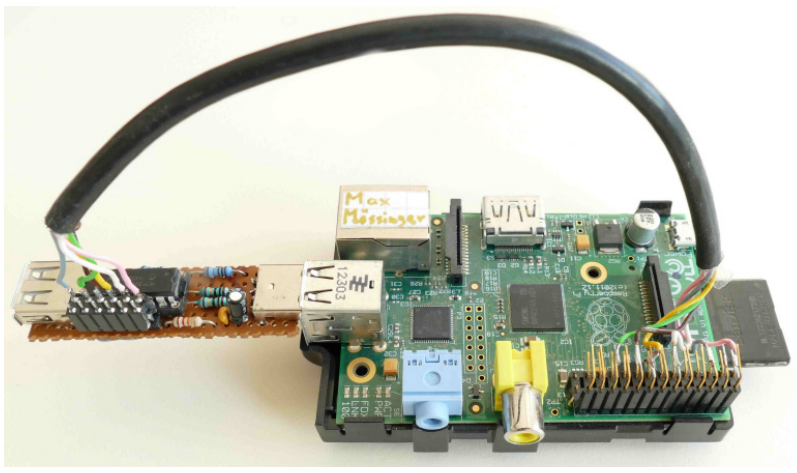


Figure 7: USB stick to measure energy consumption of the RE-Mote connected to a Raspberry Pi [M16].

A more detailed view of the actual wiring is shown in Figure 8. The capacitor and R5 stabilize the power supply for the MCP3008 from the Raspberry Pi, in order to gain a more accurate reading at the measurement points.

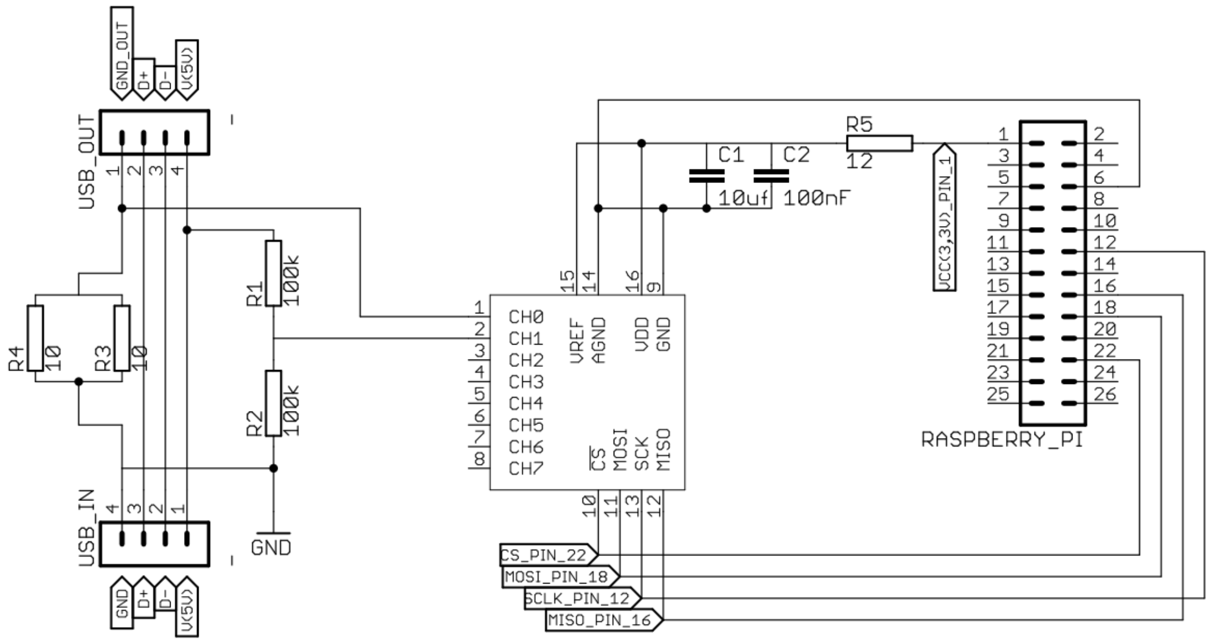


Figure 8: Detailed wiring diagram [M16].

2.2.2.2 Powertracer

Over the USB output, the timelib generates a timing trace, saved as a log file. In a second run the Raspberry Pi generates the power data for the test. To automate the process of analysing these traces, we developed the powertracer-tool. It consist of four stages: (i) first, the given traces can be read, (ii) split into tasks, then (iii) synced with the power data and, finally (iv), the statistics are calculated for the given tasks.

There are two kinds of trace: first the timelib trace which is the output over the USB device (see Figure 9). It consists of the different task names and their duration. The second trace is the power trace, which consists of the voltage on the different measurement points and the marker positions of the glowing LED marker points (see Figure 10).

```
start: crypto_sign_keypair
innerStart: hash
innerStop: 3082 us hash
stop: 3521148 us crypto_sign_keypair
```

Figure 9: Sample timelib trace from Curve25519 crypto_sign_keypair.

```
ohm:  5
sleeptime:  0
now:  1436314021.22 time:  0.00122785568237 AD1 768.0 AD0 27.0
now:  1436314021.22 time:  0.00127983093262 AD1 768.0 AD0 26.0
```

Figure 10: Sample power trace.

In a first step, the power trace is searched for the string “PowerEvents”, which marks the power consumption of the LEDs when switched on. Afterwards, a filter is applied to normalize the peaks. For example, events which are not possible since their occurrence is too short, are split depending on the time range where they occur (Figure 11).

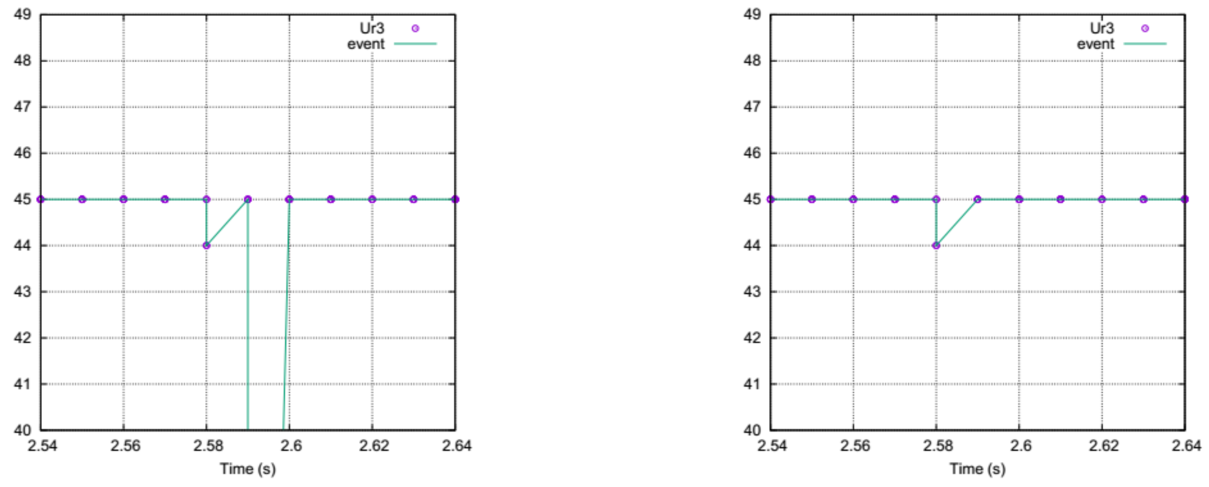


Figure 11: Power event concatenation before and after.

A “PowerTask” represents the time and power usage of a given task, e.g. of signing a message. After the power events are filtered, the “PowerTasks” are now recognizable and comparable to the tasks out of the timelib trace. The “PowerTask” and the tasks triggered by the timelib can be mapped to a “TimeLibTask” to accumulate the measurement information from the power trace to the information of the RE-Mote console output. Than the different runs of a task are summarized to create the average for given measurement. The difference between data of two sources is also an indicator about how credible the measurement of the different methods is. The “TimelibTask” results in the following information:

- name (Timelib)
- time (Timelib)
- time (PowerTrace)
- power data (PowerTrace)

2.2.3 KPIs

In this experiment, we measure the Crypto Energy Consumption (AL.EF.5) as defined in Deliverable D5.1 [RD5.1] for ECDSA with TweetNaCl, with Secp256r1 by Piñol, with MicroECC, and as well with microECC on the RE-Mote with Contiki.

2.2.4 Performance evaluation

Here we illustrate and compare the power consumption for the implementations TweetNaCl, sep256r1 and MicroECC, in time and size.

2.2.4.1 Power consumption of ECDSA with TweetNaCl

The graph in Figure 12 shows a power trace of an ECDSA signature followed by an ECDH signature (both from the TweetNaCl implementation).

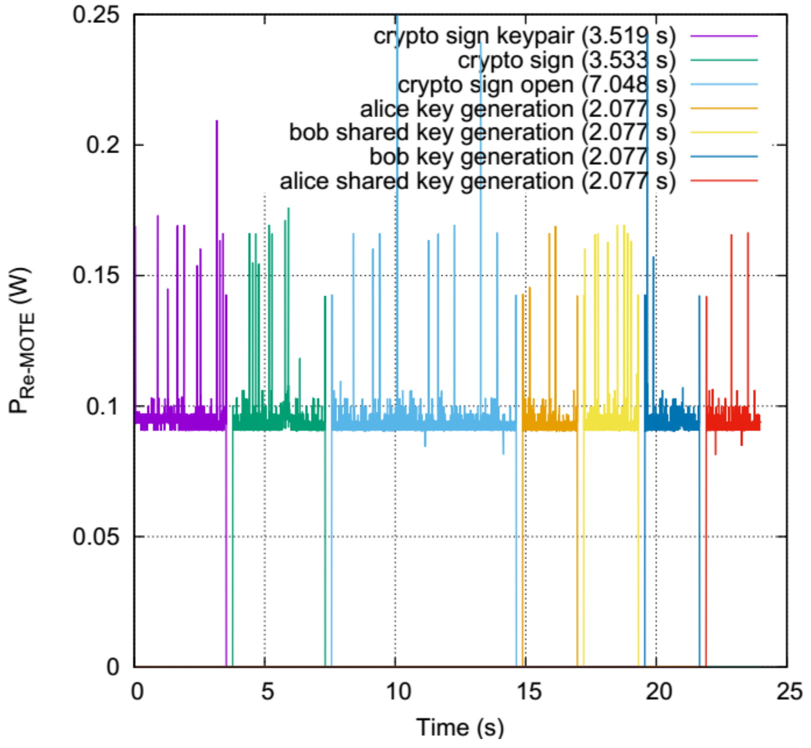


Figure 12: Average power for TweetNaCl [M16].

The measured power values for the ECDSA algorithm are shown in Table 10.

Implementation	Curve	keygen	sign	verify
TweetNaCl	ed25519	0.332	0.333	0.665

Table 10: TweetNaCl ECDSA power consumption.

2.2.4.2 Power consumption of ECDSA with Secp256r1 by Pinol

The data-dependent functions show differences in the power trace for the same kind of tasks. For the tasks that run more than 29 seconds, the power trace shows inaccuracies, due to the then insufficient measurement span of 10 minutes to accumulate sufficient data to calculate an meaningful average value. Another point worth to mention is the runtime of longer tasks. For example: In the configuration affine coordinates with 16-bit the task *alice key generation* took in the first run 43.21 seconds, but in the second run 46.3 seconds. This could also be verified in the powertrace.

An explanation would be the data dependent code inside of the Piñol implementation, where the run of a similar task (*alice key generation* 4.602 watt seconds, *bob key generation* 4.550 Ws) results in different power usages, even though they are using the same functions but with different data. Table 11 shows measured power values for an ECDSA with different configurations.

configuration	key gen	signing	verification
affine 16 bit	4.407 Ws	4.550 Ws	9.289 Ws
affine 16 bit sliding window	4.528 Ws	4.544 Ws	9.277 Ws
affine 32 bit	2.712 Ws	2.709 Ws	5.494 Ws
affine 32 bit sliding window	2.569 Ws	2.542 Ws	5.073 Ws
homogenous 16 bit	2.902 Ws	2.905 Ws	5.784 Ws
homogenous 16 bit sliding window	2.874 Ws	2.878 Ws	5.829 Ws
homogenous 32 bit	1.050 Ws	1.048 Ws	2.112 Ws
homogenous 32 bit sliding window	1.045 Ws	1.032 Ws	2.094 Ws
jacobian 16 bit	1.763 Ws	1.753 Ws	3.473 Ws
jacobian 16 bit sliding window	1.733 Ws	1.759 Ws	3.518 Ws
jacobian 32 bit	0.632 Ws	0.638 Ws	1.276 Ws
jacobian 32 bit sliding window	0.625 Ws	0.623 Ws	1.248 Ws

Table 11: Piñol ECDSA power comparison.

2.2.4.3 Power consumption of ECDSA with microECC

The power usage for MicroECC is analogue to the already presented results, and it confirms the measured timings using our timelib-tool. In the powertrace for ECDSA of MicroECC we can also measure differences between the three tasks, but for Secp256r1 they are marginal and rest up on a difference of 6mWs. Table 12 presents the results of the powertrace.

curve	config	key gen	sign	verify
secp160r1	fast	0.016 Ws	0.020 Ws	0.021 Ws
secp160r1	none	0.030 Ws	0.033 Ws	0.036 Ws
secp160r1	small	0.024 Ws	0.027 Ws	0.030 Ws
secp192r1	fast	0.016 Ws	0.019 Ws	0.020 Ws
secp192r1	none	0.037 Ws	0.040 Ws	0.044 Ws
secp192r1	none	0.028 Ws	0.031 Ws	0.034 Ws
secp224r1	fast	0.024 Ws	0.027 Ws	0.029 Ws
secp224r1	none	0.055 Ws	0.058 Ws	0.065 Ws
secp224r1	small	0.041 Ws	0.044 Ws	0.049 Ws
secp256k1	fast	0.038 Ws	0.042 Ws	0.043 Ws
secp256k1	none	0.085 Ws	0.089 Ws	0.093 Ws
secp256k1	small	0.064 Ws	0.069 Ws	0.071 Ws
secp256r1	fast	0.044 Ws	0.048 Ws	0.054 Ws
secp256r1	none	0.105 Ws	0.109 Ws	0.123 Ws
secp256r1	small	0.075 Ws	0.080 Ws	0.089 Ws

Table 12: MicroECC ECDSA power comparison.

2.2.4.4 Power consumption of ECDSA with MicroECC on the RE-Mote with Contiki

Energy measurements for this task were taken utilizing a special hardware circuit developed by Max Mössinger [M16]. This circuit is placed between the RE-Mote and a USB-power-supply. The actual metering is performed by the circuit's MCP3008 D/A-Converter monitored by a Raspberry PI over a SPI interface. Both, the current voltage, as well as the current impedance of the RE-Mote was monitored for a resolution of 2.5 milliseconds. The energy consumption of the RE-Mote was derived by calculating the actual wattage and multiplying this with the resolution of the time measurements at 2.5 milliseconds. The evaluation was split in two main parts: the measurement of the energy consumption on a standalone implementation of microECC for specific tasks, and the analysis of the actual RE-Mote-device prototype's power consumption with and without signing. Both scenarios utilize the microECC library and the Secp192r1 ECC curve. Table 13 shows the results of the standalone measurements.

Table 13: Average energy consumption of MicroECC.

Task	Key gen	Sign	Verify
Energy Consumption	0.016 Ws (≈ 0,0044mWh)	0.019 Ws (≈ 0.0052mWh)	0.020 Ws (≈ 0.0055mWh)

The monitoring of the energy consumption of the actual RE-Mote prototype was performed over 10 minutes. The device measured the temperature every 30 seconds, generated the JSON message and signed it. This message was then sent to a gateway utilizing COAP observe. This series of measurements resulted in 20 interactions between the RE-Mote and the gateway and a total of approximately 240.000 monitoring points for each of the two experiments. The resulting power-trace of the RE-Mote with enabled signing is depicted in Figure 13. This powertrace shows the current wattage every 2.5ms. The average wattage of around 190mW, as well there are some increases to 280mW and 380mW, indicating network activity and signing.

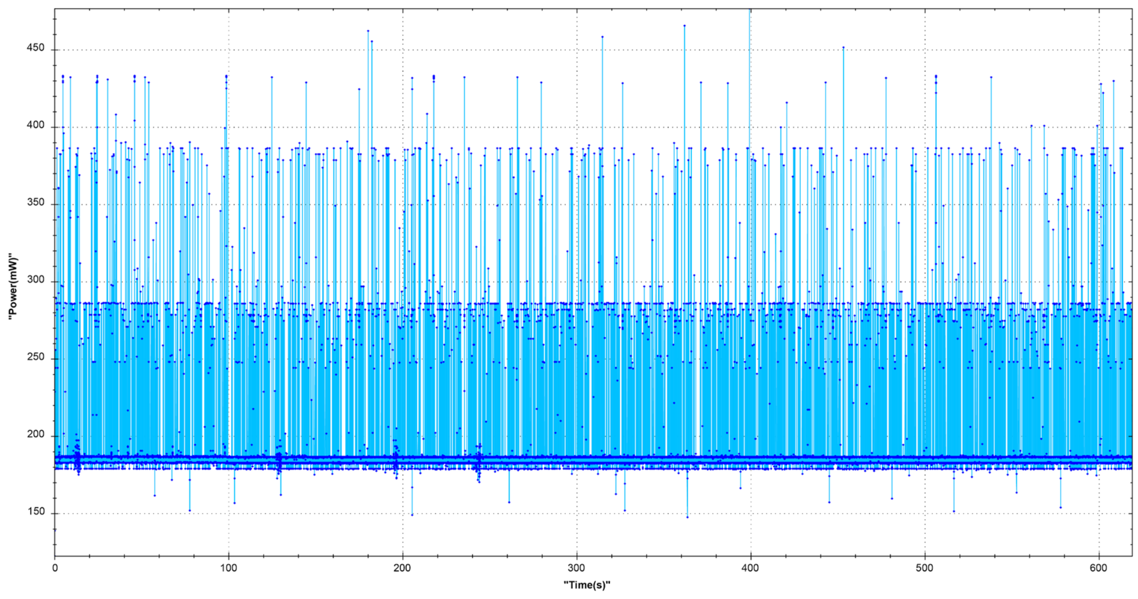


Figure 13: Powertrace of the RERUM-device over 10 minutes with Time(s) on the x-axis and Power(mW) on the y-axis.

Table 14 shows our measurements of the average power consumption of the RERUM-device, as well as the respective overheads.

Table 14: Energy Consumption of the RERUM-device and consumption overhead involving standard signatures.

Energy Measurements	With Signing	Without Signing	Overhead		
			Absolute	Per Signing	Relative
Average Wattage (mW)	188.9622	188.4694	0.4928	0.0246	0.26%
Average Amperage (mA)	30.2474	30.1525	0.0949	0.0047	0.31%
Average Consumption (mWs)	0.4724	0.4711	0.0013	0,00006	0.26%
Total Consumption (Ws)	119.8307	119.4552	0.3755	0.0188	0.31%

2.3 Overheads of signing and verifying with Malleable Signatures

Malleable signatures allow the signer to authorise subsequent transformations of the signed document, such that the transformed signature over the transformed document still verifies under the signer's public signature verification key. Furthermore, they decouple the transformation, called sanitization or redaction, from the signing and verification process. The class of redactable signatures allows to remove/cloak/redact parts from signed documents, if the signer has marked such parts as admissible to subsequent modification. RERUM sees this as a tool to carry out privacy enhancing changes to signed data, such that the remaining data still can be verified to be of authentic origin. Redactable signature schemes have many applications in the use cases of RERUM as described in Deliverable D3.2 [RD3.2] or in [PK14].

For malleable signatures there are many cryptographic schemes in the cryptographic literature as well as those developed in the course of this project (see Deliverable D3.1 [RD3.1] and D3.2 [RD3.2]). The results obtained from comparing different implementations of standardised cryptographic operations in the domain of ECC signatures that was carried out as an initial laboratory experiment yielded that implementing crypto operations in sufficient speed is a not a trivial task. In the ideal case, the malleable signature scheme shall be instantiated on already well implemented and at best hardware-accelerated basic cryptographic operations. As we saw the MicroECC implementation of ECDSA signatures seems to be a good choice.

In the course of RERUM several malleable signature schemes with a focus on the subsequent transformation of redact/cloak where developed [PS15], [MPPS14], or [PS14]. Due to limited time and resources we could only implement one scheme on actual hardware. So only the latter scheme, presented at the ACNS conference in 2014 [PS14], was chosen for implementation on a RE-Mote. The reason for choosing this specific scheme is because it has some cryptographic building blocks that were not found in the already existing ECC libraries. It also it offers a lot of features that we consider to be of unique interest to the IoT community. Therefore, we did not want to rely solely on analytical overhead calculations, but we wanted to have reliable statements.

In detail the scheme offers the following properties: (1) an attacker cannot generate non-legitimate signatures, (2) redacted information remains hidden, and (3) that all algorithms create linkable versions of the same message. The third requirement allows this scheme to be usable in the flow of information in the IoT as an attacker is not be able to generate "clones" of a signed message by gradually redacting redactable message parts.

This scheme presented in [PS14] explicitly defined an operation called merging that among other purposes can serve as a test when it is questionable whether or not two messages are derived from the same original. However, we also performed some analytical calculations of the overhead based on the actual numbers that we got from our laboratory experiments on standard ECC signatures, namely on the most efficient MicroECC implementation, for other malleable signature schemes.

Therefore in the remainder of this section, the following schemes will be evaluated that we implemented and measured on the RE-Mote:

1. Updatable Redactable Signature from ACNS 2014 [PS14]
2. Analytical overhead based on ECC measurements done on RE-Mote (see previous sections):
    a. EUROPKI12 Non-Interactive Public Accountability for Sanitizable Signatures [BPS12]
    b. ICETE12 Redactable Signature Schemes for Trees with Signer-Controlled Non-Leaf-Redactions [MPPS14]

2.3.1 Implementation details

The ACNS2014 scheme bases upon a cryptographic trapdoor accumulator as presented in [PS14]. It is a building block which must be coded on the RE-Mote as it previously did not exist. The following is taken from [PS14]:

We use the ideas given in [BP97], but make use of the trapdoor φ(n).

Construction 1 (Trapdoor-Accumulator ACC)

We require a division-intractable hash-function $H : \{0,1\}^* \rightarrow \{0,1\}^\lambda$. A formal definition is given in [24]. LetACC := (Gen, Dig, Proof, Verf) such that:

- ***Gen:***
  Generate n = pq, where p and q are distinct safe primes of length λ.
  A prime p is safe, if p=2p' +1 , where p' is also prime.
  Return (φ(n), (n, H)), where φ(pq) := (p − 1) · (q − 1).
- ***Dig:***
  To improve efficiency, we use the build-in trapdoor. A new digest can therefore be drawn at random. Return $a \in_R Z^{\times}_n$.
- ***Proof:***
  To generate a witness p_i for an element v_i, set $v_i' \leftarrow H(v_i)$.
  Output $p_i \leftarrow a^{v_i'^{-1} \pmod{\varphi(n)}} \bmod n$
- ***Verf:***
  To check the correctness of a proof p w.r.t. an accumulator a, the public key pk_{ACC}, and a value v, output 1, if $a = p^{H(v)}$ (mod n), and 0 otherwise.

We do note that this construction is related to GHR-signatures [GHR99]. Due to the build-in trapdoor, we do not require any auxiliary information as proposed in [BP97]. The use of safe primes allows us to almost always find a root. If we are not able to do so, we can trivially factor n.

The proofs that our trapdoor-accumulator is strongly collision-resistant can be found in the appendix.

Note, a is drawn at random for efficiency. We can also use the slower method aforementioned: a will be distributed exactly in the same way.

To implement the primes in the size suitable to provide cryptographic protection on the RE-Mote, we used the functions for big integer computation provided by the Contiki implementation of Piñol bigint.h (available: https://github.com/oriolpinol/contiki). This library was also part of the code that was measured for ECC in the previous sections. The library bigint.h provides functions that help perform arithmetical operations on big integers of unlimited size. The implementation of the accumulator on a RE-Mote was done by Noëlle Tiana Lilie Rakotondravony [R15].

The algorithm requires the transported message to be split into parts. For the ease of implementation and presentation of the results we used a message encoding in JSON, that would just have two parts containing for example the information of temperature of very high precision 27,4556 °C allowing a modification such that "27," or "4556" can be redacted. The construction uses the trapdoor accumulator as documented in [PS14].

Hence, as an example, the JSON would look like the one depicted in the following code snippet:

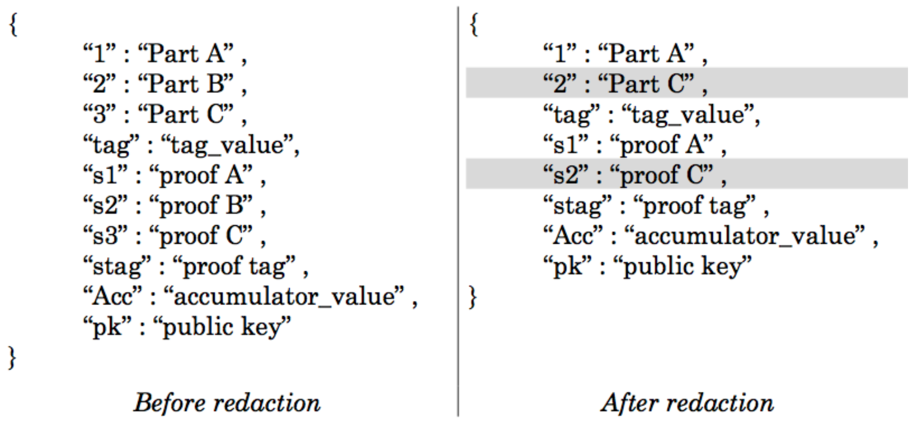
```
{
 "measurment_id":42, "1":"27,", "2":"4556",
  "tag":"0000f42d8ccff646" ,□
   "s1":"3eeb40c758de81a8ebe6bc3f07f4cd65bb49f1356adc020602a8cee8810006",□
   "s2":"5a26affcbaacc65bbf8eebf1f3b6719a54777006155ba1ab4de2c734910000",□
□"stag":"4afb990ca61a143f99c62550a32cfb4592f3f550c517148957dcfe4e540008",□
  "Acc":"3c27c776524e9ddd590b5b6a25a0d3b339bc09f6497a86af40699e6aac000f",
   "Pk":"bba15a4ed441b4ecc32dc46f09e2c32927a215181b19aec21f6806b7260014"
□}
```

Note, the above figure shows the JSON elements that are necessary for the content split into parts and the signature as well as the public key. This is the JSON string, which will be sent by the RE-Mote. The JSON structure contains:

- the remaining parts of the message (marked as `1` and `2`),
- the tag,
- the signature sigma which is the proofs of each remaining part of the message (`s1` and `s2`)
- the proof for the tag
- the Accumulator value (`Acc`),
- the public key n to be used by the verifier to check the validity of the signature.

The public key is not really needed in every packet and it can therefore be elided. Here, we consider the potential transport of the public key as the worst case scenario. All the other values are required for the scheme to work.

The sign algorithm of [PS15] will use the accumulator's digest and proof operation. It will add each part of the message (e.g. "`27,`" and "`4556`") into the accumulator value and it will produce a witness for each part. The verify algorithm will use the accumulator's verify operation to check if for each part the witness verifies w.r.t. the accumulator value. Note, that the redact operation is very cheap. To redact, only string operations are necessary as depicted in Figure 14 (taken from [R15]).



```
{                                      {
    "1" : "Part A" ,                       "1" : "Part A" ,
    "2" : "Part B" ,                       "2" : "Part C" ,
    "3" : "Part C" ,                       "tag" : "tag_value",
    "tag" : "tag_value",                   "s1" : "proof A" ,
    "s1" : "proof A" ,                     "s2" : "proof C" ,
    "s2" : "proof B" ,                     "stag" : "proof tag" ,
    "s3" : "proof C" ,                     "Acc" : "accumulator_value" ,
    "stag" : "proof tag" ,                 "pk" : "public key"
    "Acc" : "accumulator_value" ,      }
    "pk" : "public key"
}
```

Before redaction *After redaction*

Figure 14: Exemplified overview to highlight that redact in [PS15] is only a string-based operation.

2.3.2 Experimental setup

The experimental setup is identical to the one used for the performance evaluation of ECC-based signature schemes. It is described in detail in Section 2.1.2.

2.3.3 Analytical evaluation of other malleable signature schemes

Based on the results from the measurements of real ported implementations of several cryptographic schemes (including one malleable signature scheme denoted as ACNS2014 presented in [PS14]) and basic operations, we have chosen the following schemes for an analytical evaluation:

- EUROPKI12 Non-Interactive Public Accountability for Sanitizable Signatures [BPS12]
- ICETE12 Redactable Signature Schemes for Trees with Signer-Controlled Non-Leaf-Redactions [MPPS14]

In the following, we briefly explain the underlying routines of the two malleable signatures schemes. We do not consider message overhead for those schemes, as this requires to encode the actual information into a JSON message, and that can be done using various different ways. Best it shall be tailored towards the content that is transported in each use case. Hence, this was not part of the analytical evaluation.

2.3.3.1 EUROPKI12 - Non-Interactive Public Accountability for Sanitizable Signatures [BPS12]

The scheme uses two standard digital signatures that are overlapping in the information they protect and it requires two key pairs, one for the signer and one for the sanitizer. During signature generation the signer includes the sanitizer's public key in the signature and by that endorses any party that holds the corresponding secret key to carry out sanitizations. Sanitizations here means that for a part which is admissible for a subsequent change the contents of that part can be changed arbitrarily into any string. The key for a sanitizer allows the signer to appoint the sanitizer, while it is possible to attach the sanitizer's private key alongside the message and thus enable anyone to sanitize the message [P13] this has not been looked at for this deliverable.

While overhead still occurs in the parsing and generation of strings, this is neglected in the analytical overhead. Moreover, the way the scheme is constructed requires the following operations of a standard digital signature (e.g. microECC):

- EUROPKI12 Sign algorithm:
  Two digital signature signing operation both with the signer's secret key
- EUROPKI12 Sanitize algorithm:
  One digital signature signing operation with the sanitizer's secret key
- EUROPKI12 Verify algorithm:
  Worst case three digital signature verify operations, two with the signer's and one with the sanitizer's public key

Assuming for comparison that we use the MicroECC library with curve Secp192r1 in, so-called, fast mode (signing: 0.206s and verify: 0.224s) this yields the following analytical worst case runtime estimation depicted in Table 15:

Table 15: Analytical runtime estimation for sanitizable signature scheme from [BPS12] on RE-Mote in seconds.

Scheme	Sign	Redact	Verify
EUROPKI12	2 * 0.206s = 0.412s	1 * 0.206s = 0.206s	3 * 0.224s = 0.672s

2.3.3.2 ICETE12 - Redactable Signature Schemes for Trees with Signer-Controlled Non-Leaf-Redactions

The scheme from [MPPS14] uses an accumulator based on the work done in [BM93]. This relates to the same scheme and accumulator that was implemented on the RE-Mote and tested by us in our lab experiments. This accumulator is based on the RSA primitives of modulo arithmetic and it is left as open work to see if this can be sped-up by the use of RSA accelerating crypto co-processors. Additionally the scheme from [MPPS14] traverses the data structure of the tree, in RERUM the data structures being signed are assumed to be JSON (see also [P15]). It builds a Merkle hash tree over the tree. To construct a Merkle-Hash-Tree the tree is traversed such that for each node x in the tree a so-called Merkle-Hash (denoted as MH) is calculated as:

$$MH(x) = H(H(cx)||MH(x1)||\ldots||MH(xn)),$$

where H is a collision-resistant hash-function like SHA and || denotes a concatenation.

The depth and number of elements of the JSON structures sent by RERUM devices are simple, thus the overhead of traversing and building a Merkle hash tree is not that big: Also, as previously noted in our laboratory experiments, the computation of the SHA hashes does not represent the major overhead.

Thus, we expect the runtime performance of [MPPS14] to behave like that of the malleable signature scheme implemented in the laboratory experiment.

2.3.4 KPIs

In this experiment, we evaluate the following Key Performance Indicators (KPI) defined in Deliverable D5.1 [RD5.1]:

- Crypto-Runtime-Overhead (AL.PE.3),
- Crypto-Memory-Consumption-Overhead (AL.EF.3), and
- Crypto-Communication-Overhead (AL.EF.4)

for ECDSA with microECC on the RE-Mote with Contiki.

2.3.5 Performance evaluation

In this section, we illustrate and compare the measured results of ECDSA with MicroECC on the RE-Mote with Contiki with Malleable Signatures in time and size.

2.3.5.1 Runtime and memory overhead of ECDSA with microECC on the RE-Mote with Contiki in comparison to malleable signatures

The runtime overhead of the signing process was determined by using the integrated timer of the CC2538 chip. These measurements were performed multiple times for each step of the signing process. The resulting averages in comparison with standard signatures are shown in Table 16.

Table 16: Runtime of average execution time.

Runtime overhead average	Execution Time [ms]
With microECC signatures (curve Secp192r1)	216
With malleable signatures - signing	7374
With malleable signatures - verifying	5772

The execution time overhead of the different signature schemes showed significant variance; resulting in an average overhead of 216ms for the Secp192r1 signature utilizing MicroECC and 7374ms in case of the malleable signatures.

We performed the measurement of the runtime overhead for sending messages, involving both signed and unsigned payloads, by additionally measuring the transmission time for the respective messages and further adding these results to the specific runtime of the crypto algorithms. The results are summed-up in Table 17.

Table 17: Runtime overhead for sending messages involving signatures.

Time of Tasks in ms	Message transmission	Signing	Total interaction	Overhead
Without signatures	102	-	102	-
With signatures	186	216	402	300
With malleable signatures	751	7374	8125	8023

Measuring the memory overhead of the malleable signatures was performed by us on a prototype of the RE-Mote, resulting in an overhead of 10,32 kilobyte in total as shown in Table 18.

Table 18: Memory Overhead of malleable signatures.

Memory Footprint	Memory [bytes]			
	.text section	.data section	.bss section	Total
With malleable signatures	56730	1611	14081	72422
Without malleable signatures	48435	483	12941	61859
Overhead	8295	1128	1140	10563

2.3.5.2 Communication overhead of ECDSA with microECC on the RE-Mote with Contiki

A sample message generated by the implementation of the malleable signatures is shown in Figure 15. The length of a message signed with malleable signatures is 430 characters or bytes.

```
{
   "measurement_id":42,
   "1":"temp",
   "2":"27,",
   "tag":"0000f42d8ccff646",
   "s1":"3eeb40c758de81a8ebe6bc3f07f4c3d65bb469f1356adc020602a8cee8810006",
   "s2":"4dbbdd4281a84ce4f6353af78e9e4750685f4297438bb036769425c734910000",
   "stag":"82b56719aefc02795f835aee45ddc4b5792e2f9ff6c47b6d8957dcfe4e540008",
   "Acc":"3c27c776524e9ddd590b5b6a25a0d3b339bc09f6497a86af40699e6e9aac000f",
   "pk":"bba15a4ed441b4ecc32dc46f9809e2c32927a215181b19aec21f6806b7260014"
}
```

Figure 15: Malleably signed message.

Hence, a malleably signed message is over nine times bigger than an unsigned message (see Table 19):

Table 19: Average increase in message size of signing algorithms.

Message length	Bytes	Increase (bytes)
Without signatures	46	-
With standard ecc signatures	125	79
With malleable signatures	430	384

We used the sniffer tool Wireshark in order to examine the number of sent messages, a typical GET request from a client and the corresponding response from the RE-Mote were monitored (see Table 20).

Table 20: Average number of messages of a specific crypto algorithm.

GET request	Number of CoAP blocks
Without signatures	1
With signatures	2
With malleable signatures	7

An unsigned message fits in one CoAP block, resulting in one message to be sent by the RE-Mote. The message exceeding the lengths of a CoAP message, containing signed and malleably signed messages divided into several blocks. This results in the partitioning into two blocks for standard signatures and into seven blocks for malleable signatures.

Every GET request to the RE-Mote results at least two exchanged messages: one GET sent to the device and the corresponding response. In case the message is split into several CoAP blocks, the transmission of each block results in one GET request and one response. Thus, the use of signatures (or malleable signatures) increases the additional transmitted messages by two for each CoAP block. The corresponding results are shown in Table 21.

Table 21: Message overhead involving specific crypto algorithms.

GET request	Overall message count of an interaction	Overhead (Additional messages)
Without signatures	2	-
With signatures	4	+2
With malleable signatures	14	+12

The blockwise transfer of a CoAP message prevents the fragmentation of the 6LowPAN packets and thus the numbers of packets was equal to the number of COAP messages.

2.4 Energy efficiency of Malleable Signatures

This section continues the evaluation of Malleable Signature schemes that was presented in Section 2.3. We now shift our focus to the evaluation of the energy efficiency of the same signature schemes.

2.4.1 Implementation details

For the evaluation of energy consumption of malleable signatures, we use the same Contiki implementation for the RE-Mote, enhanced by an implementation of the ACNS2014 scheme [PS14]. This is identical to the implementation used for the experiments presented in Section 2.3.

2.4.2 Experimental Setup

The experimental setup is identical to the one used for the energy efficiency evaluation of ECC-based signature schemes. It is described in detail in Section 2.2.2.

2.4.3 KPIs

In this experiment, we measure the Crypto Energy Consumption (AL.EF.5) as defined in Deliverable D5.1 [RD5.1] for Malleable Signatures.

2.4.4 Performance Evaluation

Equal to experimental setup explained in Section 2.3.5.

2.4.4.1 Power Consumption of Malleable Signatures

The results of the standalone measurements show that the energy consumption for signing is 53 times higher and for verifying 68 times higher in comparison to standard ECDSA signatures. The results are summarized in Table 22.

Table 22: Average energy consumption of malleable signatures.

Task	Sign	Verify
Energy Consumption (ECDSA signature)	0.019 Ws (≈ 0.0052mWh)	0.020 Ws (≈ 0.0055mWh)
Energy Consumption (malleable signatures)	1.017 Ws (≈ 0.2825 mWh)	1.336 Ws (≈ 0.3711 mWh)

2.5 Lightweight Datagram Transport Layer Security

Transport Layer Security (TLS), based on reliable transport protocols such as the Transmission Control Protocol (TCP), is not suitable for protocols based on unreliable datagram traffic such as User Datagram Protocol (UDP). A modified version of TLS, Datagram Transport Layer Security (DTLS) has been introduced to address issues associated with unreliable connection. DTLS provides a mechanism that allows packet retransmissions and reordering whenever it is required in [MR04]. DTLS has been described more thoroughly in deliverable D3.1 [RD3.1].

2.5.1 Relevance to RERUM's Use-Cases

DTLS is applicable to any devices in the RERUM architecture that require secure communication (either end-to-end or hop-by-hop) at the transport layer level. It can be used transparently by any end applications, which themselves reside at the application layer of the TCP/IP network stack. This transparent feature implies that DTLS could be applied to all four of RERUM's Use-Cases. It directly addresses the following User Requirements, as specified in deliverable D2.1 [RD2.1]:

- UR-7: The user needs to protect his measurements from malicious users;
- UR-25: User requires open solutions for authentication between devices, ensuring the integrity of their data as well as the confidentiality.

Given how DTLS' implementation in the Android operating system is not available to RERUM for modification, in this section we focus on the evaluation of DTLS for the Zolertia RE-Mote platform. Therefore, this analysis is relevant for both indoor Use-Cases as well as UC-O2 – Environmental Monitoring.

2.5.2 Purpose of the experiment

The goal of this experiment is to check behaviour of DTLS protocol in the environment, which is closer to real deployment scenarios. The experiments should investigate two major issues: performance of implemented (in lightweight fashion) cryptographic schemes and performance of DTLS protocol itself.

The results from this experiment are part of the evaluation for the following evaluation criteria that have been defined in deliverable D5.1 [RD5.1]:

- AL.EF.3 "Crypto-Memory-Consumption-Overhead",
- AL.EF.4 "Crypto-Communication-Overhead",
- AL.PE.3 “Crypto-Runtime-Overhead”,
- AL.PE.4 “Lightweight Datagram Transport Layer Security (DTLS) Protocol”.

2.5.3 Implementation details

DTLS consist of handshake and record protocol. The main purpose of the DTLS is to authenticate communication parties, negotiate a cipher suite and exchange keys that are later used to protect traffic data. All handshake messages can be divided on six groups of messages (so-called flights) between communication parties (Figure 16); three flights are sent from a client to a server and three from a server to a client. Note that DTLS protocol consists of both secret (a pre-shared key) and public-key (raw public keys and X.509 certificates) options. A detailed description of DTLS operation is provided in D3.1 [RD3.1].

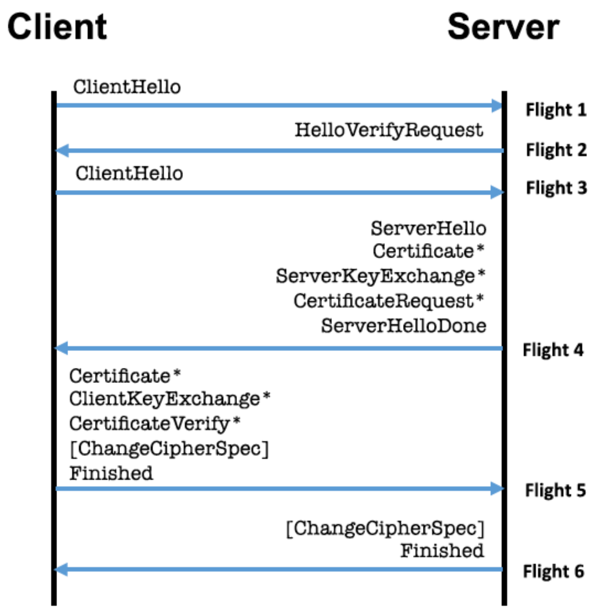


Figure 16: An overview of the DTLS handshake.

2.5.4 Experimental setup

In this set of experiments, we have employed two Zolertia Re-Mote IoT hardware boards: one was utilized as a client while the other as a server, respectively. We placed the two motes close to each other in order to guarantee the reliable and successful wireless communication (Figure 17). We ran our experiments on top of Contiki OS while each set of experiments was executed ten times to derive standard deviation values.

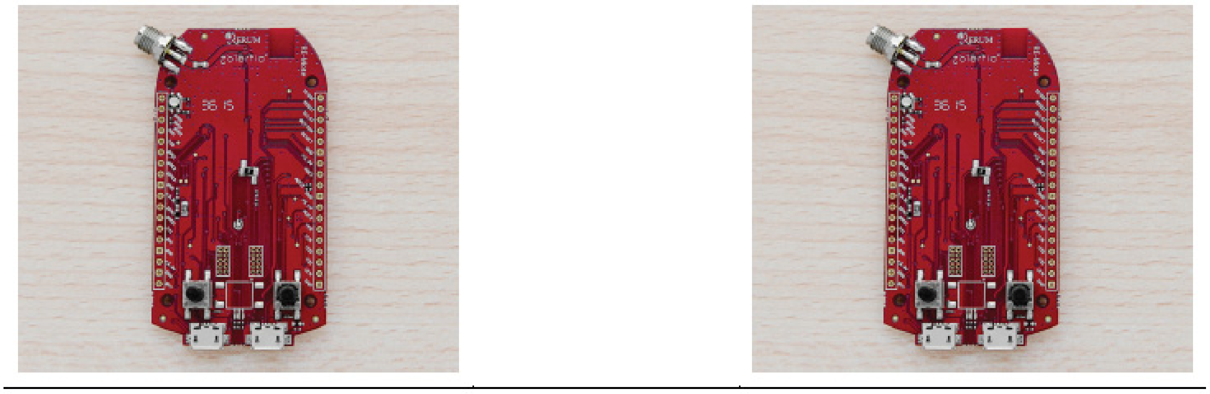


Figure 17: An overview of the Zolertia RE-Mote based experimental scenario.

2.5.5 KPIs

This section aims to benchmark the total handshake time of a specific implementation of DTLS. In this section we study the performance of DTLS version 1.2 (TinyDTLS), an optimized implementation of the DTLS protocol for embedded devices, both under Elliptic Curve Cryptography (ECC) with one of the well-known DTLS curves, the Secp256r1 curve (also known as NIST P-256, for the exact parameters see [RECFG]), and Pre-Shared Key (PSK) modes.

The KPIs that will be measured within this experiment are the following:

- Code footprint of cryptographic primitives,
- Overall latency of DTLS handshake under different modes.

2.5.6 Performance evaluation

2.5.6.1 Code footprint of cryptographic primitives

Both code footprint and RAM requirements are measured at compile-time. This is achieved by building a firmware image and by subsequently running the toolchain's -size command (e.g., $ arm-none-eabi-size obj_cc2538dk/dtls.o) on each object files. Note that both client and server are utilizing the same files in this TinyDTLS implementation. The results presented in Table 23, provide the following information about each code module:

- Code footprint, including program memory and constant (const) expressions (text)
- Variables initialized at compile time (data)
- Space reserved for variables which are not initialized at compile time (bss)

Table 23: Code size and RAM footprint in bytes for each object.

Object filename	.text	.data	.bss	Total
obj_cc2538dk/dtls.o	3224	12	8114	11350
obj_cc2538dk/crypto.o	335	0	1335	1670
obj_cc2538dk/hmac.o	1578	0	1335	2913
obj_cc2538dk/rijndael.o	6678	0	0	6678

obj_cc2538dk/sha2.o	1325	0	0	1325
obj_cc2538dk/ccm.o	1064	0	4	1068
obj_cc2538dk/netq.o	277	12	375	664
obj_cc2538dk/dtls_time.o	32	0	0	32
obj_cc2538dk/peer.o	150	12	45	207
obj_cc2538dk/session.o	148	0	0	148

Furthermore, the same command can be executed on the entire binary image, for instance $ arm-none-eabi-size dtls-client.elf. The results of this command are presented in Table 24, while the outputs provide the following information about the entire firmware:

- Code footprint, including program memory and constant (const) expressions (text): 67.77 and 67.29 KB for the client and server respectively
- Variables initialized at compile time (data): 614 (client) and 550 (server) bytes
- Space reserved for variables which are not initialized at compile time (bss): 84685 and 83899 bytes for the client and server respectively

Table 24: Code size and RAM footprint in bytes for the entire firmware image.

Firmware filename	.text	.data	.bss	Total
dtls-client.elf	67774	614	16297	84685
dtls-server.elf	67292	550	16057	83899

2.5.6.2 Overall latency of DTLS handshake

The handshake time is calculated from the first *"ClientHello"* message from the client node until the last message (i.e., "*Finished*", Handshake compete) from the server node. To measure this time, we connected each node with terminals in order to get debugging output regarding the messages exchanged between the two devices.

In Table 25, both average values along with standard deviation are provided. Our experimental performance evaluation results show that ECC requires 137 seconds in average, while PSK mode only 1.34 seconds. These results can be explained by the fact that in PSK mode, there is little computation, since the nodes generate the random values during the handshake, which is then exchanged. On the other hand, ECC, which is considered to be secure and efficient approach to public-key cryptography, which requires high computations both on the client and server side.

Note that our experiments took place on top of two constrained devices. We should consider that in most of the applications the gateways are operated over powerful devices, and the computation time for ECC would thus be halved, due to the fact that the computation time for the server side will be negligible short. Furthermore, the implementation of TinyDTLS under ECC mode was not designed to be executed explicitly on top of such constrained devices. It is important to be mentioned that there is room for optimization of the ECC both from the mathematical point of view and its implementation.

Table 25: Total handshake times for ECC and PSK.

Handshake times (in seconds)			
ECC		PSK	
Average	Standard Deviation	Average	Standard Deviation
137.324	0.65687	1.341	0.1912

Finally, the experimental evaluation results in a tradeoff between computation time and key management. Indeed, from the utility point of view, from the first glance ECC mode presents inefficient performance (i.e., long handshake time) when compared to PSK. However, we should take into account different type of applications that users may operate. For instance, it could be durable for time driven applications (e.g., temperatures or humidity readings), since it is not necessary for the devices to perform handshakes often. On the other hand, ECC is less efficient in real-time applications (e.g., turning-on the light). Furthermore, in dense scenarios where there are many nodes, ECC mode is more efficient in terms of key management, and thus, one time ECC overhead might be suitable, and moreover, the users benefit public key scheme..

2.6 Microsoft's NUMS Curves

As it was detailed in D3.1 [RD3.1], a group of researchers from Microsoft have designed a new set of elliptic curves [BCLN14] and proposed them to IEFT standardisation body [NUMS14]. Although this proposition is relatively new and security claims together with performance results have not been widely studied yet, there are several clues that the proposed set of curves could represent a good replacement for the NIST standard (especially taking into consideration curves defined over primes). MSR ECCLib [BCLN14] is an efficient cryptography library that provides functions for computing essential elliptic curve operations on a new set of high-security curves.

2.6.1 Relevance to RERUM's Use-Cases

In this section, we focus on the evaluation of MS NUMS curves for the Zolertia RE-Mote platform. Therefore, this analysis is relevant for both indoor Use-Cases as well as for UC-O2 – Environmental Monitoring.

2.6.2 Purpose of the experiment

The goal of this experiment is to check the feasibility of using NUMS curves with the Zolertia RE-Mote platform. The experiment investigates two evaluation criteria:

- The code size and RAM requirements of the implementation of NUMS curves for the Contiki Operating System and the RE-Mote platform.
- Execution time for various NUMS curve operations, again using the implementation of NUMS curves for the Contiki Operating System and the RE-Mote platform.

The results from this experiment are part of the evaluation for the following evaluation criterion that has been defined in deliverable D5.1 [RD5.1]:

- AL.PE.3 "Crypto-Runtime-Overhead"

2.6.3 Implementation details

Contrary to Curve25519 and Ed2519, and similarly to the NIST curves, one curve from NUMS set, (to be more specific those expressed as short Weierstrass equation and over primes) could be used both in ECDH and ECDSA algorithms. Below additional details are listed:

- MSR ECCLib supports six high-security elliptic curves proposed in [BCLN], which cover three security levels (128-, 192-, and 256-bit security) and two curve models. The curves have a very simple and deterministic generation with minimal room for parameter manipulation.
- It includes support for ECC functions necessary to implement most popular elliptic curve-based schemes. In particular, MSR ECCLib supports the computation of scalar multiplication for the six curves above in three variants:
    1. Variable-base scalar multiplication (e.g., this is used for computing the shared key in the Diffie-Hellman key exchange).
    2. Fixed-base scalar multiplication (e.g., this is used for key generation in the Diffie-Hellman key exchange).
    3. Double-scalar multiplication. This operation is typically used for verifying signatures.
- MSR ECCLib offers full protection against timing and cache attacks by executing crypto-sensitive operations in constant-time with no correlation between timing and secret data.
- It achieves high performance without compromising security, portability and usability.
- MSR ECCLib is supported on a range of platforms, including x64, x86, and ARM devices running Windows or Linux.

2.6.4 Experimental setup

This section aims to benchmark the previously presented NUMS library on constrained devices such as CC2538 System-on-Chip [TI13], which is the core of the Zolertia's RE-Mote platform. To this aim, we employed a Zolertia RE-Mote IoT hardware board. We run our experiments on top of Contiki OS while each set of experiments (i.e., each arithmetic) was executed ten times.

2.6.5 KPIs

In this section, we present the performance evaluation results in terms of the computation time for the MS NUMS curves. More specifically, we benchmark the computation time of the elliptic curves for various arithmetic operations.

The KPIs that will be measured in this section are the following:

- Code footprint of MS NUMS curves,
- Computation time of the elliptic curves.

2.6.6 Performance evaluation

2.6.6.1 Code footprint

Both code footprint and RAM requirements are measured at compile-time. This is achieved by building a firmware image and by subsequently running the toolchain's -size command (e.g., $ arm-none-eabi-size obj_cc2538dk/curves.o) on each object files. The results presented in Table 26, provide the following information about each code module:

- Code footprint, including program memory and constant (const) expressions (text)
- Variables initialized at compile time (data)
- Space reserved for variables which are not initialized at compile time (bss)

Table 26: Code size and RAM footprint in bytes for each object.

Object filename	.text	.data	.bss	.dec
ecc512.o	14928	0	0	14928
fp.o	9808	0	0	9808
ecc_additional.o	4232	0	1092	5324
extras.o	4632	0	0	4632
curves.o	0	3192	0	3192

Furthermore, the same command can be executed on the entire binary image (i.e., $ arm-none-eabi-size fp-test.elf). The results of this command are presented in Table 27, while its outputs provides the following information about the entire firmware:

- Code footprint, including program memory and constant (const) expressions (text): 53815 bytes,
- Variables initialized at compile time (data): 2046 bytes,
- Space reserved for variables that are not initialized at compile time (bss): 13551 bytes.

Table 27: Code size and RAM footprint in bytes for the entire firmware image.

Firmware filename	.text	.data	.bss	.dec
fp-test.elf	53815	2046	13551	69412

2.6.6.2 Computation time

This section presents experimental results on the time required for various field arithmetic computations over NUMS curves (averages and standard deviations). Note that in the NUMS library there is not specified (designed) security level lower than 256 [BCLN14].

As can be observed from the results, the higher the arithmetic value is, the longer the computation time required to perform the functions gets. Furthermore, the arithmetic for Galois Fields GF(2^256-189) as shown in Table 28 is more efficient than other two (i.e., GF(2^384-317) shown in Table 29, GF(2^512-569) shown in Table 30). However, it is used as the underlying arithmetic for the curve with the lowest security level among those curves specified in [BCLN14]. In fact, in many applications the 128-bit security level (i.e., 256) would be enough, but if users want to employ higher level of security in their applications, in this section we benchmarked the possibilities. However, there is a performance penalty, namely the increased computation time.

Table 28: Field arithmetic over GF(2^256-189).

Computation time (in microseconds)				
Functions	**curve numsp256d1**		**curve numsp256t1**	
	Average	**Standard Dev.**	**Average**	**Standard Dev.**
Subtraction	12.3	0.94868	12.6	1.26491

Division	11.1	1.44914	10.5	1.58114
Negation	6.6	1.26491	8.1	1.44914
Squaring	161	1.41421	160.1	1.44914
Multiplication	172.4	1.26491	174	2.1602
Inversion	47148.3	19.55121	47143.1	18.3875
Modular addition	46.8	1.54919	46.8	1.54919
Montgomery multiplication	486.2	1.54919	481.1	1.44914
Montgomery inversion	155276.3	68.51448	142843.2	57.08045

Table 29: Field arithmetic over GF(2^384-317).

Computation time (in microseconds)				
Functions	**curve numsp384d1**		**curve numsp384t1**	
	Average	**Standard Dev.**	**Average**	**Standard Dev.**
Addition	24.9	1.44914	25.8	1.54919
Subtraction	18	0	18	0
Division	14.1	1.44914	14.1	1.44914
Negation	9.9	1.44914	10.2	1.54919
Squaring	314.3	0.94868	314.3	0.94868
Multiplication	343.1	1.44914	343.4	1.26491
Inversion	129175.9	39.61888	129128.9	47.93387
Modular addition	72.1	1.44914	71.5	1.58114
Montgomery multiplication	1034.3	1.70294	1034	0
Montgomery inversion	491719.6	208.63535	489413.1	199.43946

Table 30: Field arithmetic over GF(2^512-569).

Computation time (in microseconds)		
Functions	**curve numsp512d1**	**curve numsp512t1**

	Average	**Standard Dev.**	**Average**	**Standard Dev.**
Addition	34.8	1.54919	35.1	1.44914
Subtraction	25.2	1.54919	25.5	1.58114
Division	19.2	1.54919	20.1	1.44914
Negation	12	0	12	0
Squaring	497	0	497	0
Multiplication	549.6	1.26491	548.1	1.44914
Inversion	269685.1	71.28885	269657.4	122.00383
Modular addition	88.3	0.94868	88.6	1.26491
Montgomery multiplication	1771.4	40.81993	1783.2	41.13069
Montgomery inversion	1083382.7	402.05557	1083054.1	669.04251

2.7 Leakage-Resilient Message Authentication Codes

Side channel leakage (e.g., via timing, power or EM side channels) enables the extraction of secret data out of cryptographic devices, as initially demonstrated by Kocher (et al.) in 1996 and 1999 [K96], [KJJ99]. The engineering community reacted quickly by developing a variety of countermeasures that are commonly described as masking and hiding (see [MOP08]). Such countermeasures intend to reduce the overall exploitable leakage via techniques that are cheap to implement.

In this section we evaluate a practical Message Authentication Codes (MAC) scheme (inspired by the bilinear ElGamal cryptosystem by Kiltz and Pietrzak [KP10]), which is provably secure against continuous leakage model under the Only Computation Leaks Information (OCLI) assumption.

A message authentication is a technique that protects message integrity, i.e., it should detect any message modifications during a communication process between a sender and a receiver. As described in D2.3 [RD2.3, Section 6.11.1.1], such an Integrity Generator / Verifier mechanism is required in order to offer a certain security feature for the RERUM architecture. It is worth noticing that the integrity protections might be also used as underlying component of D2D Authenticator (as described in Section 3.8.2). In general, the message integrity can be achieved by applying both public and private-key schemes. In the latter case, the message integrity can be achieved with use of digital signatures, in the former case, to ensure integrity one can use, i.e., MAC. The details of the proposed leakage resilient MAC are presented in D3.2, [MOSW15]. Unfortunately, similarly to other keyed cryptographic primitives they might leak secret information via side-channels. Thus considering the mentioned problem, a novel leakage resilient MAC scheme is proposed that is designed to tolerate some leakage.

2.7.1 Relevance to RERUM's Use-Cases

The proposed MAC's design might be considered for practical applications. Indeed, it improves security of employed protocols especially in embedded systems, where side-channels are more likely to be exploited. Improving security of underlying primitives and protocols would also lead to a better privacy for RERUM users. More specifically in terms of RERUM's Use-Cases, the LR-MAC mechanism applies to all UCs that employ embedded hardware such as the Zolertia RE-Mote (both indoor UCs as well as UC-

O2 - Environmental monitoring).

2.7.2 Implementation details

As documented in greater detail in D3.2 [RD3.2], a MAC scheme can be defined as a tuple that consists of three algorithms:

$$M = (\text{MAC.KeyGen}, \text{MAC.Tag}, \text{MAC.Ver}),$$

where:

- MAC.KeyGen is a probabilistic algorithm generating a suitable key K; we denote this by $K \overset{R}{\leftarrow}$ MAC.KeyGen().
- MAC.Tag is probabilistic algorithm taking as an input key K, message m and generating tag σ; we denote this by $\sigma \overset{R}{\leftarrow}$ MAC.Tag(K, m).
- MAC.Ver is deterministic algorithm taking as an input secret key K, tag σ, message m and outputting boolean variable whether the tag is correct; we denote this by $b \leftarrow$ MAC.Ver(K, m, σ).

In addition, we require (for correctness) that for all valid keys K the following equation holds:

$$\text{MAC.Ver}(K, \text{MAC.Tag}(K, m), m) = 1$$

2.7.2.1 Bilinear MAC scheme

Before we describe a leakage resilient MAC, we introduce the bilinear maps [GPS08] and the bilinear MAC scheme, on which the leakage resilient MAC is based. Let G_1, G_2 and G_3 be a cyclic groups all of prime order p with generators g_1, g_2 and g_3 respectively. The bilinear map is a function e : $GG_1 \times G_2 \rightarrow G_3$ that holds both bilinearity, i.e., $\forall u \in G_1$, $v \in G_2$, $a, b \in Z_p : e(u^a, v^b) = e(u, v)^{ab}$ and non-degeneracy $e(g_1, g_2) \neq 1$ properties. Following the general definition of the MAC scheme the bilinear MAC is defined as in Figure 18. In the bilinear MAC scheme, the key generation algorithm MAC.KeyGen returns a random element of the group G_1, i.e., K. In the second algorithm, i.e., in a tag generation MAC.Tag part, the first step hashes the message using a hash function that transforms a message of an arbitrary length into an element of the group G_2, i.e., $H : \{0, 1\}^* \rightarrow G_2$. In the second step of the Tag a bilinear map is applied, in which the first input is the previously generated key K and a hash form the first step. This step generates the desired tag σ on the output. The third algorithm, i.e., the verification algorithm MAC.Ver takes as inputs the key K, the tag T, the message m and reassembles the Tag procedure to generate a new tag T'. In the last step MAC.Ver algorithm simply checks the correctness of the newly generated tag T'.

```
proc KeyGen():
    $K \overset{\$}{\leftarrow} \mathbb{G}_1$
    Return $K$
```

```
proc Tag(K, m):
    $W \leftarrow H(m)$
    $T \leftarrow e(K, W)$
    Return $T$
```

```
proc Ver(K, T, m):
    $W \leftarrow H(m)$
    $T' \leftarrow e(K, W)$
    Return $T' = T$
```

Figure 18: Bilinear MAC scheme [MOSW15].

2.7.2.2 Leakage-Resilient MAC

Having defined the key update mechanism we can now introduce the leakage resilient MAC scheme as it is shown in Figure 19. Following the general definition of MAC, the leakage resilient version consists of three algorithms: key generation MAC.KeyGen (also referred to simply as KeyGen), tag generation MAC.Tag (also referred to simply as Tag) and verification MAC.Ver (also referred to simply as Ver). The key generation mechanism MAC.KeyGen generates two shares by applying the following procedure: firstly, the random element of the group G_1 is generated together with a random share (which is also an element of the group G_1). Furthermore, the second share is calculated by multiplying the preciously generated *K* by the multiplicative inverse of the first share and both shares are returned. The tag generation procedure Tag can be roughly split onto two main parts, i.e., Tag◓ and Tag◒. The first sub-procedure Tag◓ takes the first share of the key and the message as the input. Then the hash function of the message is calculated, followed by the bilinear mapping, in which the input, the first share and the hash is used. The mapping generates the first share of the tag. The rest part of the sub-procedure performs the share update, in which the next share of the key is generated. The second sub-procedure of the tag generation, i.e., Tag◒ performs similar steps: a bilinear mapping that takes as input the second share and the second share update for the next tag generations. The output tag *T* is a multiplication of share tags obtained in bilinear mapping steps. The verification procedure Ver is the same as the bilinear verification procedure in bilinear MAC. The security definition only allows to leak on Tag and do not allow to leak on key generation, since this would leak the original key.

The practical implementation of the leakage resilient MAC scheme might be realised using the Barreto-Naehrig (BN) [BN06] family of paring-friendly curves. The provisional evaluation results show that the scheme might be implemented on constrained devices, i.e., on RERUM devices.

```
proc KeyGen():
$K \xleftarrow{\$} \mathbb{G}_1$
$S_0^{◓} \xleftarrow{\$} \mathbb{G}_1$
$S_0^{◒} \leftarrow K \cdot (S_0^{◓})^{-1}$
Return $(S_0^{◓}, S_0^{◒})$
```

```
proc Tag◓($S_i^{◓}, m$):
$W \leftarrow H(m)$
$t_i^{◓} \leftarrow e(S_i^{◓}, W)$
$r_{i+1} \xleftarrow{\$} \mathbb{G}_1$
$S_{i+1}^{◓} \leftarrow S_i^{◓} \cdot r_{i+1}$
Return $(S_{i+1}^{◓}, r_{i+1}, t_i^{◓}, W)$
```

```
proc Tag◒($S_i^{◒}, t_i^{◓}, W, r_{i+1}$):
$t_i^{◒} \leftarrow e(S_i^{◒}, W)$
$S_{i+1}^{◒} \leftarrow S_i^{◒} \cdot r_{i+1}^{-1}$
$T \leftarrow t_i^{◓} \cdot t_i^{◒}$
Return $(S_{i+1}^{◒}, T)$
```

```
proc Ver($K, T, m$):
$W \leftarrow H(m, w)$
$T' \leftarrow e(K, W)$
Return $(T' = T)$
```

Figure 19: Leakage Resilient MAC [MOSW15].

2.7.3 KPIs

The KPIs that will be measured in this section are the execution times of random point generation methods and of the various operations used by LR-MAC.

2.7.4 Experimental setup

All algorithms in our scheme were implemented using the MIRACL software library [PSNB10], which is a portable C/C++ library that supports a wide-range of different platforms including embedded ones. The advantage of the selected library is the extensive support for highly efficient pairing operations. In addition, we extended and adopted functionality provided by the library to our particular case, whenever required. For the underlying pairing operations, MIRACL uses the well-known Miller algorithm [M05]. Furthermore, it also applies the Galbraith-Scott method [FPS12], which allows computing a mapping between the groups G_2 and G_3 efficiently. All mentioned optimisation strategies increase efficiency of pairing computations, thus speeding up our proposed scheme and making it suitable for more resource-constrained environments.

We implemented and measured execution times of the schemes on both an embedded platform and a high-end device:

- For the former case, as a target platform, we selected a popular STM32F4 Discovery[2] board, which houses 32-bit ARM Cortex-M4 CPU. This device is more powerful than the Zolertia RE-Mote (Cortex M4 instead of M3, 2x flash size, 6x RAM size) and was chosen in order to mitigate the risk of being unable to implement the scheme for the target platform due to code size restrictions. As we see in the remainder of this section, experimental results are very promising. We believe that implementing LR-MAC scheme on the Zolertia RE-Mote will be feasible, provided some code size optimisations are undertaken. In terms of execution times, the two micro-controllers (ARM Cortex M3 and M4) are fundamentally very similar. The former uses the ARMv7-M architecture, whereas the latter uses ARMv7E-M. The differences between the two architectures are of an incremental nature: Both have the same instruction set, with the ARMv7E-M offering Digital Signal Processing (DSP) extensions and an optional single-precision Floating Point Unit (FPU). Taking those under consideration, we anticipate execution times on the M3 to remain within deployable rages. This investigation remains part of our future work.
- For the latter case, we utilised the 64-bit Intel Core i7 CPU. The internal clock of the Cortex-M4 was set to available maximum, i.e., 168MHz, whereas the Intel i7 ran with a 3GHz clock. We ran our benchmarks several times to derive median timings.

2.7.5 Performance evaluation

In this section we present the performance evaluation of the leakage resilient MAC which might be used in practical applications.

```
Random_Sampler:
loop
    $x \xleftarrow{\$} \mathbb{F}_p^*$
    if $x \in E(\mathbb{F}_p)$ then
        $i \xleftarrow{\$} \{0,1\}$
        $y \leftarrow -1^i \cdot \sqrt{x^2+b}$
        Return $R = (x, y)$
    end if
end loop
```

```
Try_and_Increment:
$x \xleftarrow{\$} \mathbb{F}_p^*$
loop
    if $x \in E(\mathbb{F}_p)$ then
        $i \xleftarrow{\$} \{0,1\}$
        $y \leftarrow -1^i \cdot \sqrt{x^2+b}$
        Return $R = (x, y)$
    else
        $x \leftarrow x + 1$
    end if
end loop
```

```
Random_Scalar:
Let $P$ be a generator of $E(\mathbb{F}_p)$
$k \xleftarrow{\$} \mathbb{F}_p^*$
return $R = [k]P$
```

```
SWEncoding:
$t \xleftarrow{\$} \mathbb{F}_p^*$
$w \leftarrow \sqrt{-3} \cdot t/(1+b+t^2)$
$x_1 \leftarrow -1 + \sqrt{-3}/2 - tw$
$x_2 \leftarrow -1 + x_1$
$x_3 \leftarrow 1 + 1/w^2$
$r_1, r_2, r_3 \xleftarrow{\$} \mathbb{F}_p^*$
$\alpha \leftarrow \chi_p(r_1^2 \cdot (x_1^3 + b))$
$\beta \leftarrow \chi_p(r_2^2 \cdot (x_2^3 + b))$
$i \leftarrow [(\alpha - 1) \cdot \beta \mod 3] + 1$
$R \leftarrow (x_i, \chi_p(r_3^2 \cdot t) \cdot \sqrt{x_i^3 + b})$
Return $R$
```

Figure 20: The algorithms for the four point samplers considered.

Generating random group elements securely is vital for key generation and key updates (which happen in TAG procedure). We found in [MOSW15] four options (see Figure 20 for algorithmic descriptions) for this purpose, which we now discuss in turn. The first one, the Random_Sampler procedure, randomly selects an *x*-coordinate and checks if it is on the BN curve. In case of success, the procedure computes an associated *y*-coordinate, otherwise randomly selects another *x*-coordinate. The second

[2]http://www.st.com/web/en/catalog/tools/FM116/SC959/SS1532/LN1848?icmp=ln1848_pron_pr-stm32f446_dec2014

key generation procedure, Try and Increment is very similar to the first one. It differs only in re-selection of *x*-coordinate in which the procedure increments *x*-coordinate by 1 and repeats the assessment of whether a new *x* is on the curve. The third one, Random_Scalar selects a scalar at random and performs a scalar multiplication using a fixed group generator. The last procedure uses the encoding to BN curve [FT12], where a random element $t \in F_p$ is transformed into an element of the curve $E(F_p)$, which was used by Galindo et al. [GGLVV14]. Note that when using this encoding one has to perform it twice in order to generate a point distributed uniformly at random [GGLVV14]. Hence, the timings are effectively twice as long in practice.

2.7.5.1 Execution time

For a fair comparison of timings, all procedures were implemented without blinding. Applying blinding to the Random_Sampler, Try_and_Increment and Random_Scalar methods requires one additional multiplication. A more involved blinding method for the BN encoding has been proposed in [FT12]. Table 31 shows that Random_Sampler and Try_and_Increment are by far the most efficient (the SWEncoding method needs to be performed twice for uniformly distributed points), and it is clear that this advantage would also hold when blinding is included. As the Random_Scalar method is not only slow, but also known to be very vulnerable to power analysis attacks [C99].

Table 31: Performance comparison of random point generation methods.

Device	**Cortex-M4**		**Intel i7**	
Operation	**Time (usec)**		**Time (usec)**	
	128-bits	**192-bits**	**128-bits**	**192-bits**
Random_Sampler	36,000	588,000	96	1,159
Try_and_Increment	34,000	569,000	76	1,119
Random_Scalar	173,000	2,827,000	389	5,186
SWEncoding	30,000	616,000	121	1,217

Attack vectors: It has been shown practically that the Random_Scalar method can completely leak the entire secret randomness and hence strictly speaking, it cannot be used for schemes proved secure in the continual leakage model. Security aspects of the SWEncoding scheme have been discussed by Galindo et al. [GGLVV14]. They conclude that the SWEncoding might leak via the Jacobi symbol. The security of the other two methods with respect to the continual leakage model has not yet been investigated. Hence we will now discuss the security considerations here.

The Try_and_Increment procedure will leak information about the number of increments via its overall execution time (which will be visible from power or EM traces). It is not obvious how this information could be utilised efficiently. However, it will contribute to the amount of leakage per call for the λ security bound [MOSW15].

The Random_Sampler method chooses values for *x* independently of previous choices and hence does not leak any additional information on the *x* from its high-level functionality. The only part, which may reveal information about the point is the calculation of the Jacobi symbol.

2.7.6 Overview of the overall performance

Lastly, we give an overview of the performance of the high level functions of the MAC constructions in Table 32. The basic MAC scheme had identical operatiosa ns in the MAC.Tag and MAC.Ver procedures (bar the additional equality check in MAC.Ver which is extremely fast), hence the resulting identical timings.

Table 32: Performance comparison (Bilinear and LR MAC).

Device	**Cortex-M4**		**Intel i7**	
Operation	**Time (usec)**		**Time (usec)**	
	128-bits	**192-bits**	**128-bits**	**192-bits**
MAC.Tag	2,146,000	30,317,000	7,935	68,692
MAC.Ver	2,146,000	30,317,000	7,958	68,687
*KG**	72,000	1,126,000	170	2,128
*TAG**	4,059,000	57,274,000	15,473	130,612
VRFY = MAC.Ver*	2,146,000	30,317,000	7,958	68,687
Share	34,000	566,000	94	1,082
TAG◓	2,183,000	30,883,000	7,974	69,650
TAG◒	1,874,000	26,382,000	7,162	60,841
Recombine	2,000	11,000	<1	<1

Switching then to the leakage resilient version, it is clear that the cost essentially doubles for the MAC.Tag computation. Since we had to assume MAC.Ver was not leaking (recall that our construction, like other MAC constructions requires the reconstruction of MAC.Tag during MAC.Ver, which is seemingly impossible to do securely allowing adaptive adversaries in the continual leakage model), the timings for MAC.Ver in the leakage resilient scheme are the same as for the scheme without leakage.

2.8 RSSI-based CS encryption keys

In RERUM Deliverable 3.1 [RD3.1] we described an algorithm for on-the-fly secret key generation to be used in a Compressive Sensing (CS) encryption scheme. CS has the advantage of simultaneous data encryption and compression. Regarding its encryption functionality, CS operates as a symmetric block cipher where cipher-text $y \in R^M$ is produced by multiplying plaintext $x \in R^N$ $(M \ll N)$ with the measurement matrix $\Phi \in R^{M \times N}$, as shown below:

$$y = \Phi x$$

This forms an undetermined system that can be solved using a number of optimisations algorithms like the OMP algorithm [TRGI07]. The measurement matrix $\Phi \in R^{M \times N}$ plays the role of the encryption key.

The key generation scheme we proposed is based on the RSSI (Received-Signal-Strength-Indicator) having a number of advantages that make it suitable for use by resource constrained devices like the sensors: (i) it does not require any key pre-storage, (ii) it does not require any key distribution mechanism, and (iii) keys can be updated dynamically. Briefly, the proposed scheme consists of the following phases:

- **RSSI sampling**: RSSI measurements are periodically collected by the sensors.
- **Quantisation**: The collected RSSI samples are quantised using a pre-defined number of bits/sample.
- **Key uniformity**: CS performance (in terms of the reconstruction/decryption error) minimises when matrix Φ is generated based on a uniform distribution. To achieve this, quantisation data are hashed using a secure hashing function that generates a uniform distribution.
- **Information reconciliation:** One of the communicating parties (sensors) creates a secure hash from its generated RSSI key. This secure hash is then transmitted to the second sensor, which, based on its own hash, it chooses the most appropriate common secret key to be used for the CS encryption.

2.8.1 Network topology and software layout

In order to evaluate the key generation scheme, we developed a CoAP Resource within the RD Adaptor for the RSSI collection. As running a full implementation of the key generation algorithm on the sensors is currently infeasible, due to their resource constrained nature, a two-step approach is used: (i) we first collect the RSSI measurements, and (ii) then we execute and evaluate the algorithm offline. Figure 21 shows the network topology, and the main building blocks of the RSSI collection module. In this scenario, there are two RE-Mote RDs (S1 and S2) that collect RSSI measurements, while a third RD acts as a Border Router (BR), forwarding the collected measurements from the sensor network (IEEE 802.15.4) to a laptop that executes the key generation scheme.

The purpose of this scenario is to create a secret key between S1 and BR, but without S2 being able to correctly decrypt the data that flow between S1 and BR, using its own key (that is also derived from RSSI measurements). Here, we assume that S2 acts as a passive adversary that is fully aware of the key generation algorithm used, and which can successfully eavesdrop on the encrypted data exchanged between S1 and BR.

As the performance evaluation will be executed offline, we collect the RSSI measurements for S1-BR as follows:

- BR periodically transmits UDP packets to S1.
- Upon receiving a UDP packet from BR, S1 computes and stores the corresponding RSSI measurement for this packet.
- S1 sends a reply to the BR that contains the computed RSSI value along with the measurement ID of the original initial packet.
- When BR received this reply, it computes a second RSSI value.

For each packet BR transmits to S1, the nodes produce a pair of RSSI values (one for BR and another one for S1) along with the measurement ID. For example: {"id":1, RSSI_S1: "-78", RSSI_BR: "-80"}. Essentially, probe UDP packets are transmitted between S1 and BR to collect the RSSI values for that link. Each measurement is stored on the BR, and is exposed as a CoAP resource, thus making feasible its collection from a CoAP client that is located outside from the sensor network (Figure 21). Figure 22 shows a snapshot of RSSI measurements collected for the link of BR-S1 using this topology, and the developed software that is based on Contiki OS, while Figure 23 shows a snapshot of RSSI measurements collected for the link of BR-S2.

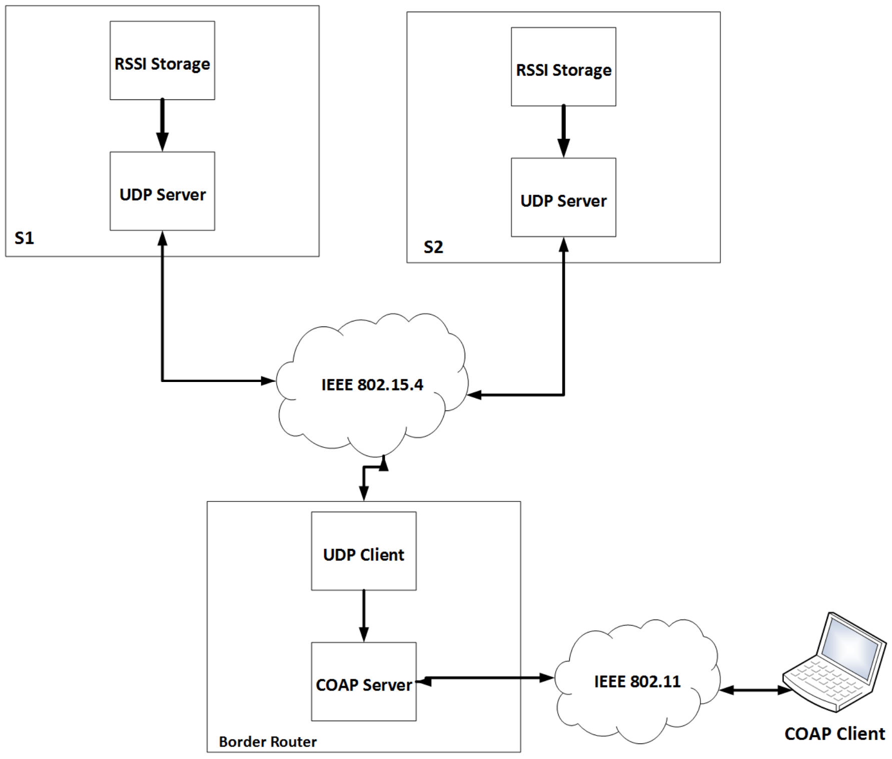


Figure 21: Software layout and network topology for secret key generation.

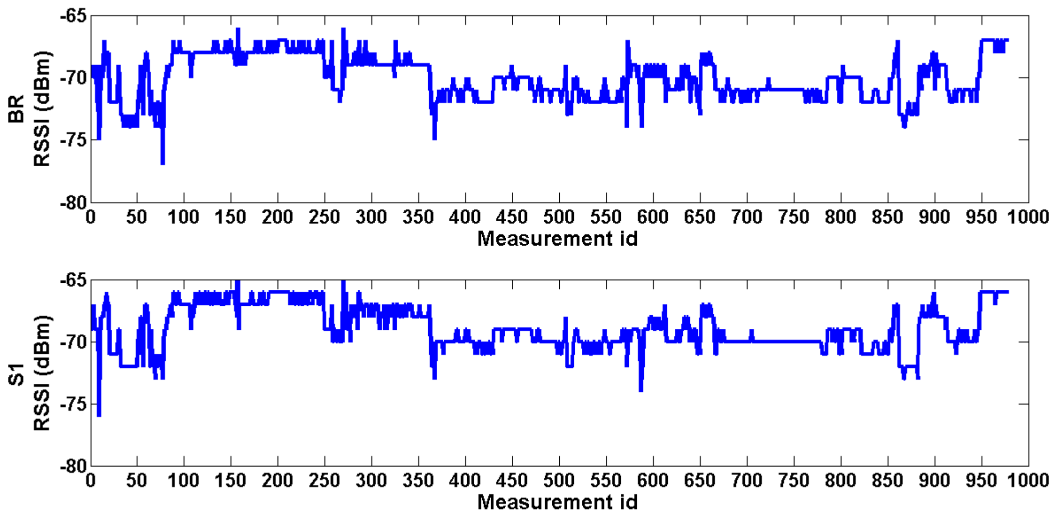


Figure 22: RSSI measurements for the link between BR and S1.

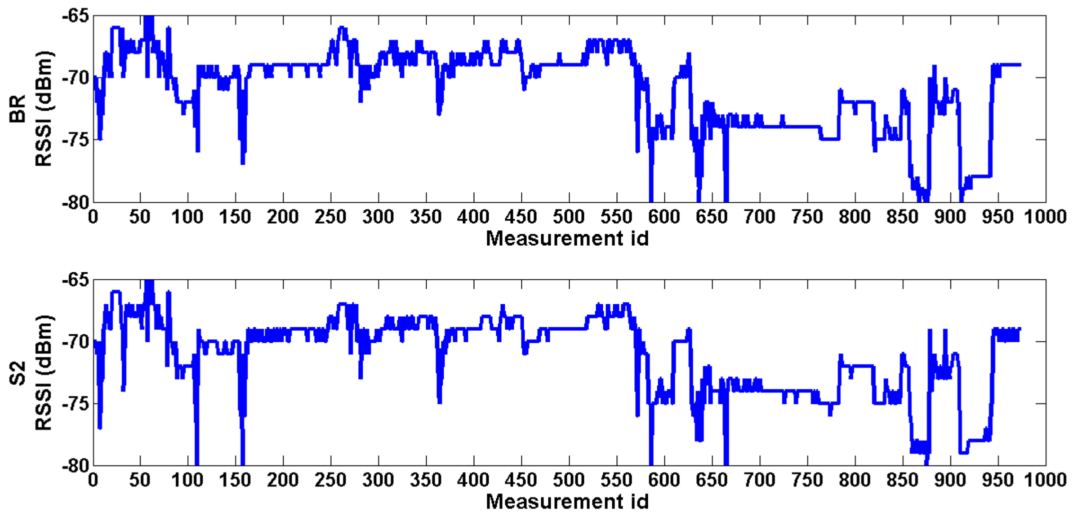


Figure 23: RSSI measurements for the link between BR and S2.

Note that in a real secret key generation implementation, each RD would store its own RSSI values, without the need to piggyback replies to BR. The probing mechanism would still be necessary in real scenarios as it is used to take advantage of the channel reciprocity: the multipath properties between a sender and a transmitter (gains, phase shifts, and delays) are identical on both directions of the communication link.

2.8.2 KPIs

The KPIs measured in this experiment are as follows (see also evaluation criterion AL.PE.6 in RERUM Deliverable D5.1 [RD5.1]):

- Bit mismatch rate between the encryption keys of the legitimate and the malicious devices.
- Reconstruction error at the two receivers.
- Time required to agree upon a common secret key.

2.8.3 Performance evaluation

We evaluate the secret key generation scheme in terms of the reconstruction (decryption) error, defined as $e = \frac{||x-\hat{x}||_2}{||x||_2}$, where x and $\hat{x}$ are the original and decrypted plaintexts, respectively; hence, this error expresses the fidelity between the original and the decrypted plaintext.

We use three RE-Mote RDs, deployed as shown in Figure 21, in an indoor lab environment. Sensors S1 and S2 are probed periodically by BR, and encryption keys Φ_1 and Φ_2 are generated based on the collected RSSI values, and for the optimum RSSI block derived after information reconciliation (described in [RD3.1]). As S2 is fully aware of the key generation scheme, and based on the optimum block id between S1 and BR it has overheard, it generates its key (Φ_2), and then it attempts to decrypt subsequent ciphertext data (assumed to be) transmitted from BR to S1. The cipher-text data are stored in the external PC we use for the offline performance evaluation.

After the RSSI measurements have been collected, and keys Φ_1 and Φ_2 are generated, we compute the reconstruction error when decrypting data that have been encrypted using CS, and for four different compression ratios: 25%, 50%, 75%, 90%. For S1, the data are encrypted and decrypted using the same key Φ_1, while for S2, data are encrypted using Φ_1 and decrypted using Φ_2. Unlike other symmetric encryption algorithms, where the encryption and the decryption keys have to be identical, in CS they can differ, as lossy compression is also involved. The smaller the difference between the encryption and the decryption key, the smaller the CS reconstruction error.

The following figures (Figure 24, Figure 25, Figure 26, and Figure 27) show the empirical cumulative density function (CDF) of the reconstruction error for the different compression ratios, and for different quantisation levels when encrypting/decrypting the S1 data.

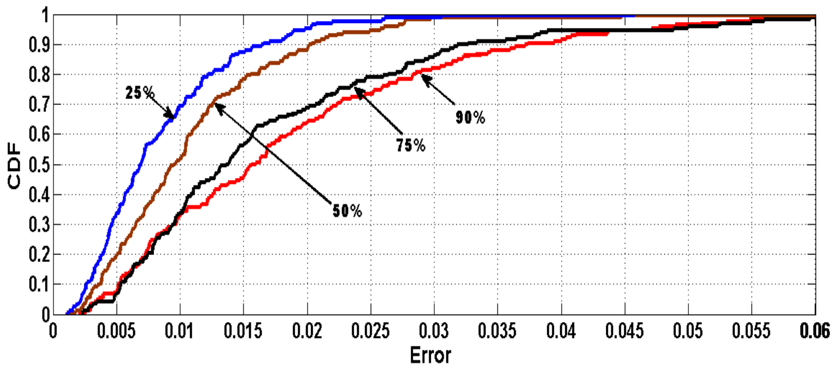


Figure 24: Error for S1 when using 1-bit quantisation.

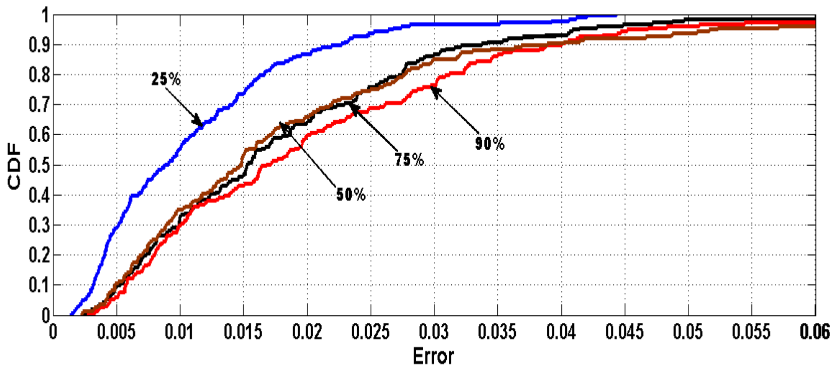


Figure 25: Error for S1 when using 2-bit quantisation.

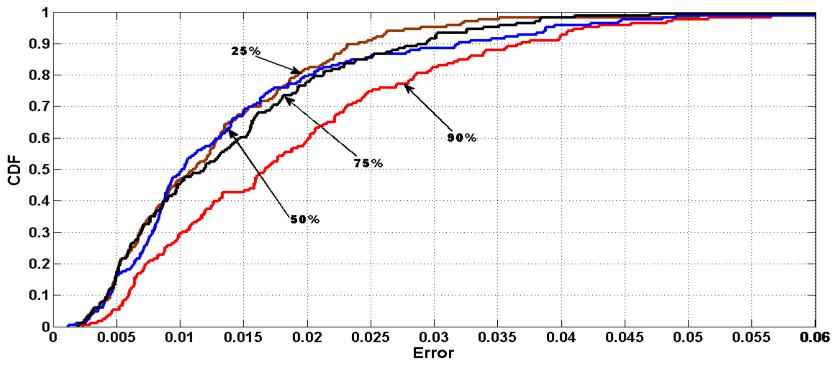


Figure 26: Error for S1 when using 3-bit quantisation.

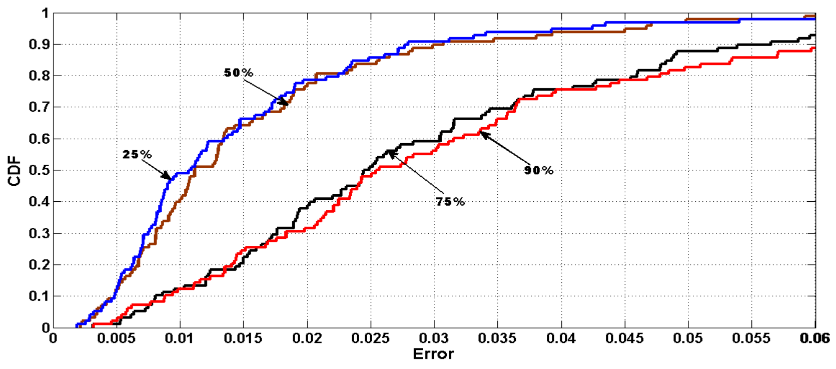


Figure 27: Error for S1 when using 4-bit quantisation.

Observe that as the compression ratio increases, the error increases, as expected in such a CS implementation. When the quantisation level increases, error increases because the bit-mismatch rate of the generated keys (between S1 and BR) increases. However, in all cases, the error remains very low.

We now present, in the following figures (Figure 28, Figure 29, Figure 30 and Figure 31), the reconstruction error for S2, that is, when using key Φ_2 to (maliciously) decrypt data that have been encrypted with key Φ_1.

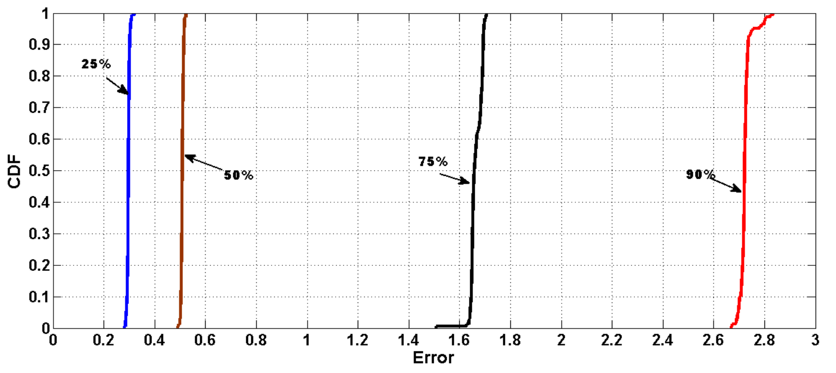


Figure 28: Error for S2 when using 1-bit quantisation.

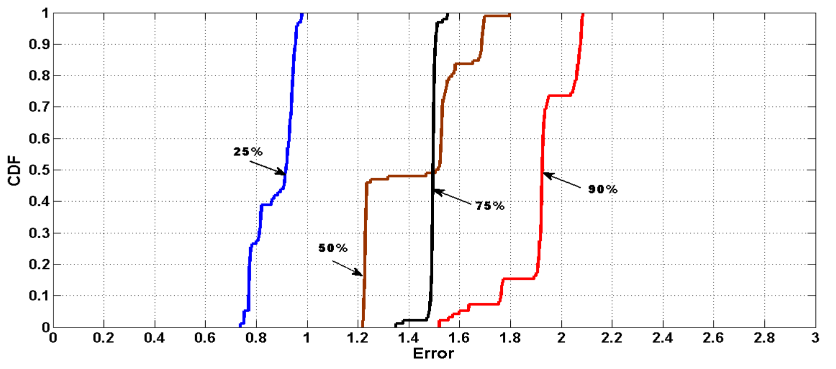


Figure 29: Error for S2 when using 2-bit quantisation.

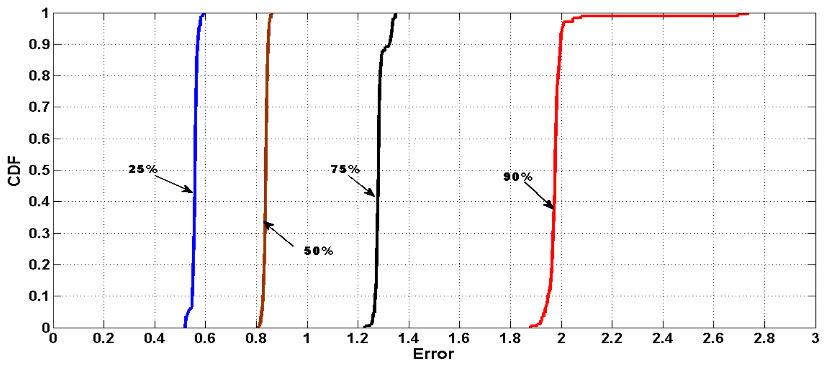


Figure 30: Error for S2 when using 3-bit quantisation.

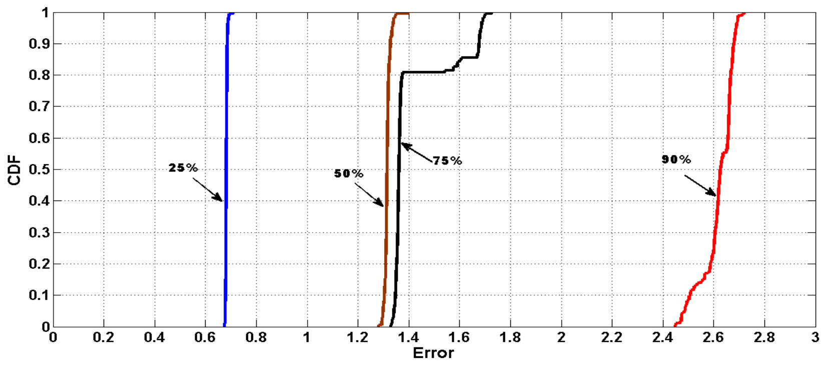


Figure 31: Error for S2 when using 4-bit quantisation.

Observe in the above figures that the reconstruction error is extremely high, regardless the quantisation level used. This is because S2, although is aware of the optimum block id of S1-BR, its generated key (Φ_2) is not able to correctly decrypt the ciphertext data. Figure 32 shows the CDF of difference, as a percentage, between keys Φ_1 and Φ_2 for all measurements. For most cases, the difference between the keys is at least 28%.

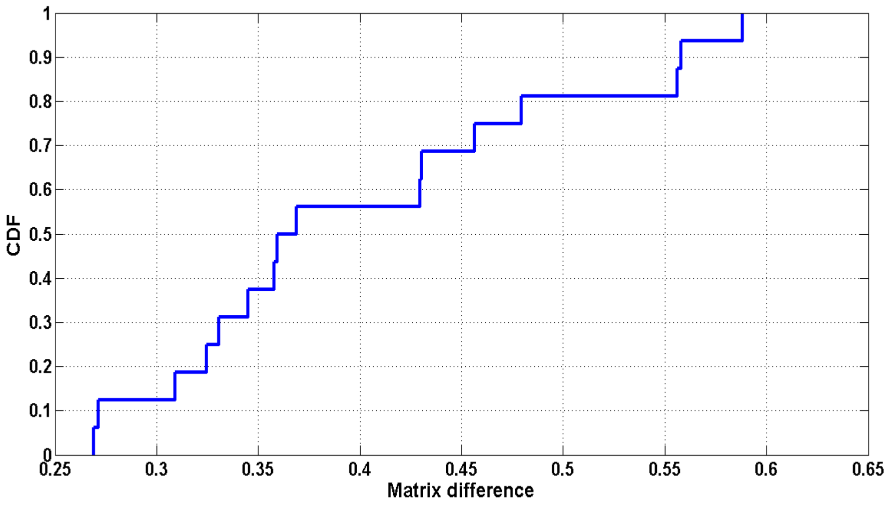


Figure 32: Matrix difference between Φ_1 and Φ_2.

3 Networking Lab Trials

In this section we shift our focus on the evaluation of networking lab trials. In particular, the section presents results on the performance of: i) the adaptive CS-based data gathering (Sec. 3.1), ii) the sensor self-monitoring component (Sec. 3.2), iii) the lightweight spectrum sensing component (Sec. 3.3), iv) the Cognitive Radio (CR)-based gateway (Sec. 3.4) and v) the BMFA multicast forwarding algorithm for 6LoWPANs (Sec. 3.5).

3.1 Adaptive CS-based data gathering

3.1.1 Purpose of the experiment

The purpose of this experiment is to evaluate the Adaptive Compressive Sensing (ACS) scheme in the lab environment. The ACS adapts its compression rate based on the estimated reconstruction error. So, if the error increases, then the compression rate is reduced, and vice-versa. A detailed description of the algorithm can be found in RERUM Deliverable D4.2 [RD4.2].

3.1.2 Experimental setup

The experimental setup consists of a RERUM Gateway (GW) communicating with a RE-Mote device, through a Zolertia (Z1) border-router. Initially, RERUM GW requests data without any compression. These data are later loaded in the RE-Mote and compressed using the ACS scheme, so as to achieve an accurate evaluation. The software developed for the RERUM GW and the RE-Mote are described in RERUM Deliverable D5.5 [RD5.5].

Initially, the RE-Mote device collects sensory data without performing any CS compression, and this data are stored in RERUM GW. In the next phase, the stored data are sequentially compressed using CS and transmitted to the RERUM GW. Later on, ACS operates, selecting the appropriate compression rate when signal sparsity (compressibility) changes.

3.1.3 KPIs

The KPIs measured in this experiment are as follows (see also evaluation criterion AL.EF.6 in RERUM Deliverable D5.1 [RD5.1]):

- The reconstruction error defined as $e = \|x - \hat{x}\|_2 / \|x\|_2$, where x and $\hat{x}$ are the original and reconstructed signals, respectively, and $\|.\|_2$ stands for the ℓ_2 norm.
- The detection probability that gives the number of sparsity detections over the total number of sparsity changes.
- The false alarm rate defined as the ratio of the false alarms over the total duration of the experiment in terms of the number of blocks.
- False negatives that are the number of missed detections over the total duration of the experiment.
- The average detection delay defined as $DD = \sum_{i=1}^{S} \Delta b(i)/S$, where S is the total number of sparsity changes, and $\Delta b(i)$ the interval, expressed in number of blocks, between the true sparsity change, and the corresponding sparsity change detection.

3.1.4 Performance evaluation

The ACS scheme uses a change-point detection algorithm (CPM), used to detect the sparsity changes. An important characteristic of this algorithm is the average run length (ARL) that expresses the average number of observations between two false positives. In the subsequent experiments, we use three values for the ARL: 100, 500, and 1000. Furthermore, we compare the evaluation of the ACS scheme with a non-adaptive one.

We begin the evaluation by showing in Figure 33 the cumulative density function (CDF) of the reconstruction error e, for the two approaches, and for the various ARL values. The solid curves correspond to the proposed adaptive scheme, while the dashed ones correspond to the non-adaptive strategy. It is clear that the adaptive scheme substantially outperforms the non-adaptive for all ARL values. Also, the more sensitive the CPM algorithm is, the less erratic the signal decompression is. This happens because of the decreased delay in the sparsity-change detection phase of the CPM that enables a fast adaptation of the compression rate. As a result, more than 90% of the blocks have a normalized reconstruction error of 0.01, for $ARL = 100$.

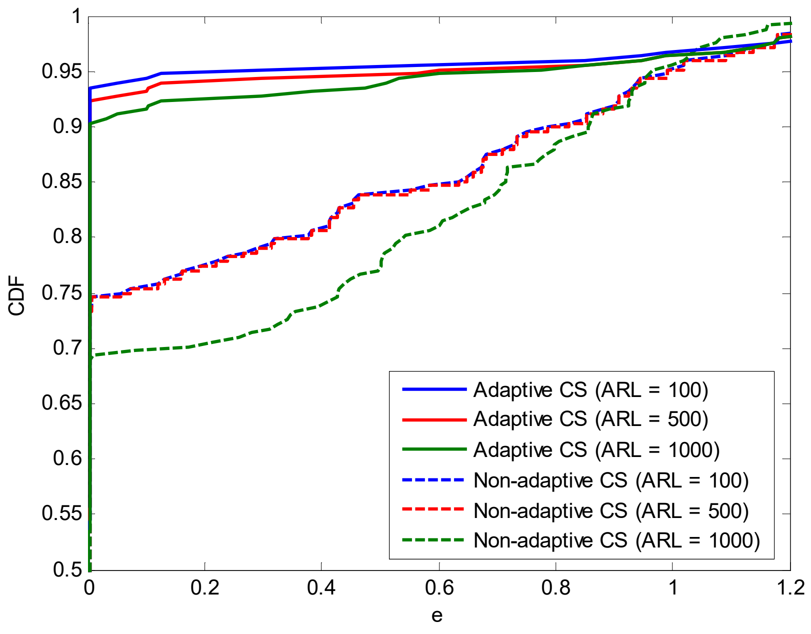


Figure 33: CDF of the reconstruction error for large intervals between the sparsity changes

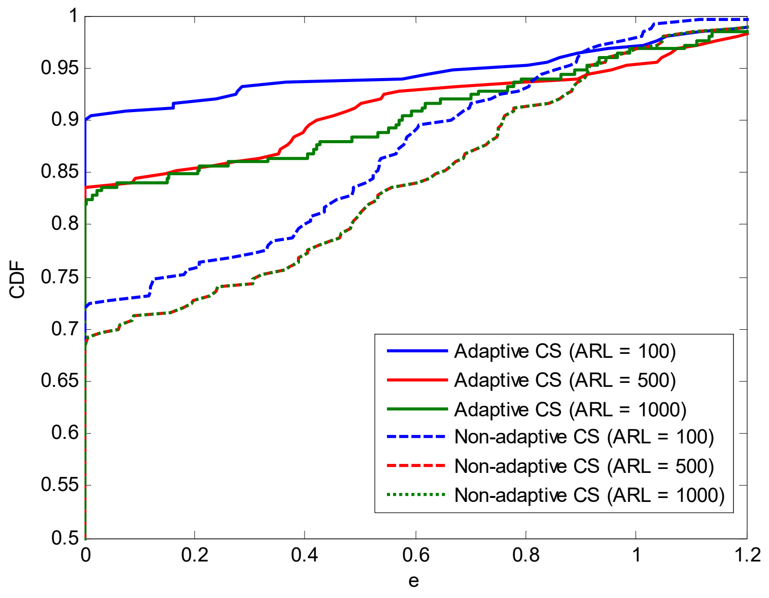


Figure 34: CDF of the reconstruction error for short intervals between the sparsity changes.

In order to investigate the behaviour of the proposed scheme for the frequent sparsity changes, we change the interval (in number of blocks) between two successive sparsity changes so that it is uniformly chosen in [10,15]. The CDF of e is presented in Figure 34 for ARL values in $\{100, 500, 1000\}$.

In this case, observe that the performance of the adaptive CS scheme degrades for all three values of ARL, especially for lower sensitivity of the CPM ($ARL = \{500,1000\}$). This is normal, since the CPM statistic is unable to converge to a flat value, due to the low number of initialization samples, resulting in an increased number of the false negatives. However, even in this case, more than 90% and 82% of reconstructed blocks, for $ARL = 100$ and $ARL = \{500,1000\}$, respectively, exhibit reconstruction error lower than 0.01. Additionally, ACS still outperforms the non-adaptive scheme that uses a constant compression rate for all compressed blocks.

In the next figures, we present the evaluation in terms of the detection probability (Figure 35), false alarm rate (Figure 36), false negatives (Figure 37), and the sparsity detection delay (Figure 38), considering two types of experiments: (i) EXP1 where there is a long interval between the sparsity changes, and (ii) EXP2, where this interval is short. As expected, as CPM sensitivity decreases, detection probability (Figure 35) also decreases, while the false negatives (Figure 36), and the detection delay (Figure 37) increase, for both experiments. For example, for EXP1, the detection probability is more than 94% for $ARL = 100$ and 80% for $ARL = 5000$, while the false negatives change from approximately 0.002 to 0.008 false negatives/data block. The false alarm rate (Figure 38) remains low for all cases, mainly due to the accurate sparseness of the synthetic signals.

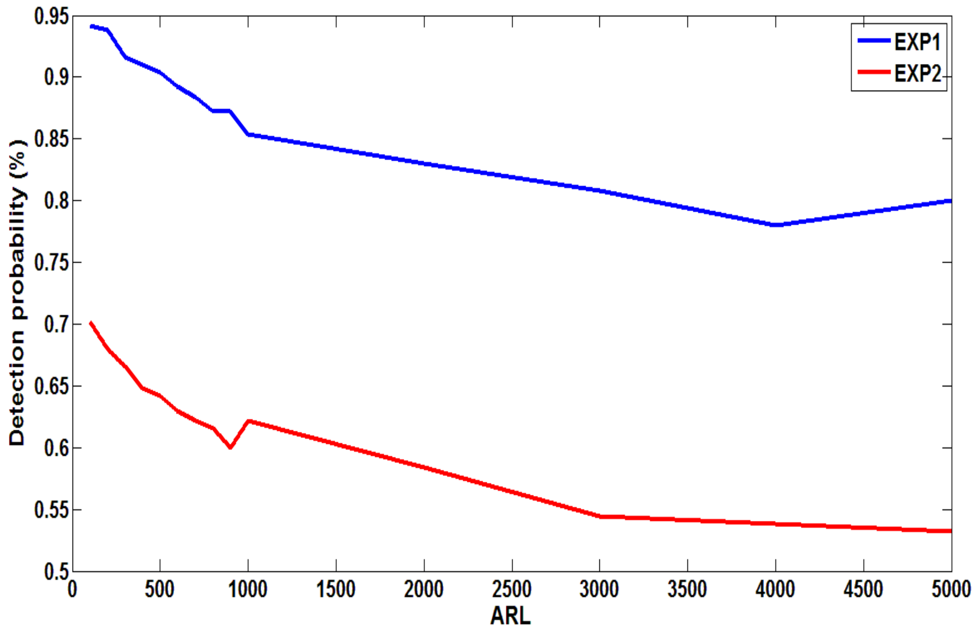


Figure 35: Sparsity change detection probability.

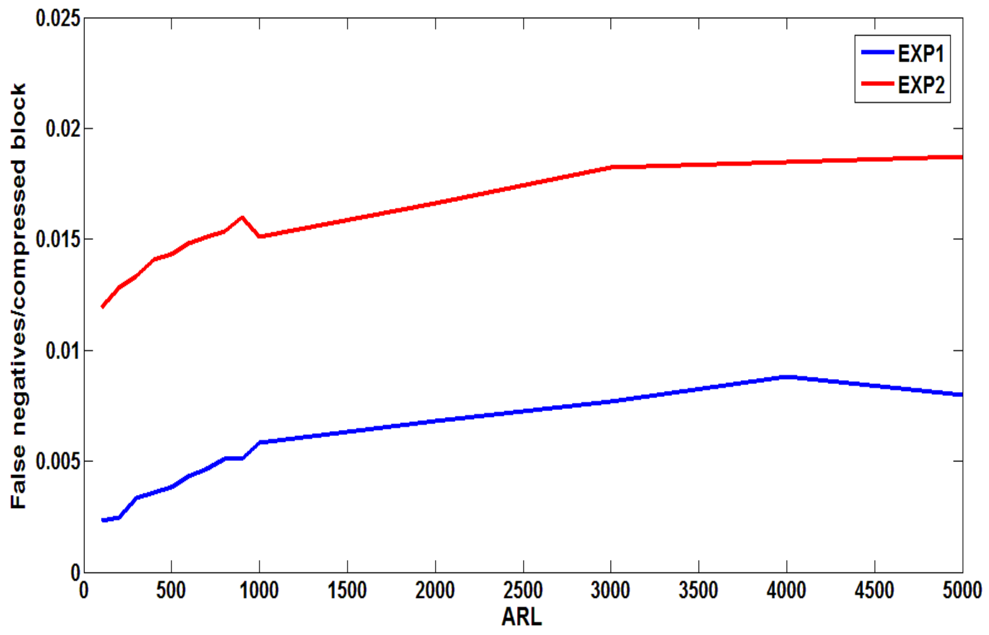


Figure 36: False negatives.

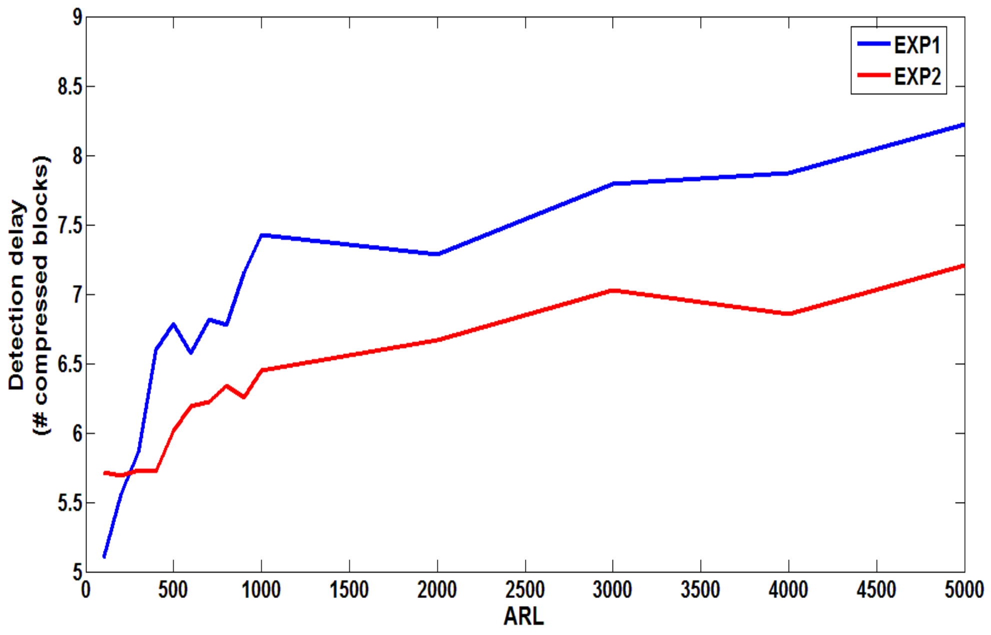


Figure 37: Sparsity change detection delay.

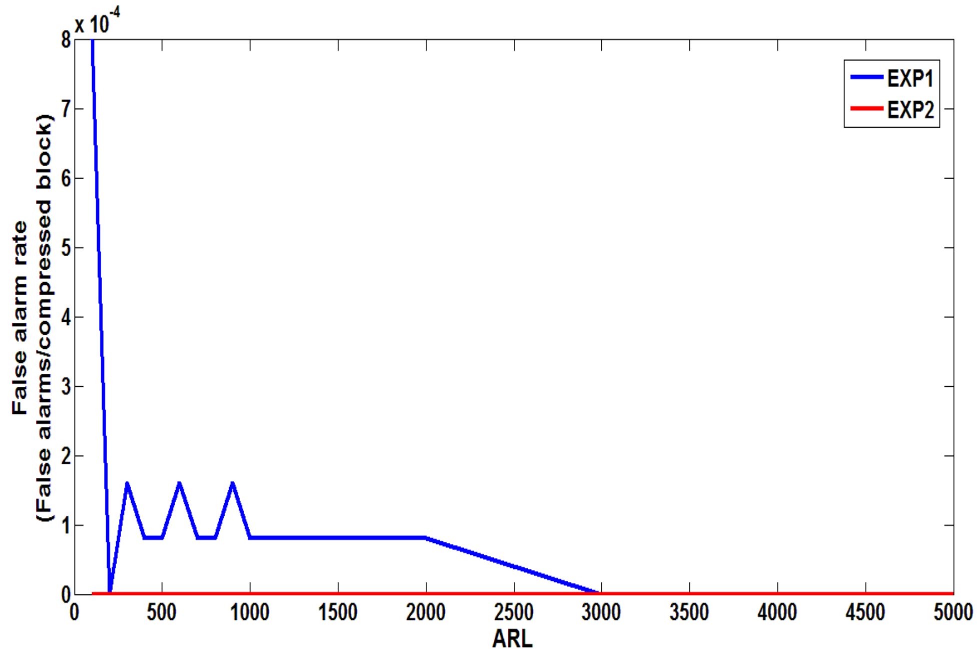


Figure 38. False alarm rate.

Short intervals between sparsity changes degrade the detection probability and increase the false negatives of the adaptive scheme, since the sample size for the CPM initialization after each sparsity change detection is rather small for the statistic to be accurately estimated. For example, the maximum detection probability (achieved in both experiments for $ARL = 100$) reduces from 94% to 70%, (Figure 35). As Figure 37 shows, there is a slight difference in the sparsity change delay between the two scenarios, with EXP1 requiring one data block more by the average to detect sparsity changes, for all the ARL values.

Finally in the next three figures (Figure 39, Figure 40 and Figure 41), we show the timeliness of the CPM algorithm when sparsity changes occur, for the various ARL values, and for EXP1 (we get analogous results for EXP2). Observe in the following figures that the delay for the CPM to react (denoted by the dotted lines), is relatively low (in terms of the number of blocks).

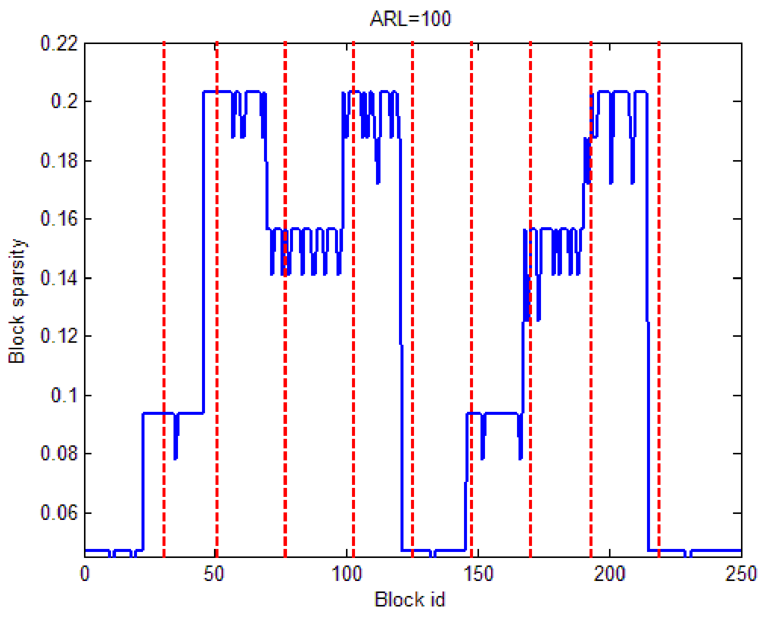


Figure 39: Sparsity change detection for ARL=100.

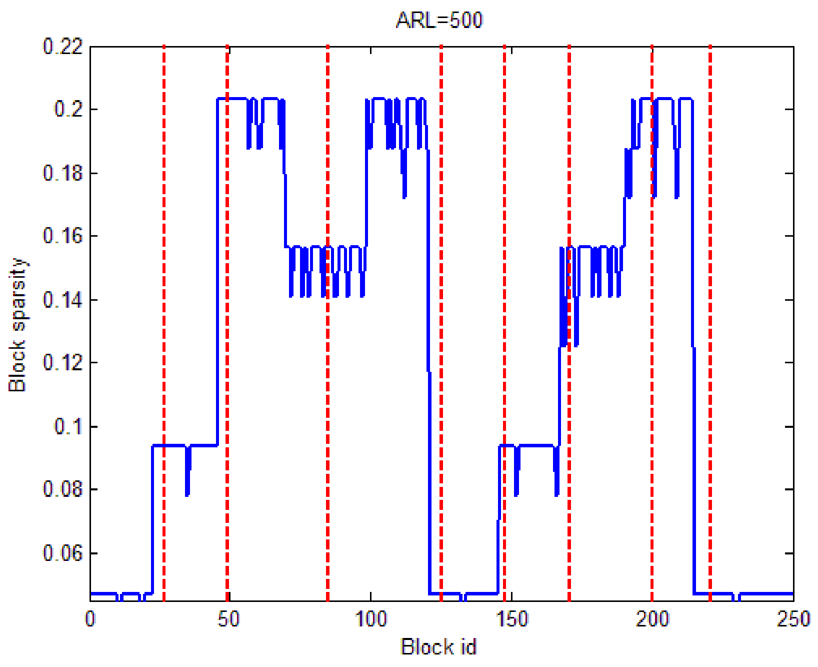


Figure 40: Sparsity change detection for ARL=500.

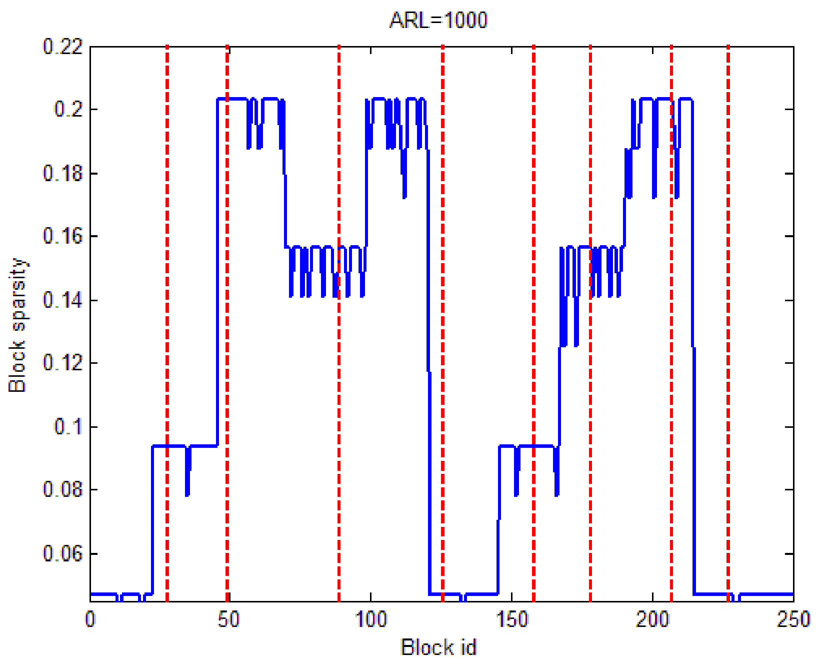


Figure 41: Sparsity change detection for ARL=1000.

3.2 Sensor self-monitoring

3.2.1 Purpose of the experiment

The goal of the experiment is to test in a real-world environment the performance of the self-monitoring component of RERUM. The theoretical description of the module was presented in RERUM Deliverable D3.1, while the implementation of the component on the RERUM Devices (RDs) is described in D5.2 and D5.5. The objective of the self-monitoring mechanism is to present a powerful component that will be able to gather in a simple and efficient way both network and device statistics from the RERUM Devices (RDs). These statistics are sent to the RERUM Gateway that forwards them to the centralized RERUM MW. There, the statistics are gathered by the Resource Monitor and the Alert Processor for identifying possible problems with the network mechanisms (i.e. channel assignment, routing) or with the devices themselves (i.e. device is shut down) and try to resolve the issues. In this respect, the devices gather both types of statistics and periodically sends them to the MW in a period that can be assigned remotely and can be either static or variable.

The experiment aims to be a first step of testing the mechanism on real devices and assess its efficiency before it is run on the trials. The mechanism is tested only on Zolertia RE-Motes that are playing the role of the RDs. The mechanisms to gather the statistics are implemented on Contiki and installed on the devices (see D5.2 and D5.5). The experiment focuses only on gathering and transmitting these measurements/statistics and not on the server-side exploitation of these statistics. Furthermore, the test includes the running of the mechanism for different network states to show the performance of the RD when the network is congested, when there is interference in the wireless channel, etc.

3.2.2 Implementation details

The module that gets the statistics and transmits them to the MW (or to the server) is implemented as a resource within the RD Adaptor and a Service is exposing this Resource on the MW. More details regarding the implementation of the self-monitoring component can be found in D5.2 and D5.5. Briefly, there are many statistics that can be measured via this component:

- Device statistics
- Network statistics
    - ICMP: number of received, dropped and sent packets.
    - IP: number of received, sent, forwarded and dropped packets.
    - Link layer: number of packets that are too long, too short, badly synced, with bad CRC, transmitted, received and number of collisions.
    - TCP: number of packets received, sent, dropped, reset, with acknowledgment errors, with checksum errors, retransmitted, number of synchronization errors and number of synchronization resets.
    - UDP: number of packets received, sent, dropped and with checksum error.
    - RPL: various statistics that are not of importance here, since the project does not work on modifying RPL.
    - ND6 (neighbour discovery): various statistics regarding number of sent, received and dropped packets.

3.2.3 KPIs

The KPIs that will be measured within this experiment are the following (see also evaluation criterion AL.EF.6 in RERUM Deliverable D5.1 [RD5.1]):

- Network statistics: the experiment involves the collection of link layer, IP layer and UDP layer statistics.
- Device statistics: only the chip temperature is collected, since the measurement of the energy consumption of the device can’t be done accurately with a software application.
- The communication overhead (increased signalling) for transmitting the statistics will be calculated. This KPI shows how much load the self-monitoring mechanism itself causes to the wireless spectrum.
- The performance of the device under various network loads and interference: The goal here is to show how the network measurements fluctuate under different types of network load and how responsive is the self-monitoring mechanism.

3.2.4 Experimental setup

The setup of the experiment is a very simple one as it can be seen in Figure 42. There is a RE-Mote device playing the role of the RD that we want to monitor. This device is connected wirelessly with a RERUM GW that is in turn connected to the RERUM Middleware. In the same area of the RD there are also standard WiFi networks operating, which may create interference and this is something that we want to exploit in order to collect the statistics under various amount of interference. In order to get the measurements, we used a Java Californium[3] program that plays the role of the CoAP client and accesses a CoAP Resource. The resulting data stream (with the network measurements) flows through the GW and the MW and is stored in a txt file for further processing. In a similar way, the actual network statistics can flow towards the MW's components, namely the Resource Manager and the Alert Processor in order to identify the current available network resources of the RD (and see i.e. how many requests it can handle). These statistics can also be used to raise alerts in case of i.e. high network load in order to execute mitigation procedures like channel re-assignment and change of routing tables.

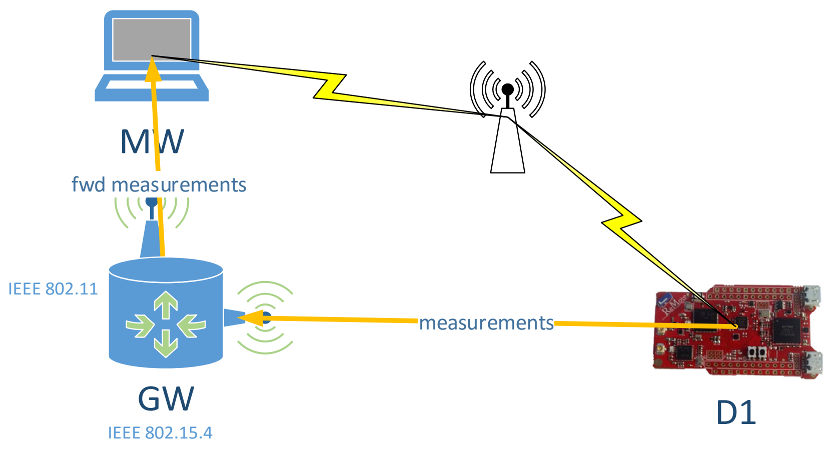


Figure 42: Experimental setup for network monitoring.

3.2.5 Evaluation results

The experiments were run both in a semi-controlled environment, and in an uncontrolled environment. In the first case, the load of the external WiFi access point that causes interference to the RD is controlled and is set in three different levels: low, medium, high. In these three experiments, the setup of the RD was to use an entirely different (not overlapping) channel (so that no interference is caused by the WiFi), to use an adjacent channel (so that low/medium interference in caused) and to use an overlapping channel, on which high interference is caused on the RD. In the case of the uncontrolled environment, the RD was tested at FORTH's TNL lab and the RD was using an overlapping channel with FORTH's public AP, which handles many users and has variable load. Furthermore, in order to ensure that there is also a continuous type of traffic from the RD, we used backbone ICMP traffic with packet interval of 2 seconds.

Below we present the results for these cases for various measurements. In Figure 43, we show the number of packets with link layer collisions, in Figure 44 the number of packets with bad CRC, in Figure 45 the number of transmitted and received link layer packets and in Figure 46 the IP layer transmitted

[3] http://www.eclipse.org/californium/

and received packets. As expected, when the external traffic is high, the monitoring component reports a lot of bad CRC errors and a lot of collisions due to the strong interference that the wireless link is suffering.

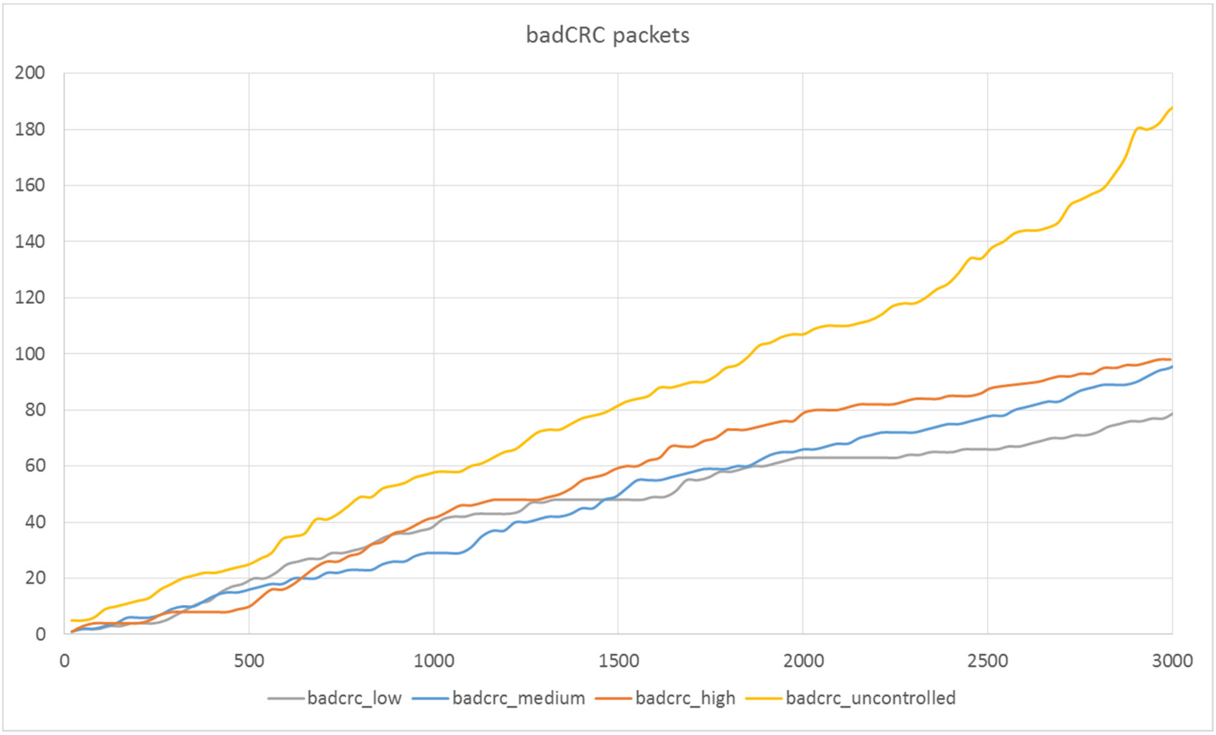


Figure 43: Packets with bad CRC under various load.

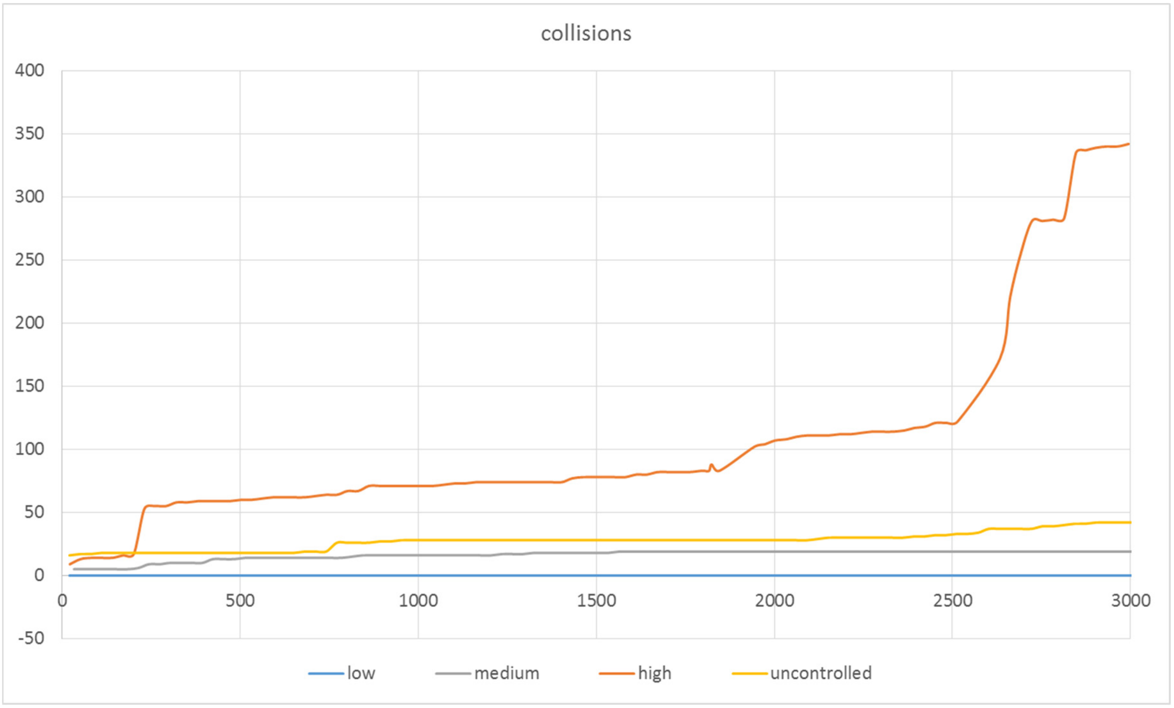


Figure 44: Number of collisions under various load.

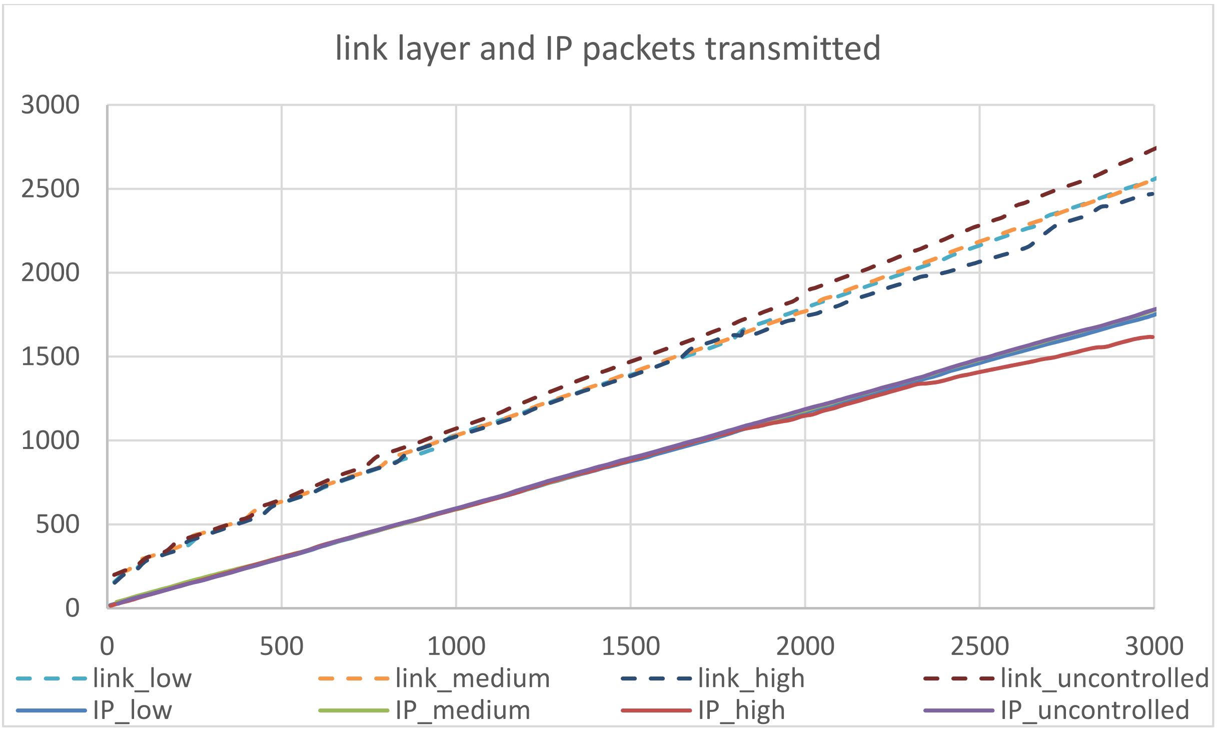


Figure 45: Number of link layer and IP packets transmitted under various load.

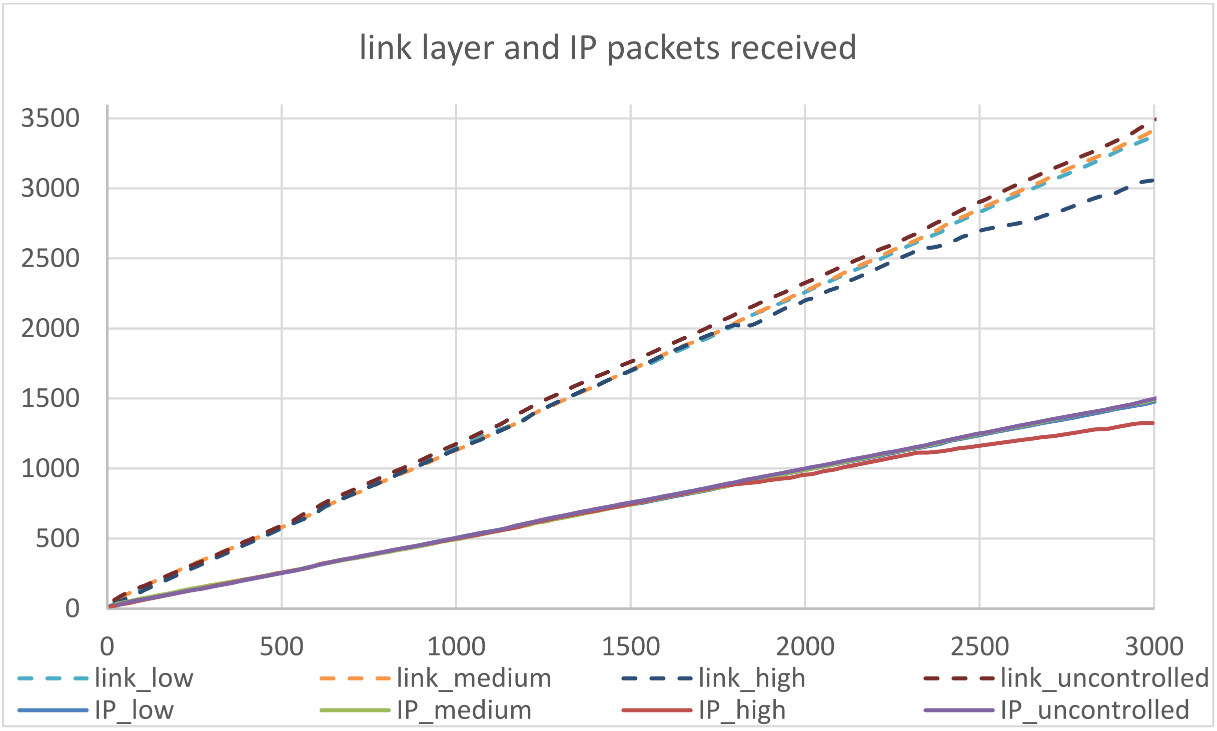
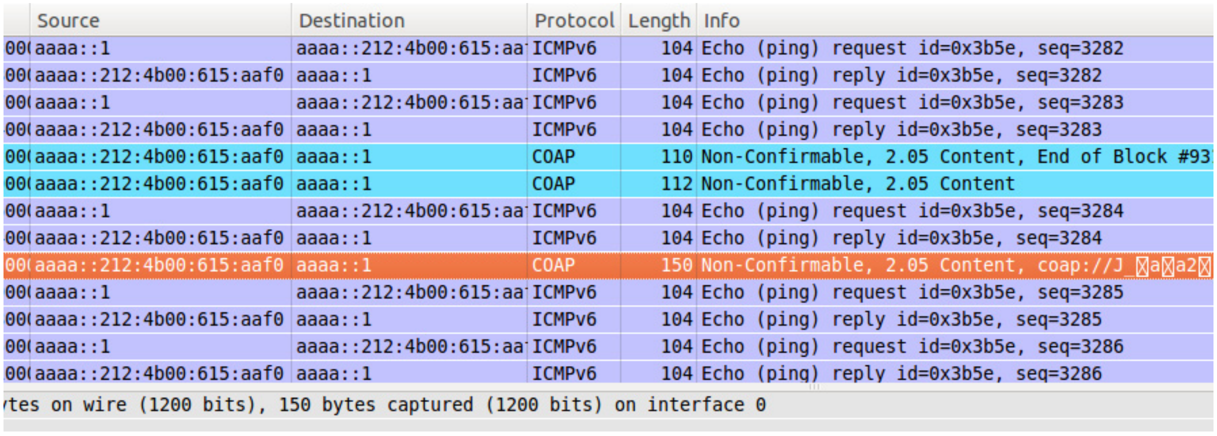


Figure 46: Number of link layer packets received under various load.

In Figure 47, we show a trace of Wireshark for the packets that are transmitted by the self-monitoring mechanism. In the figure, there are two types of packets, the ICMPv6 packets and the COAP packets. The first ones are the packets that are exchanged via the ping6 command (for the background traffic as explained before), while the COAP packets are those of the self-monitoring mechanism. As mentioned above, we are gathering three types of measurements, for link layer statistics, for IP statistics and for chip temperature. Hence, there are three COAP packets depicted in Figure 47. As it can be seen, these packets are very small with a size that varies between 110 and 150 bytes, which shows the lightweight characteristic of the self-monitoring mechanism.



```
    Source               Destination          Protocol Length Info
00(aaaa::1              aaaa::212:4b00:615:aa ICMPv6    104 Echo (ping) request id=0x3b5e, seq=3282
00(aaaa::212:4b00:615:aaf0 aaaa::1            ICMPv6    104 Echo (ping) reply id=0x3b5e, seq=3282
00(aaaa::1              aaaa::212:4b00:615:aa ICMPv6    104 Echo (ping) request id=0x3b5e, seq=3283
00(aaaa::212:4b00:615:aaf0 aaaa::1            ICMPv6    104 Echo (ping) reply id=0x3b5e, seq=3283
00(aaaa::212:4b00:615:aaf0 aaaa::1            COAP      110 Non-Confirmable, 2.05 Content, End of Block #93
00(aaaa::212:4b00:615:aaf0 aaaa::1            COAP      112 Non-Confirmable, 2.05 Content
00(aaaa::1              aaaa::212:4b00:615:aa ICMPv6    104 Echo (ping) request id=0x3b5e, seq=3284
00(aaaa::212:4b00:615:aaf0 aaaa::1            ICMPv6    104 Echo (ping) reply id=0x3b5e, seq=3284
00(aaaa::212:4b00:615:aaf0 aaaa::1            COAP      150 Non-Confirmable, 2.05 Content, coap://J_▯a▯a2▯
00(aaaa::1              aaaa::212:4b00:615:aa ICMPv6    104 Echo (ping) request id=0x3b5e, seq=3285
00(aaaa::212:4b00:615:aaf0 aaaa::1            ICMPv6    104 Echo (ping) reply id=0x3b5e, seq=3285
00(aaaa::1              aaaa::212:4b00:615:aa ICMPv6    104 Echo (ping) request id=0x3b5e, seq=3286
00(aaaa::212:4b00:615:aaf0 aaaa::1            ICMPv6    104 Echo (ping) reply id=0x3b5e, seq=3286
tes on wire (1200 bits), 150 bytes captured (1200 bits) on interface 0
```

Figure 47: Wireshark trace of self-monitoring packets.

Figure 48 shows the communication overhead of the self-monitoring mechanism over several experiments. The communication overhead is shown against the background traffic that is created via ping for different intervals of the ping packets from 0.5 to 4 seconds. The monitoring packets are sent every 30 seconds. This is the minimum interval allowed currently by the monitoring mechanism to avoid flooding the network with messages. We can see that the communication overhead rises to almost 25% when the ICMP packets are sent scarcely. In the cases when the ICMP packets are sent very frequently, the percentage of the monitoring packets is very low. In general, each monitoring statistic is sent with one CoAP packet every 30 seconds (in the worst case), which is not creating a significant load in the network. We can therefore assume that the self-monitoring mechanism is lightweight.

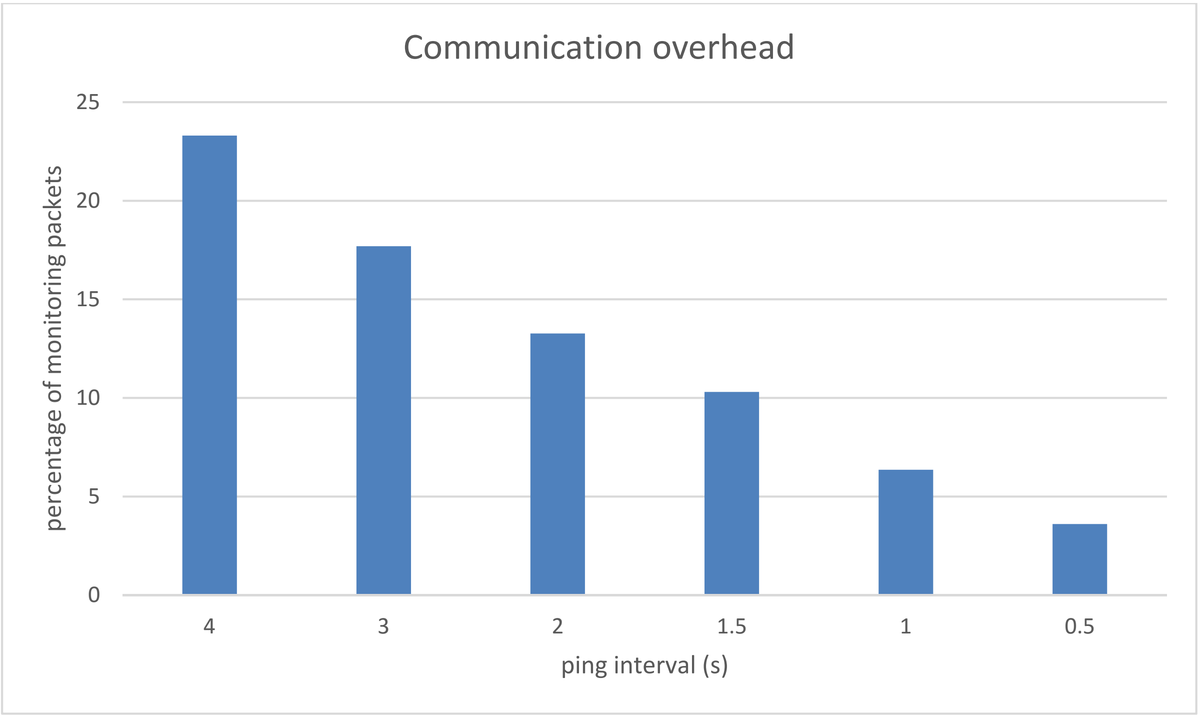


Figure 48: Communication overhead of self-monitoring.

3.3 Lightweight spectrum sensing framework

3.3.1 Purpose of the experiment

The goal of this experiment, as described in RERUM Deliverable D5.1 [RD5.1], is to evaluate in a controlled laboratory environment the efficiency of the lightweight spectrum sensing framework that was developed within Task 4.1. The target modules to be evaluated are those for gathering spectrum occupancy measurements and modelling the spectrum occupancy. The theoretical description and the simulation results of these mechanisms are presented in detail in deliverable D4.1. Here, the goal is to test the practical implementation of these mechanisms in a laboratory experiment, using real Software Defined Radio (SDR) devices.

As a brief reminder, the spectrum sensing module aims to enable the Cognitive Radio-based RDs to gather spectrum occupancy statistics in an energy efficient way, consuming as led energy as possible, while keeping the accuracy of the statistics on a good level. Then, the module will extract models of the spectrum occupancy at specific bands of interest. The minimization of the energy consumption by the RDs for spectrum sensing is achieved by identifying an “optimum” period for sensing each band. At the same time, we have to avoid sensing the bands very frequently because spectrum sensing is a process that consumes a lot of energy.

For the realization of the experiment scenario and since Cognitive Radio mechanisms cannot be run on existing sensor platforms, Software-Defined-Radio (SDR) devices and the GNU Radio platform were used. GNU Radio[4] is an open source platform that is part of the GNU project. The core of the platform and the demanding signal processing tasks are performed using C/C++, where tasks like input and output connections are accomplished using Python. The approach of GNU Radio signal processing is based on multiple data streams. Each stream may be the output of a processing unit called block or the input to another processing block. GNU Radio is compatible and supports multiple hardware SDR devices like Universal Software Radio Peripheral (USRP), BladerRF, HackRF and others. One of the most attractive features of this platform is that allows developers to add easily their own modules, extending the capabilities of the platform. Furthermore, the large, active and growing community makes GNU Radio and ideal choice of an SDR platform.

The following sections provide details for the implementation and evaluation of the Dyna-Q Reinforcement Learning algorithm described in D4.1 [RD4.1], utilizing the GNU Radio platform.

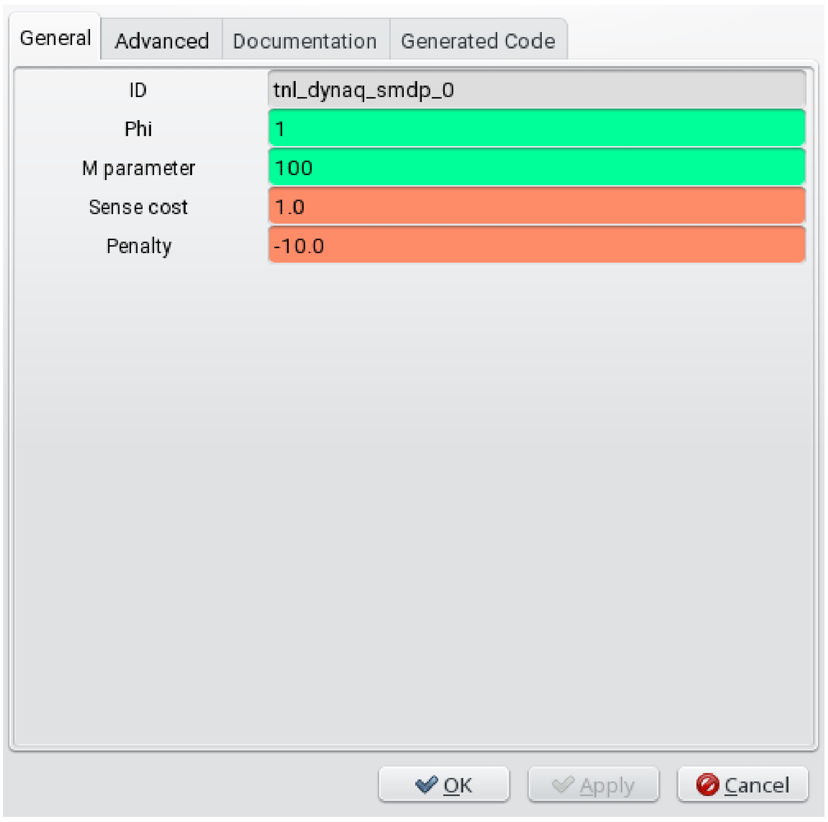
3.3.2 Implementation details

The GNU Radio block takes as input integer samples that correspond to the PU spectrum state at the i-th time-slot. The block is configured using a set of necessary parameters, depicted in Figure 49.

The ϕ parameter corresponds to a threshold that indicates the tolerance of the system with respect to the PU being in a different state than the one observed during the start of the observation period. The M parameter is the duration in time-slots of the largest available period. The sense cost parameter is the energy cost of the SDR device for sensing the spectrum at a single time-slot, whereas the penalty parameter represents the energy cost induced to a SU due to inaccurate collection of spectrum occupancy data.

As long as the block is provided with spectrum occupancy samples, the algorithm converges to the optimum spectrum period, targeting the best energy efficiency. When the user terminates the learning process, the block provides two different optimum sensing durations. The one corresponds to the optimum sensing period when the PU is not transmitting, while the second one indicates the optimum sensing period when the PU utilizes the medium.

[4] http://gnuradio.org/redmine/projects/gnuradio/wiki



ID	tnl_dynaq_smdp_0
Phi	1
M parameter	100
Sense cost	1.0
Penalty	-10.0

Figure 49: **SMDP Q-Learning GNU Radio block parameters dialog.**

A code snippet from the algorithm's GNU Radio implementation is presented in Snippet 1. The *work()* method is a special method of GNU Radio that is called by the scheduler whenever enough processing items are available. The available input items during a *work()* invocation are obtained through the *noutput_items* input parameter. The *in* pointer refers to the available input spectrum occupancy samples, whereas *out_0* and *out_1* are the estimated best spectrum periods when the PU is inactive and active respectively. The *d_distribution* is a uniform random generator engine that produces random numbers in the range of [1, M]. In addition the *d_dynaq_state* is a structure that holds various values necessary for the functionality of the algorithm. At every available input time-slot, the GNU Radio processing block assigns a reward or a penalty depending on the current and previous spectrum states and updates the best estimated sensing periods during absence or presence of the PU.

```
int
  dynaq_smdp_impl::work (int noutput_items,
                         gr_vector_const_void_star &input_items,
                         gr_vector_void_star &output_items)
  {
    int i;
    size_t j;
    float e;
    double reward;
    double a;
    const uint8_t *in = (const uint8_t *) input_items[0];
    float *out_0 = (float *) output_items[0];
    float *out_1 = (float *) output_items[1];

    for (i = 0; i < noutput_items; i++) {
      if (!d_action_slots) {
        init_mdp (&d_dynaq_state, in[i]);
```

```
          e = d_distribution_e (d_rnd_gen_e);

          /* NON uniform sampling */
          if(e < d_e) {
            d_a = d_distribution (d_rnd_gen);
          }
          else {
            d_a = d_M - 1;
          }
          d_action_slots = d_a - 1;
          d_action_reward = 0.0;
        }
        else{
          /* Update MDP structure according to the spectrum state */
          d_dynaq_state.b_p = d_dynaq_state.b_c;
          d_dynaq_state.b_c = in[i];

          if (d_dynaq_state.b_p == d_dynaq_state.b_c) {
            d_dynaq_state.n_a++;
          }
          else {
            d_dynaq_state.n_a = 1;
          }

          d_dynaq_state.n_c++;

          if (d_dynaq_state.b_c != d_dynaq_state.b_i) {
            d_dynaq_state.n_w++;
          }

          if (d_dynaq_state.n_w == d_phi && d_dynaq_state.n_phi == 0) {
            d_dynaq_state.n_phi = d_dynaq_state.n_c;
          }
          d_action_slots--;

          update_D(d_dynaq_state);
          a = get_a(d_dynaq_state);

          /* Perform immediately intra-option learning */
          if (d_dynaq_state.n_phi) {
            update_Q_penalty(d_dynaq_state, a, d_dynaq_state.b_c);
          }
          else {
            update_Q_reward(d_dynaq_state, a);
          }
        }

        out_0[i] = (float) max_Q_index (0);
        out_1[i] = (float) max_Q_index (1);
      }

      // Tell runtime system how many output items we produced.
      return noutput_items;
```

Snippet 1: GNU Radio implementation of spectrum sensing algorithm.

3.3.3 KPIs

The KPIs that are measured within this set of experiments are the following (see also evaluation criterion AL.PE.7 in RERUM Deliverable D5.1 [RD5.1]):

- Speed of convergence to the identified period for spectrum sensing:

  - This KPI shows how fast the algorithm converges to a specific period. Theoretically, the Q-Learning algorithms converge in infinite time. However, practically the convergence time can be very high. The goal of the experiments is to show here that with the modifications that we did to the Q-Learning algorithm, our version converges very fast.
- Comparison of the identified period with regards to the "optimum" period:
  - This KPI will show how close is the period that our algorithms finds with regards to the "optimum" period (which is found when the standard algorithm is run for a very long time). This will prove that our algorithm not only converges very fast, but also the period that it finds is very close to the optimum one.
- Energy consumed for sensing using the optimum period.

3.3.4 Experimental setup

The experiment involves only one SDR device that plays the role of the RD, running the implementation of the mechanism for lightweight spectrum sensing as presented above. Then, this RD senses the spectrum at specific spectrum bands, in order to identify the optimum period for sensing. However, since this experiment will be executed at indoor environments and the RD will select to sense spectrum fragments at the TV-bands (below 900MHz) it will be difficult to sense accurately real transmissions and to know their transmission model in order to evaluate if the spectrum occupancy model extracted by the RD is accurate. Thus, for the sake of the experiment and to be able to make a proper evaluation of the results, another SDR device will be used to play the role of a licensed user that transmits according to pre-defined models at a specific spectrum band.

To validate the algorithm, we used the flow-graph of the implementation of the spectrum sensing module using GNU Radio as depicted in Figure 50. This flow-graph receives time domain samples from the SDR device and performs frequency transformation using a Fast Fourier Transform (FFT) block. Then it computes the energy of the result and groups the samples into the corresponding time-slots. If the mean energy of each time-slot is above a user-defined threshold the flow-graph propagates a 1 to the Q-Learning block, indicating that this time-slot was an ON period and the PU was transmitting (spectrum band occupied). Otherwise, the Q-Learning block receives a 0, which means that the spectrum band is free. When the flow-graph stops, it provides the optimal period duration estimated for the ON and OFF spectrum states respectively.

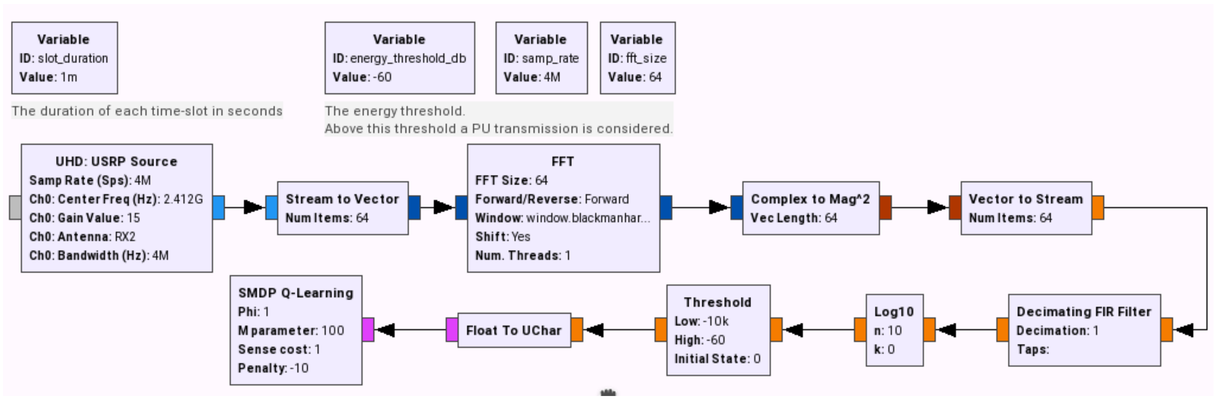


Figure 50: **SMDP Q-Learning algorithm GNU Radio flow-graph.**

3.3.5 Experimental scenarios

The scenario run in the experiment is described below.

- The RD is an SDR-based device (laptop with an SDR card) capable of sensing a wide spectrum band.
- The licensed user is another SDR device that has a specific transmission model.

- The RD will start sensing the spectrum band that is only used by the licensed user according to the spectrum sensing mechanism.
- After an amount of time the RD will have converged to the optimum sensing period and will have extracted a model of the transmission of the licensed user (depending on the characteristics of the model).
- The speed of convergence to the optimum period in terms of time (number of timeslots) will be assessed.
- The energy consumed until the convergence will be measured together with the energy consumed for each period of sensing.

3.3.6 Energy measurements

One of the key parameters of the Q-learning algorithm is the energy cost induced by sensing the spectrum for the duration of a time-slot. This cost is highly correlated with the hardware device that is used, the bandwidth of interest and the RF gain.

The SDR device chosen to be used was the Nuand bladeRF[5] that has the advantage of using a USB 3.0 port which can be bypassed for the power supply with an external adapter. For this purpose, it provides two special jumpers on board that reroute the power source input. Unfortunately, a small amount of power is still drawn from the USB port, but only to provide power to the USB interface chip. The FPGA and RF segments of the board may be powered by an external dc adapter. In order to provide a stable external power source, we used the AXIOMET AX-3003D-3 bench DC power supply as a constant voltage source [Voltage stabilization ≤0.01% ± 1mV]. For the DC voltage and current measurements, we used the UNI-T UT61E multimeter with 22000 counts resolution and ±0,1% voltage & ±1,2% current accuracy respectively.

We first fixed the voltage to 4.99V and then we used the multimeter in a series with the power source electrical circuit. Therefore, we were able to measure with high accuracy the current flowing towards the device.

Table 33: Current measurements of BladeRF at 4 MHz bandwidth.

	RF Gain (dB)				
Operation Mode	**0**	**10**	**15**	**20**	**25**
RX	498 mA	508 mA	508 mA	508 mA	508 mA
TX/RX	540 mA	561 mA	583 mA	625 mA	692 mA

Table 34: Energy of BladeRF at 4 MHz bandwidth.

	RF Gain (dB)				
Operation Mode	**0**	**10**	**15**	**20**	**25**
RX	2.485 W	2.534 W	2.534 W	2.534 W	2.534 W
TX/RX	2.694 W	2.799 W	2.909 W	3.118 W	3.453 W

During our power measurement, the power source voltage was 4.99 V and the bandwidth 4 MHz. This is the bandwidth utilized by the IEEE 802.15.4 telecommunication standard. The resulting current values for different setups of RX RF gain can be retrieved from Table 33. This table also provides the energy measurements when the device is concurrently transmitting a signal at 4 MHz bandwidth. In addition, Table 34 provides the energy consumption in Watts.

[5] http://nuand.com/

3.3.7 Evaluation results

3.3.7.1 Optimum period vs penalty

As described in deliverable D4.1 [RD4.1], the key parameters of the lightweight spectrum sensing algorithm are the sensing cost e_c and the penalty E_c induced by a wrong estimation of the PU spectrum state. For the e_c, we used the normalized energy consumption for the device to perform spectrum sense during a time-slot, with the RF gain of the SDR device configured at 15 dB. The penalty E_c corresponds to the energy that will be spent for the device to find a new available channel among the 16 different plus the energy that is required for switching to the new target frequency. Because the frequency switching energy overhead is very difficult to be computed, we consider only the energy cost for the search of a new idle channel.

Assuming that in order to decide if a channel is idle or not a sensing period of two time-slots is needed the penalty is:

$$E_c = \#\text{num}_{channels} * 2 * e_c = 32 * e_c$$

The E_c value greatly affects the learning algorithm. Small penalty values will allow the algorithm to converge to larger sampling periods. On the other hand, large penalty values will force to algorithm to be more conservative and try not to erroneously predict the spectrum state, hence it will converge to smaller sampling periods. This behavior is depicted in Figure 51.

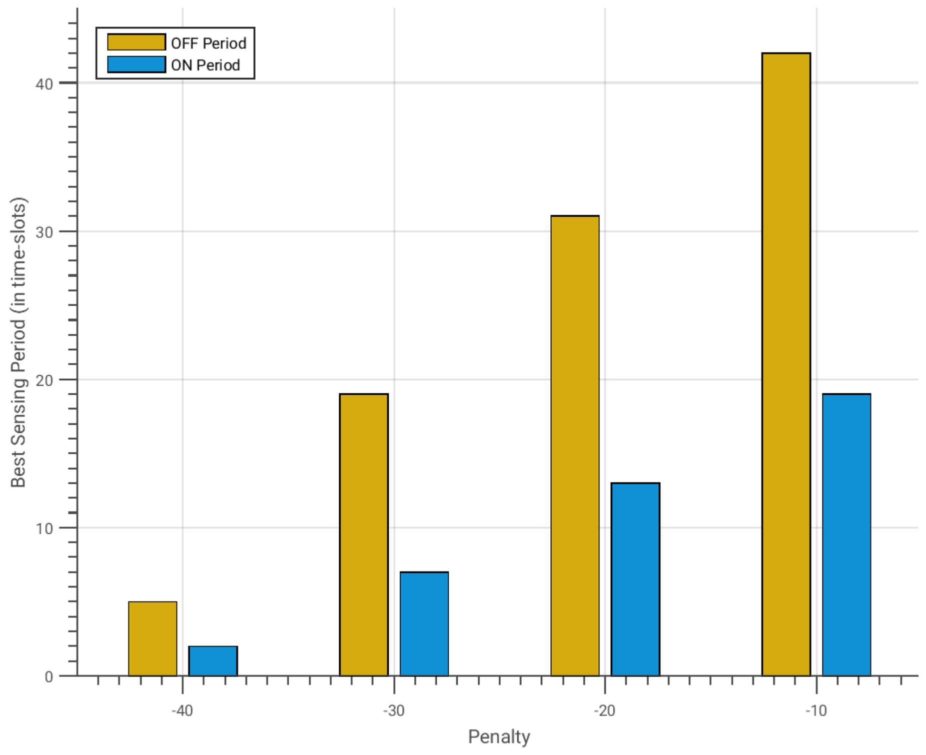


Figure 51: Sampling periods for different E_c penalty values.

3.3.7.2 Optimum period convergence speed

To validate the algorithm we used the flow-graph of depicted in Figure 50. This flow-graph receives time domain samples from the SDR device and performs frequency transformation using an FFT block. Then it computes the energy of the result and groups the samples into the corresponding time-slots using an FIR filter for the purpose of a moving sum. The resulting samples are then passed through the Threshold block. This block produces a “1” at its output port whenever the mean energy of the corresponding input time-slot is above a user-defined threshold indicating that during this time-slot the PU was active. Otherwise the output is “0” which is a direct indication that the PU was idle and the medium was free. Then the resulting samples of the Threshold block are properly transformed into the desired 8-bit integer form and are driven towards the Q-Learning block.

As an initial step, we conducted an experiment to retrieve the optimal spectrum sensing periods for both active and inactive PU state. For this reason we used a block that generates artificial spectrum occupancy data. The active PU periods are generated using a random geometric distribution with a mean of value 10. On the other hand, the inactive periods of the Primary User (PU) follow also a random geometric distribution with a mean value of 70. Based on the fact that the algorithm will converge to the optimum results with probability “1”, we let the experiment to run for several hours in order to get the best possible results.

Table 35: Q-Learning results for e_c=1 and E_c=32.

PU State	Optimum sampling period (in time-slots)
OFF	11
ON	45

Assuming a normalized sensing cost e_c equals to “1”, the penalty E_c based on the penalty estimation equation previously discussed is equals to 32. The resulting optimum sampling periods after a long running experiment are presented in Table 35.

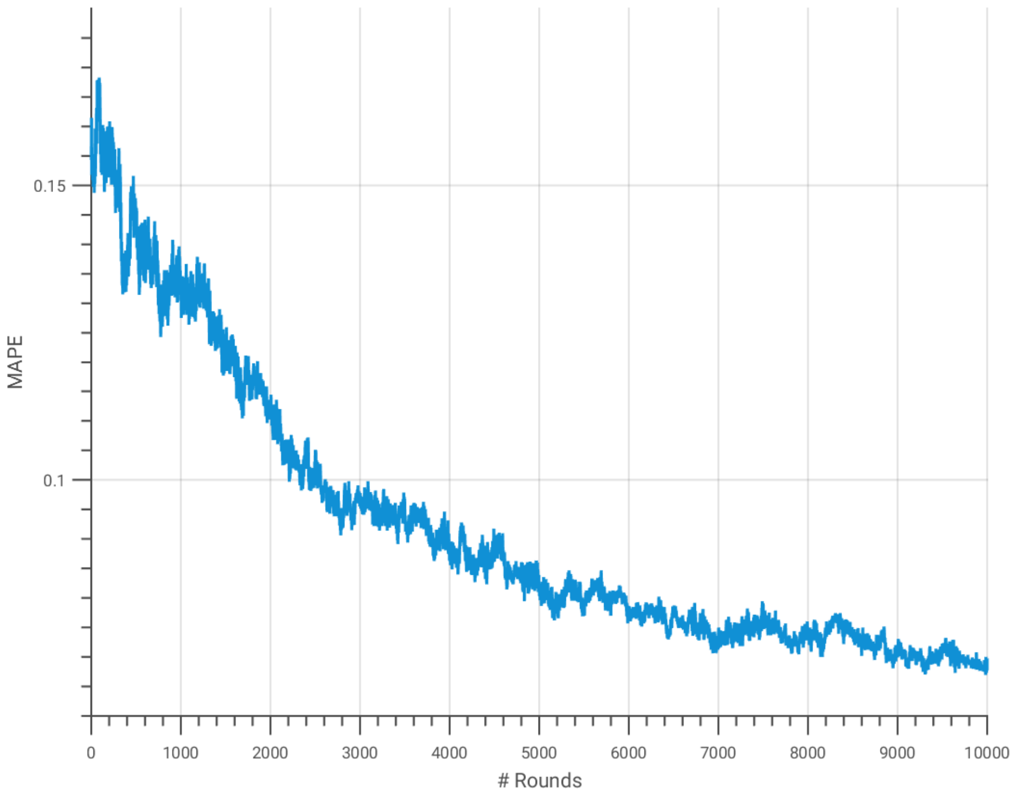


Figure 52: Q-Learning convergence.

However, these results alone have limited importance. The actual performance of the algorithm is based on the speed of convergence towards the optimum spectrum sensing periods. To evaluate the convergence rate we use the Mean Absolute Percentage Error (MAPE) that is calculated at the end of each sensing period of the algorithm. After a warm-up period of 1000 rounds the MAPE is depicted in Figure 52. This plot reveals that the algorithm has a relative small error below 15% even after a few hundreds rounds after the beginning of the experiment. After 3000 rounds the error is below 10%. This indicates that the algorithm has a very satisfactory convergence rate towards the optimum solution and thus it can run for a small period of time drastically reducing the power requirements. As a result significant energy savings are gained not only from the learned periodic spectrum sensing periods, but also from the short amount of time that the algorithm should be executed.

3.3.7.3 Energy consumed

Computing the energy that the CR-inspired RD consumes when it uses the proposed mechanism for spectrum sensing is a rather simple task. Assuming that we have an antenna with 0 dB gain, we can see from Table 34 that it consumes ~2.5Watts for the RX (which is also consumed for spectrum sensing). Considering the nature of the proposed algorithm, the RX component of the RD can "sleep" for many timeslots. The number of timeslots that it sleeps depends on the output of the algorithm, which is the sensing period. Thus, it can be seen that for the specific experiments we run, the RD should sense the spectrum either every 11 time-slots (if the spectrum was free) or every 45 time-slots (if the spectrum was occupied). This means that if the spectrum was free, the RX component should sleep for 10 time-slots and then wake up in order to sense the spectrum again. Furthermore, if the spectrum was occupied, the RX component will sleep for 44 time-slots and then wake up to sense the spectrum. It can be realised easily that when the RX component "sleeps" it does not consume the energy mentioned in Table 34. Thus, with the timeslot set as "1ms", we can see that the RD saves 2.5W*10timeslots*1ms/timeslot=25W*ms=6.95*10E-6 Wh each time that the spectrum is found unoccupied and 3.05*10E-5 Wh each time that the spectrum is found occupied.

3.4 CR-based gateway

This experiment tries to evaluate in a real deployment the performance of the Cognitive Radio based IoT gateway, which was described in Deliverable D4.1 [RD4.1]. The implementation of the CR gateway can be realised either with one Software Defined Radio (SDR) for each one of the radio technologies we want to utilize or by using only one SDR card that will emulate all technologies. The first case is the easiest one and the simplest one, but the second case is more cost-efficient. In our experiments, we have used two SDR cards, one for each of the wireless technologies to show the worst case in terms of energy consumption. The CR-based IoT gateway can be assumed as very important in IoT, due to the fact that it is controlled completely by software and can be utilized to serve many technologies simultaneously, only by installing the required software. The antennas that are used span from few MHz up to 6GHz so they can be used for many transmission technologies. Thus, in future IoT scenarios with RDs that apply Dynamic Spectrum Access mechanisms, this type of a gateway will be mandatory to ensure an efficient interconnection of RDs with diverse and heterogeneous types of traffic and different technologies. More details regarding the actual implementation of the CR-based gateway can be found in D4.1 [RD4.1].

3.4.1 KPIs

The KPIs that will be measured within this experiment are the following:

- CPU and RAM utilization for each of the networking technologies and under various types of load: the goal here is to see how the load of the networks affect the resources of the CR-based gateway. This can be also seen as a metric of the scalability of the device, because a high increase of either CPU or RAM with an increase in the traffic load will show that the performance of the gateway can be very limited.
- Power consumption: this KPI will show how much is the power consumption of the device and how this power consumption changes depending on the network traffic in both wireless technologies.

3.4.2 Experimental scenarios

The experimental setup is shown in Figure 53 below. There are two RDs, one being connected with a IEEE 802.15.4 interface (the Zolertia RE-Mote) and another one connected with a standard IEEE 802.11 interface (laptop or smartphone). These devices are connected with the SDR-based gateway, which can route traffic between these technologies, as well as with the RERUM MW through an Ethernet connection.

For the evaluation of the CD-based IoT gateway we run the following three experiments:

Experiment A:

In this experiment, only the 802.11 device was running, sending measurements and background traffic to the GW and we measured the performance and the energy consumption of the gateway.

Experiment B:

In this experiment, only the RE-Mote connected with 6LoWPAN was operating. The RE-Mote was running a firmware that enabled it to become a web-server that allowed the MW to ask for various amounts of UDP-based traffic. That way we were able to perform the experiment and test the performance of the gateway for different loads of traffic.

Experiment C:

In this experiment, both the 802.11 and the 6LoWPAN devices were operating and we tested the performance of the gateway for various types of traffic load when it was serving both technologies.

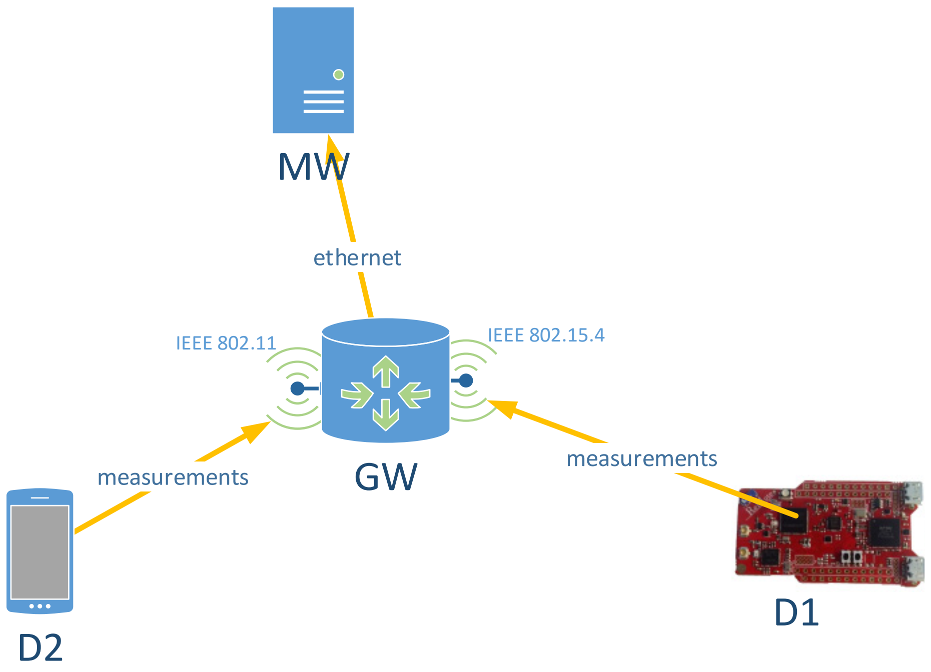


Figure 53: **Topology of the SDR-based gateway experiment.**

In all the experiments, we used three loads of traffic (i) low, (ii) medium and (iii) high traffic. This traffic was generated through the cURL software accessing the web server at the RD for the 802.15.4: low traffic (20%), medium traffic (50%) and high traffic (80%). In case of 802.11 the traffic was generated through video streaming (low traffic, of the packet or (in case of 802.11) via video streaming of different quality videos (240p, 480p, 1080p).

3.4.3 Evaluation results

The results of the performance evaluation are given in Table 36 for each one of the three experiments. It has to be noted here that the values in the table correspond only to the real operation of the CR gateway software and not of any other software that runs on the device. Thus, these results show only the overhead of running the SDR gateway under various conditions.

Table 36: CR-based gateway performance results.

Measurement	Experiment 1 (802.11)	Experiment 2 (802.15.4)	Experiment 3 (802.15.4 + 802.11)
Memory consumption (low)	199916 KB	154044 KB	353960 KB
Memory consumption (medium)	199916 KB	154044 KB	353960 KB
Memory consumption (high)	199916 KB	154044 KB	353960 KB
CPU percentage (low)	7.00%	6.00%	13.00%
CPU percentage (medium)	7.00%	6.00%	13.00%
CPU percentage (high)	9.00%	6.00%	15.00%
Consumption (low)	51W	45W	56W
Consumption (medium)	53W	45W	56W
Consumption (high)	54W	45W	59W

The first thing that one can notice is that the memory consumption of the gateway remains the same regardless the amount of traffic that the RDs produce. It can also be seen that the 802.11 implementation uses 45MB of RAM more than the 802.15.4 implementation and that when both interfaces run simultaneously, the amount of RAM that the gateway uses is equal to the sum of the memory consumption that the interfaces use when they run as standalone components. This is quite reasonable, because each interface technology runs as a separate software component, so it uses a specific amount of memory when it runs, regardless if it runs together with other software components or not. It has to be noted here that the required amount of memory (i.e. 350MB) is only a small percentage of the available memory of the PC that plays the role of the gateway (16GB of RAM) so of course the two software components do not compete for the free memory.

In terms of CPU consumption, it can be seen that both for 802.11 and 802.15.4 the required CPU is quite low (7% and 6% respectively) in low and medium traffic. What is interesting is that the CPU consumption for 802.15.4 remains the same even if the traffic is high, while for 802.11 there is a small increase in the CPU usage (mainly due to the large number of packets that have to be decoded very rapidly). When both interfaces run simultaneously, similarly as in the memory consumption, the total CPU consumption is equal to the sum of the respective CPU consumptions of the standalone interfaces.

The last test was for the energy consumption of the CR-based IoT gateway. In the table, the total consumption of the PC that was playing the role of the gateway is displayed. It has to be noted that the energy consumption of the PC when it is idle (not running anything) is 20W. We can see from Table 36 that the energy consumption of the 802.15.4 interface remains static regardless of the load, while for the 802.11 the consumption increases slightly. The interesting note is, however, that when both interfaces run simultaneously the total energy consumption is only slightly higher than that of the 802.11 interface.

As a general remark, the results show that the current implementation of the CR-based IoT gateway scales very well under load and it is very lightweight with very low memory, CPU and energy consumption. However, due to the fact that we didn’t have many RE-Mote devices, we were not able to test the scalability of the Gateway with regards to serving a large number of clients. This remains as a future action.

3.5 6LoWPAN Multicast

In scenarios involving point-to-multipoint network traffic, transmitting to each destination individually with unicast leads to (i) poor utilization of network bandwidth, (ii) excessive energy consumption caused by the high number of packets and (iii) suffers from low scalability as the number of destinations increases.

3.5.1 Relevance to RERUM's Use-Cases

For UC-O2 - Environmental monitoring in particular, it is expected that networks will be formed by a potentially very high number of RDs and therefore scalability is a requirement.

In cases when RDs are powered by batteries, it is impractical or outright untenable to replace batteries very frequently due to high management cost and possibly hard-to-reach installation locations. Thus, long battery life is important.

For devices powered from mains, low energy consumption is also important in order to reduce financial cost, but also in order to comply with national and international regulations where applicable. For those reasons, RERUM has designed and implemented the Bi-Directional Multicast Forwarding Algorithm (BMFA) for 6LoWPANs.

3.5.2 Purpose of the experiment

The purpose of these experiments will be to demonstrate how M/W functions can leverage IPv6 multicast in order to improve network performance and decrease energy consumption, ultimately increasing the lifetime of a smart object deployment.

The results from this experiment are part of the evaluation for the following evaluation criterion that has been defined in deliverable D5.1 [RD5.1]:

- AL.PE.5 "6LoWPAN Multicast"

3.5.3 KPIs

This section presents a thorough performance evaluation of the BMFA and ROLL_TM multicast forwarding algorithms, which were developed during Task 4.2 and introduced in RERUM deliverable D4.1 [RD4.1].

More specifically, the KPIs that are measured within this set of experiments are the following:

- **Suitability for embedded devices**: by measuring code size and RAM footprint.
- **Reliability**: by measuring packet loss / packet delivery ratio. Target: This metric is highly-sensitive to traffic rate, network topology, node configuration etc.

3.5.4 Experimental setup

Our thorough empirical campaign was conducted with five RE-Mote devices, with each one of them running either BMFA or TM as multicast forwarding algorithm. We configured one node as a root, two as 1-hop sinks, while another two devices as 2-hops sinks (Figure 54). We implement downstream traffic, and thus, the root node transmits both in CBR (i.e., with time interval of 0.5 or 1 sec) and VBR with random time interval between 0.5 and 2 seconds. Table 37 summarises the experimental setup, as discussed throughout this section.

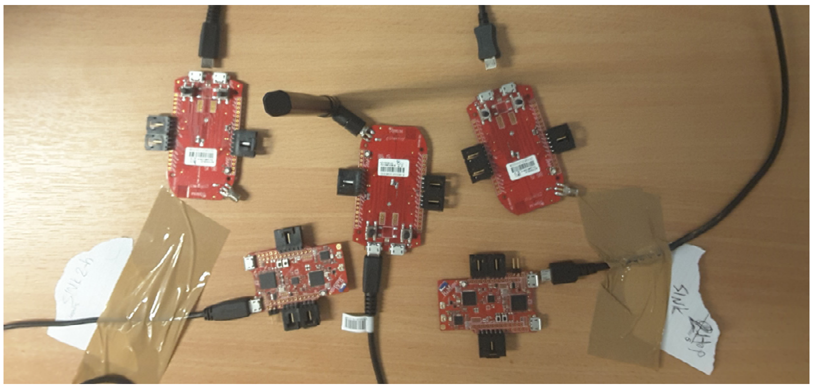


Figure 54: Studied topology.

Table 37: Experimental setup.

Topology	
Number of nodes	5 RE-Mote devices
Number of sources	1 root node
Studied protocols	BMFA / TM
Experiment parameters	
Traffic pattern	CBR: 0.5, 1 sec VBR: [0.5 – 2] sec
Number of events	100
Payload size	5 / 20 / 40 bytes
Routing	RPL
MAC	CSMA
Duty Cycling	ContikiMAC
PHY	IEEE 802.15.4
Hardware parameters	
Antenna model	Omnidirectional CC2538
Radio Propagation	2.4 *GHz*

3.5.5 Performance evaluation

3.5.5.1 Code size and RAM footprint

Code footprint and RAM requirements are measured at compile-time. This is achieved by building a firmware image and by subsequently running the toolchain's -size command on object files. In this case we ran the following command: $ arm-none-eabi-size obj_remote/(roll-tm.o / smrf.o / bmfa.o)

Table 38: Code size and RAM footprint in bytes for each algorithm.

Algorithm (filename)	text	data	bss	dec	hex
obj_remote/roll-tm.o	3224	12	8114	11350	2c56
obj_remote/smrf.o	335	0	1335	1670	686
obj_remote/bmfa.o	1578	0	1335	2913	b61

Each roll-tm.o, smrf.o and bmfa.o modules are core of the implementation of the ROLL-TM, SMRF and BMFA algorithms respectively. The results (Table 38) provide the following information about each code module:

- Code footprint, including program memory and constant (const) expressions (text)
- Variables initialized at compile time (data)
- Space reserved for variables which are not initialized at compile time (bss)

As can be observed from Table 38, our proposed BMFA algorithm achieves the targets defined in D5.1 for the RE-Mote platforms. More precisely, both code size (i.e., text) and RAM footprint (i.e., data + bss) present results below 3 KB for Multicast functionality when compared to ROLL-TM (> 3 KB and >8 KB respectively).

3.5.5.2 Reliability

For each multicast group, we calculate Packet Delivery Ratio (*PDR*) with the following equation:

$$PDR = \frac{\sum_{i=1}^{M} R_i}{N \times M}$$

where *N* is the number of unique multicast datagrams sent by traffic sources to the group's IPv6 address, *M* is the number of multicast group members and R_i is the number of unique multicast datagrams sent to this group and received correctly by node *i*. This equation is valid under the assumption that multicast group membership remains unchanged throughout the multicast flow's entire lifecycle, which holds true in the experiments presented here.

PDR is an application layer metric and can take values in [0, 1]. Packet loss is calculated as 1 - PDR, thus packet loss 0% is the equivalent of 100% *PDR*, which means that all multicast datagrams were received correctly by all multicast group members.

The two graphs in Figure 55 and Figure 56 respectively, illustrate the results for three different datagram transmission rates in multi-hop topology over ContikiMAC (i.e., 125ms sampling frequency).

We should take into consideration that in tree topologies, TM has a comparative advantage since there is path redundancy: a node may receive a multicast datagram from any of it's potentially multiple neighbours. We thus anticipated its packet losses to be considerably lower. Multicast traffic transmission rates once again appeared to have minimal impact on the results and are thus averaged out. Thus, TM presents comparative results in PDR regarding the 2-hops (see Figure 55) nodes.

Furthermore, investigating the behaviour of BMFA and TM schemes for different CBR and VBR values, we observe that the shorter is the time interval (i.e., 0.5 sec) the less efficient is the TM algorithm. Indeed the packet delivery ratio may vary between 70% - 50% for 1-hop and 2-hops respectively. On the other hand, BMFA scheme presents more stable results (independently the packet size) for 1-hop nodes, while for 2-hops nodes its performance is similar to TM algorithm.

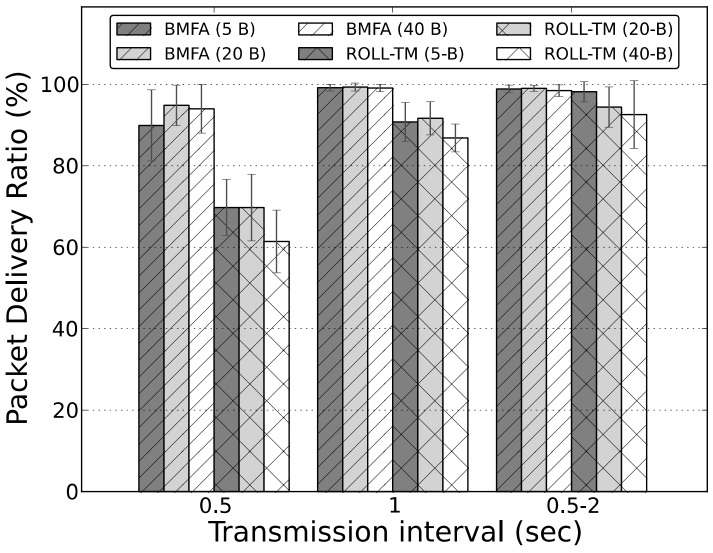


Figure 55: Reliability for 1-hop nodes.

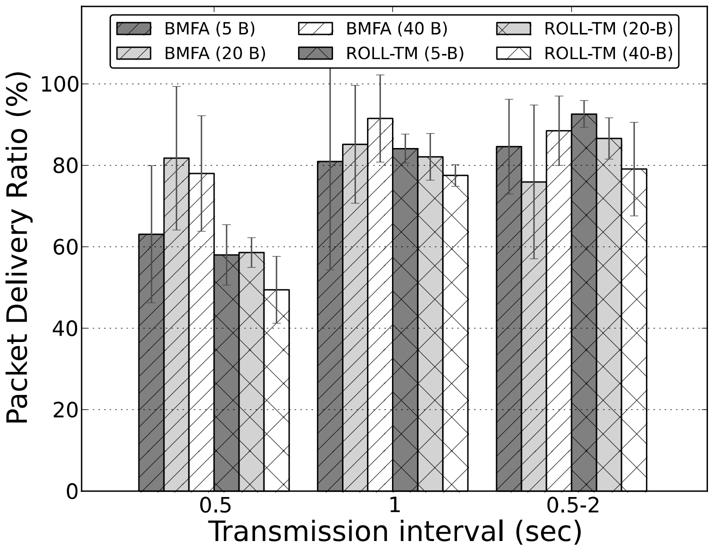


Figure 56: Reliability for 2-hop nodes.

4 Lab trials of the traffic monitoring Use-Case

This section discusses lab trial testing of the traffic monitoring Use-Case. More specifically, in 4.1 the results of the experimental evaluation of the battery drain of the participatory sensing android app are presented, while in 4.2 the evaluation of the participatory sensing app in terms of CPU usage is demonstrated. Subsection 4.3 evaluates the scalability of the server side implementation of the smart transportation use case. Furthermore, in 4.4 demonstrates the experimental evaluation how the stability of the android app is affected by the sampling period. Finally, Subsection 4.5 explores the usage of the SNR measurement precision for the RERUM android application.

4.1 Android app power drain

4.1.1 Purpose of the experiment

These experiments aim to evaluate the battery drain of the RERUM traffic sensing app. Battery drain is a key component of participatory sensing, since users will be reluctant to install apps that are causing their phone battery to drain much faster than it would without the app.

4.1.2 KPIs

The performance metric is defined as time to depletion of battery.

4.1.3 Experimental setup/scenarios

Experiments are performed when the app is actively used and the GPS is continuously active. The phone is continuously kept moving using a pendulum and a fan and the movement is large enough to trigger the accelerometer movement detection, see [RD5.2] for a more detailed description of the motion detection.

4.1.4 Evaluation results

Table 39 shows the results for a range of devices with different characteristics. The results heavily depend on battery capacity, which is also indicated in the table. Note that the performance may vary from device to device of same the model, with a window of 2 hours depending on location of the device, availability of GPS fix, and number of state changes from active to sleep modes.

Table 39: Time to depletion of battery for different phone models when the app is actively used with GPS continuously active and no other battery draining apps running.

Device	Manufacturer	Android Version	Battery	Battery Runtime
Nexus -5	LG	5.0.1	2300 mAh	36 hrs 10 mins
Nexus-4	LG	5.0.1	2100 mAh	33 hrs
Moto-E	Motorola	4.4.2	1980 mAh	31 hrs
Moto-G	Motorola	5.0.1	2070 mAh	34 hrs 36 mins
Galaxy Young 2	Samsung	4.4.2	1300 mAh	21 hrs 10 mins
Galaxy S5	Samsung	4.4.2	2800 mAh	44 hrs 23 mins
Galaxy S3	Samsung	4.3	2100 mAh	32 hrs
Xperia Z2	Sony	5.0.1	3200 mAh	51 hrs 22 mins
Xcover 2	Samsung	4.1.1	1700 mAh	23 hrs 20 mins

Galaxy Y	Samsung	2.3	1200 mAh	Not supported
One X	HTC	4.1.1	1800 mAh	27 hrs
Desire 300	HTC	4.1.2	1650 mAh	23 hrs
Galaxy S Duos II	Samsung	4.4.2	1500 mAh	25 hrs 50 mins
Xperia E1 D2005	Sony	4.4.2	1700 mAh	26 hrs 20 mins

The results in Table 39 indicate that the battery drain of the app is not prohibitively high even if the GPS is continuously active. Since GPS is one of the most battery-draining peripherals, most of the battery efficiency improvements, described in more detail in [RD5.2], are based on reducing unnecessary GPS usage.

4.2 Android app CPU

4.2.1 Purpose of the experiment

The aim of these experiments is to evaluate the participatory sensing app in terms of CPU usage. Extensive CPU usage can cause impaired user experience, smartphone overheating and poor battery lifetime.

4.2.2 KPIs

The KPI of this experiment is the CPU load of the RERUM device.

4.2.3 Experimental setup/scenarios

CPU usage is measured when the Android application is used in both the foreground and the background (GUI minimized by the user), since the difference in CPU load for these different states differs significantly. The foreground and background CPU load is measured using built-in functions in Android studio.

In order to minimize battery consumption, the application will be running different processes in different states of the application, e.g. depending on whether mobility is detected or not. To identify how the CPU varies during different phases of the application the CPU usage has been monitored separately for different phases. The different phases are *idle*, *data sharing*, *GPS monitoring* and *accelerometer monitoring*. All phases are monitored when the application is in the foreground. Data sharing means that data is transmitted to middleware servers, GPS monitoring means that the GPS receiver is active to provide a location fix, and accelerometer monitoring means that the accelerometer is active to detect mobility of the device.

CPU activity measurements for different phases of the application are collected using the android application “OS Monitor” available in the android market [OSMON].

4.2.4 Evaluation results

For benchmarking, we have compared with average CPU usage for different application categories running the application in foreground and background, respectively. The data is provided by M2 App Insight [APPINS1] and shown in Figure 57. Note that the Foreground CPU usage is very high in comparison to background CPU usage as foreground activities activate GUI processes, which are usually high CPU consuming processes.

The average CPU usage by the RERUM traffic sensing application is shown in Table 40. It can be noted that the foreground CPU usage of the application is quite low in comparison to almost all application categories in Figure 57. The background CPU usage is similar to application categories with the highest

CPU usage, which is expected for this type of application where mobility is monitored in the background. However, an important result is that these levels of CPU should not cause any issues related to impaired user experience or battery drain.

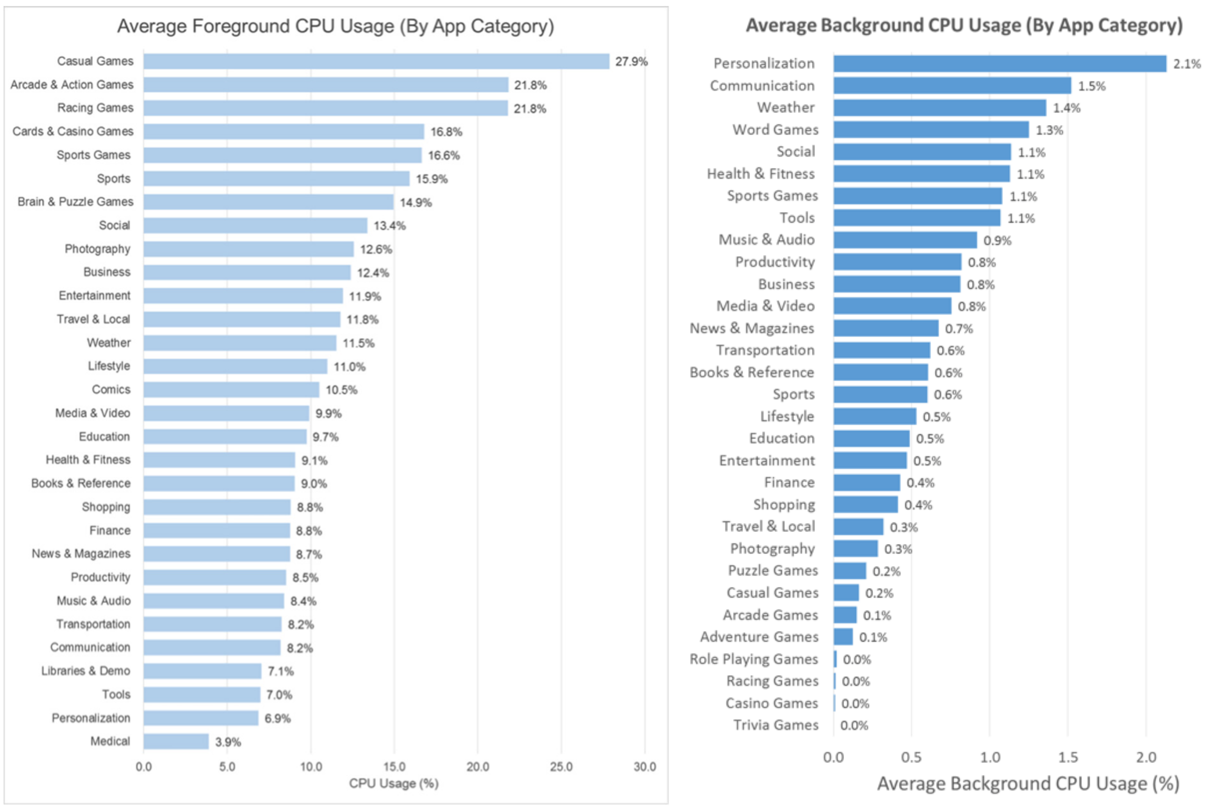


Figure 57: Average foreground CPU usage by application category (left) [APPINS1] and average background CPU usage by application category [APPINS2].

Table 40: Average CPU usage for RERUM Traffic sensing app on different devices

Device	CPU	Foreground CPU usage	Background CPU usage
LG Nexus 5	Quad core, 2260 MHz	7.64%	1.50%
HTC Desire 300	Dual-core 1 GHz	8.68%	3.03%
Sony Xperia M4 Aqua	Quad-core 1.5 + 1.0 GHz	3.86%	0.44%
Motorola Moto G	Quad-core 1.2 GHz	4.80%	0.78%
Samsung Galaxy S Duos II	Dual-core 1.2 GHz	7.62%	1.35%
Average		6.52%	1.42%

Example data of the variations of CPU load over time is shown in Figure 58, where a Nexus 5 phone is monitored when you switch the application between foreground and background use of the app.

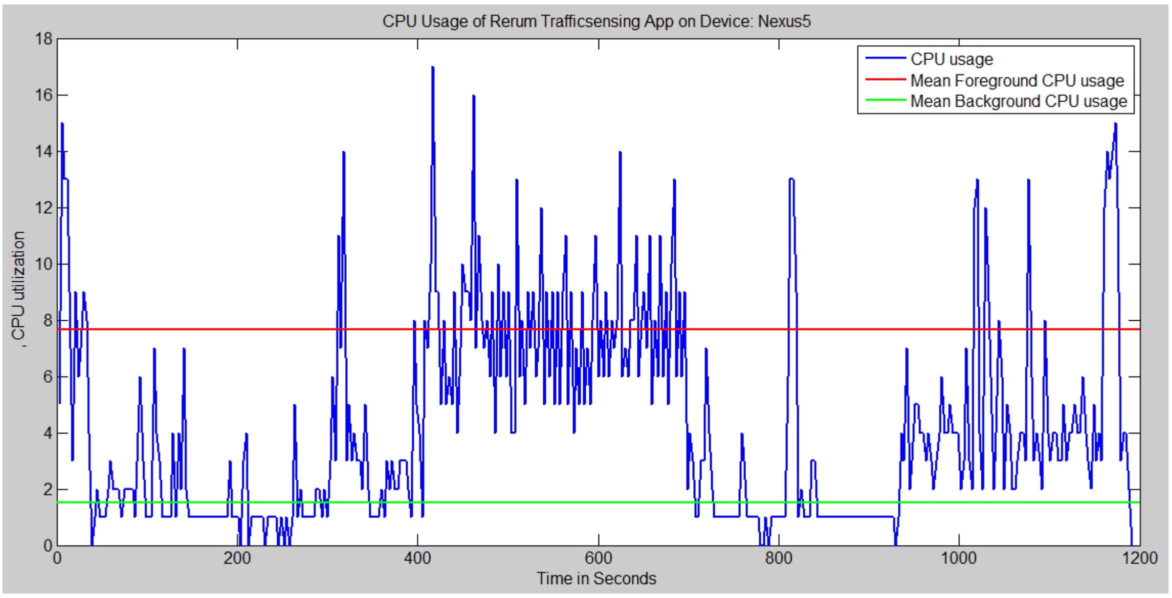


Figure 58: CPU usage for the participatory sensing app on LG Nexus 5 comparing the usage as a background application, and as a foreground application.

Table 41 shows the average CPU utilization for the processes related to different phases and it can be noted that none of the different phases are causing extensive CPU load.

Table 41: Average CPU usage for different phases related to different sub processes of the app

Device	CPU	Average CPU Utilization for sub processes				
		Idle	Data Sharing	GPS monitoring	Accelerometer monitoring	Total
LG Nexus 5	Quad core, 2260 MHz	2%	1%	0.5%	2%	7.5%
Sony Xperia M4 Aqua	Quad-core 1.5 + 1.0 GHz	1%	0.5%	0.5%	1%	3%
HTC Desire 300	Dual-core 1 GHz	6%	2%	3%	3%	14%
Samsung Galaxy S Duos II	Dual-core 1.2 GHz	5%	2%	2%	2%	11%
Moto G	Quad-core 1.2 GHz	3%	1%	1%	1%	6%

4.3 Server side scalability

4.3.1 Purpose of the experiment

The aim of these experiments is to evaluate the scalability of the server-side implementation of the smart transportation use case. The experiments evaluate both the middleware interface and the traffic application server, but not the middleware itself.

4.3.2 KPIs

The KPIs for this experiment is the CPU load and memory usage of the application server.

4.3.3 Experimental setup/scenarios

The interface between the traffic application server and the middleware is implemented using a standard HTTP interface, see [RD5.2] for a more detailed description, and instead of pulling live data from the middleware, the experiments are performed by pulling data from a temporary server that generates more GPS observations.

In phase one of the Heraklion trials, approximately 30 buses transmitted two GPS locations every 30 seconds. For the experiments we have evaluated three different scenarios; 50, 500 and 5000 messages per 30 second interval. One important part of the scalability of the system is related to the local geofencing that is performed in the RERUM devices, which means that only devices within the geofencing zones are actually transmitting data. In the trials so far, a rough estimate is that approximately 10% of the vehicles are transmitting data at the same time, with this ratio the most loaded scenario would incorporate data from about 50 000 devices.

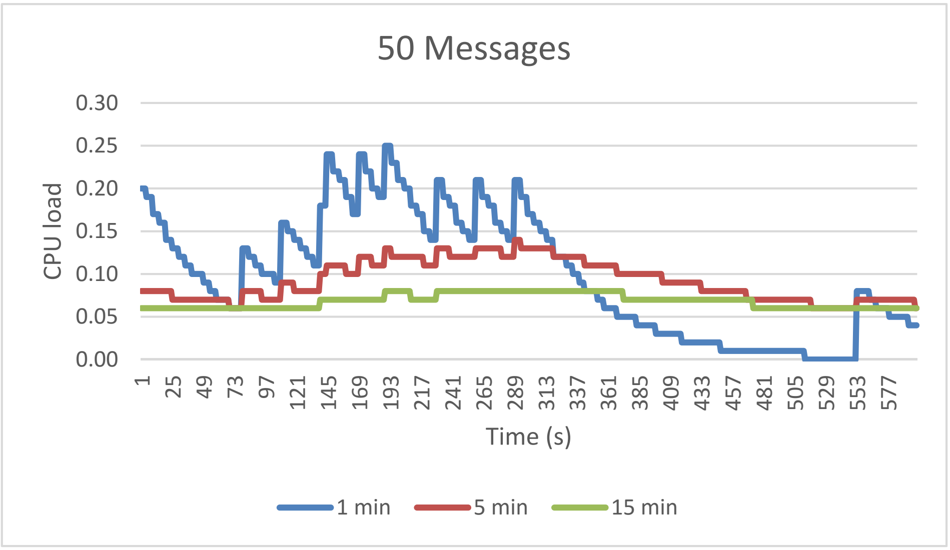
The experiments are performed on a Linux server with 4 CPUs running at 2 GHz and 10 GB RAM with a few other processes running in parallel. The CPU load and memory usage is monitored using the load average function and the memory statistics stored in /proc/meminfo for CPU and memory load, respectively. The load average function stores data about CPU usage for 1, 5 and 15 minute intervals and the values are logged every second. The data in /proc/meminfo is logged every second and the used memory is estimated as

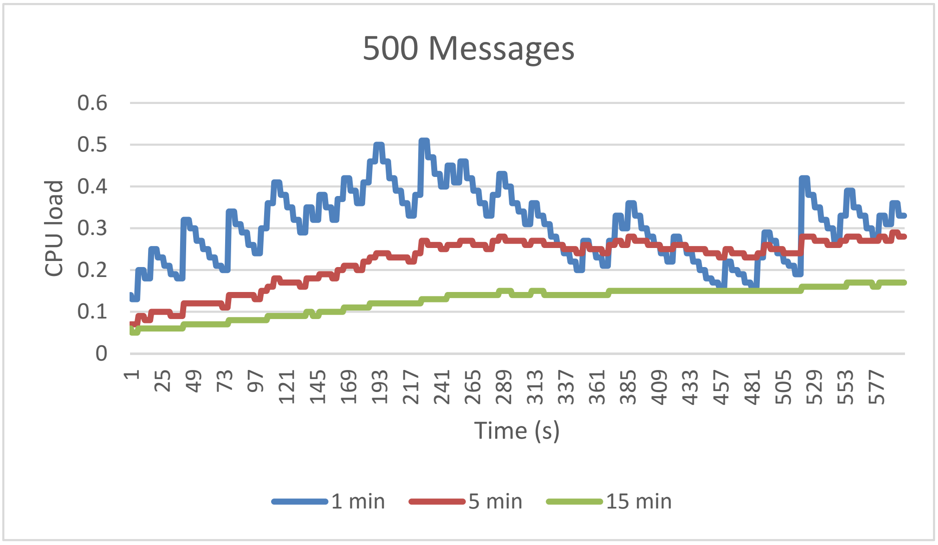
MemUsed = MemTotal - (MemFree + Buffers + Cached + SwapCached)

A more detailed description of the type of data available in /proc/meminfo is available in [MEMINF].

4.3.4 Evaluation results

CPU loads for the three different scenarios are shown in Figure 59.





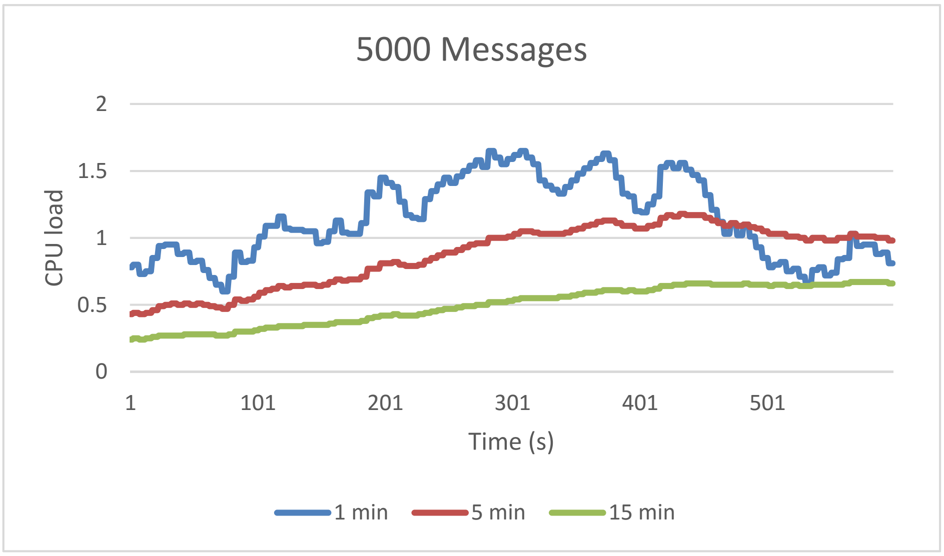


Figure 59: CPU load for the traffic application in the three different scenarios including 50, 500 and 5000 messages per 30 second interval.

Memory usage for each of the three different scenarios is shown in Figure 60.

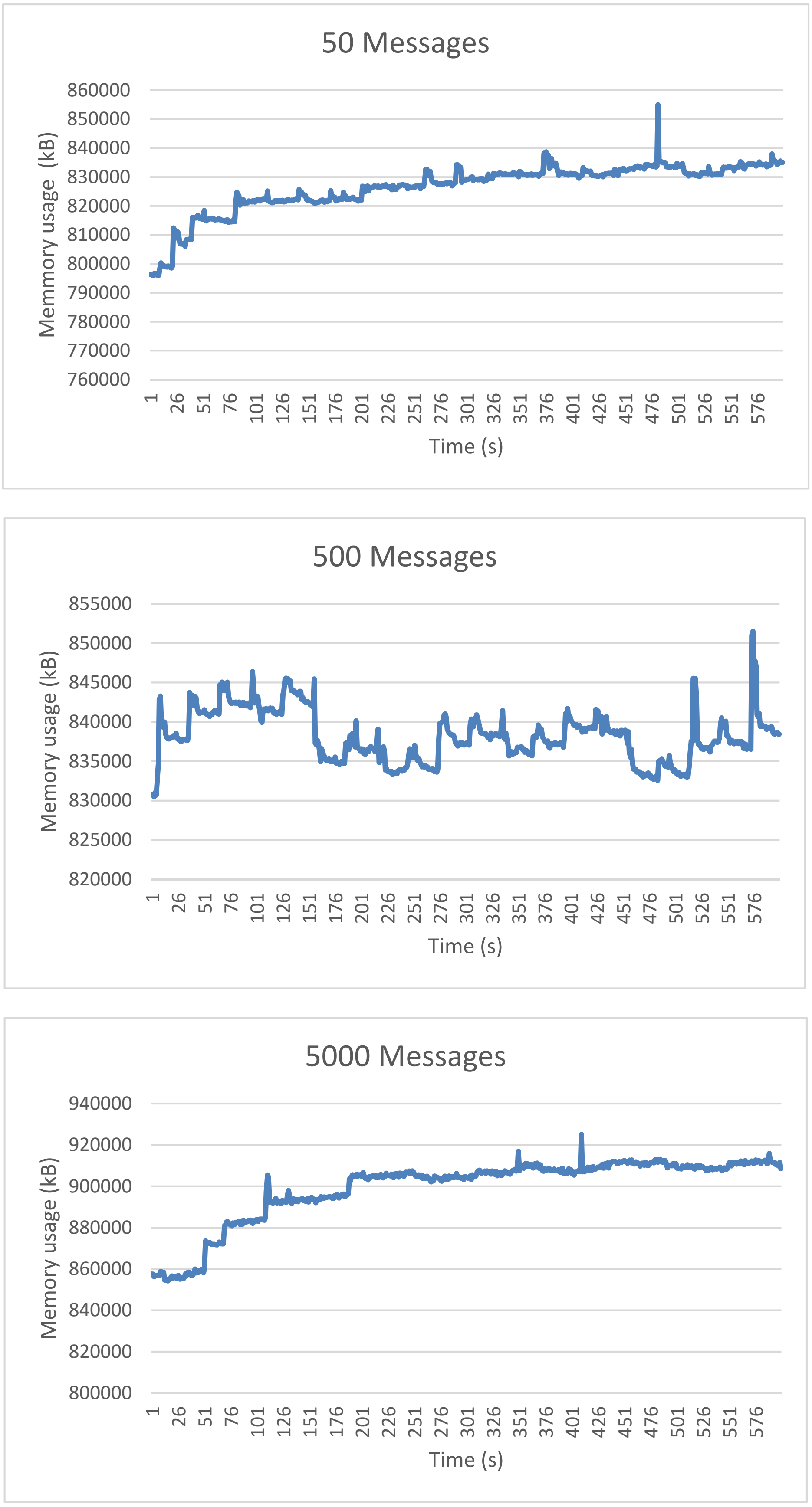


Figure 60: Memory usage for the traffic application in the three different scenarios including 50, 500 and 5000 messages per 30 second interval.

From Figure 60 it can be seen that the CPU load increases clearly when the number of messages increases and this could be a potential bottle neck of the system using the current implementation if a vast amount of users would be included. However, the process that uses most of the CPU is the database, which has some clear potential in performance improvement by tuning polling intervals of the java application and the database structure. It could also be an alternative to use several threads to enable better use of the four CPUs that are available on the machine.

In Figure 60 it can be seen that also the memory usage of the server increases when the number of messages is increased. However, the memory usage does not increase as much as the CPU load and is far from the capacity of the server even in the scenario with the largest number of messages.

4.4 Android app stability

4.4.1 Purpose of the experiment

The location data collection and transmission is the most resource consuming process of the application and depending on the type of monitoring that the RD is used for the sampling frequency could vary. The aim of these experiments is to evaluate how the stability of the application is affected by the sampling period for location data transmission.

4.4.2 KPIs

The KPI of this experiment is the crash frequency of the RERUM traffic sensing application.

4.4.3 Experimental setup/scenarios

The standard sampling period of the app is 30 seconds and we have tested the app for two hours' normal usage using a sampling period of 15, 7 and 3 seconds, respectively.

4.4.4 Evaluation results

The application did not crash in any of the tests for the different sampling intervals. When a sampling period of 3 seconds is used, the app is collecting data ten times more frequent than we do in the Smart Transportation trial. In the area of road traffic monitoring in a participatory sensing context, it is not reasonable to collect data from participants more frequent than every three seconds and it seems like in this range of sampling intervals the application is not affected by resource usage from the location data collection and transmission process in such a way that it causes the application to freeze or crash.

4.5 SNR measurement precision

4.5.1 Purpose of the experiment

The aim of these experiments is to understand the potential of using signal strength-based positioning and mobility detection for the RERUM traffic sensing application. Signal strength-based positioning and mobility detection has the potential to improve battery efficiency of participatory sensing apps considerably, and battery efficiency together with privacy are key components in the RERUM participatory sensing framework.

4.5.2 KPIs

The KPI of this experiment is the precision of received signal strengths, as observed through the Android signal strength API.

4.5.3 Experimental setup/scenarios

Data has been collected for seven different locations in the city of Norrköping (Figure 61).

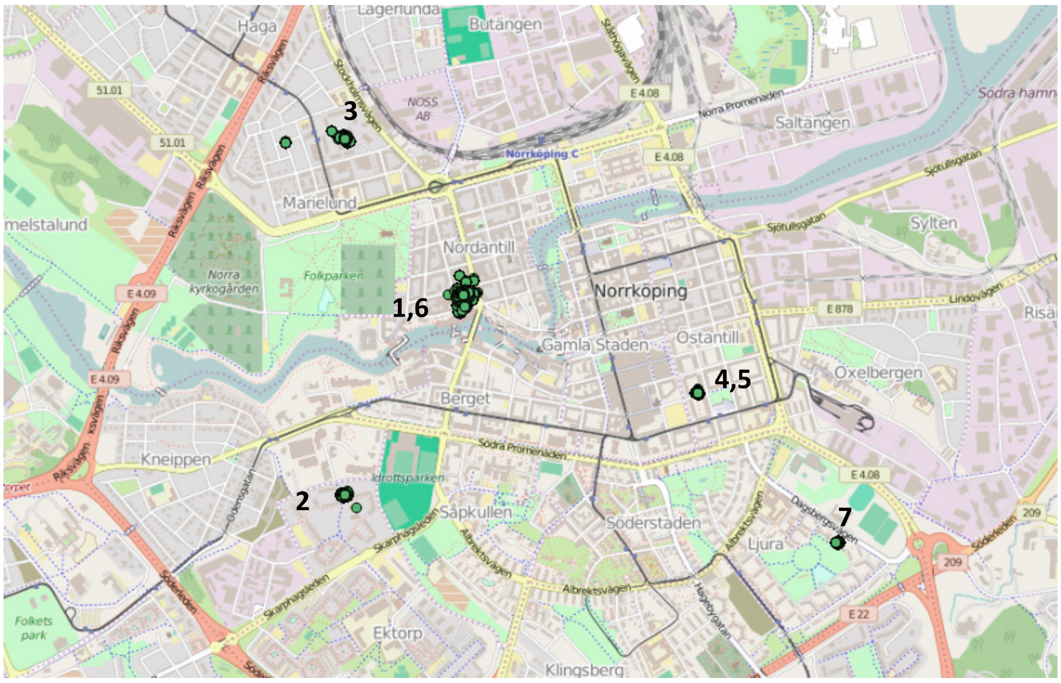
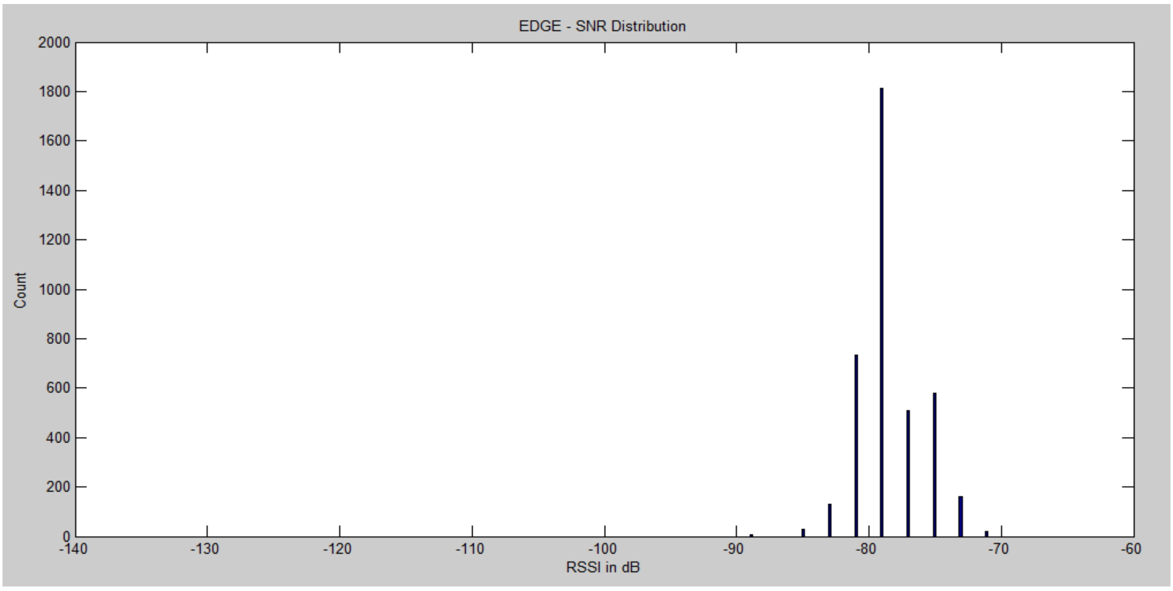


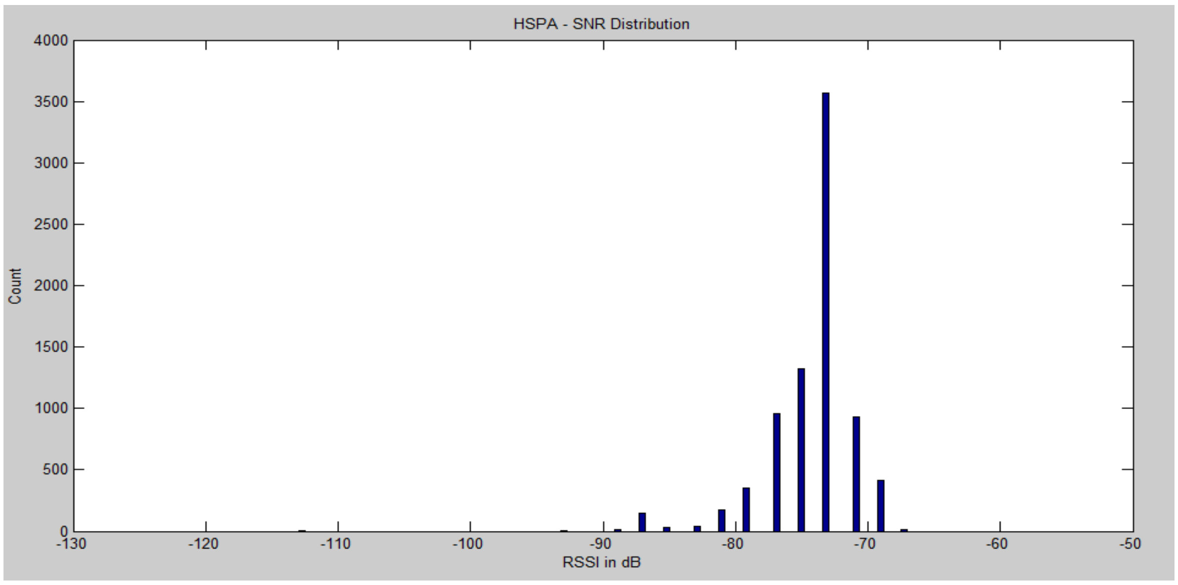
Figure 61: Locations in the city of Norrköping where the SNR data collection has been performed.

In the seven locations data has been collected using a dedicated Android app for three types of networks: EDGE, HSPA and LTE.

4.5.4 Evaluation results

Example histograms for the different networks are shown in Figure 62. Similar patterns are identified for all locations, where the LTE observations have a much wider spread. It is not clear from the experiments whether this is caused by actual variation of signal strength, e.g. caused by power control, or some error in mapping of signal strength protocol data to dB for LTE in the Android signal strength API.





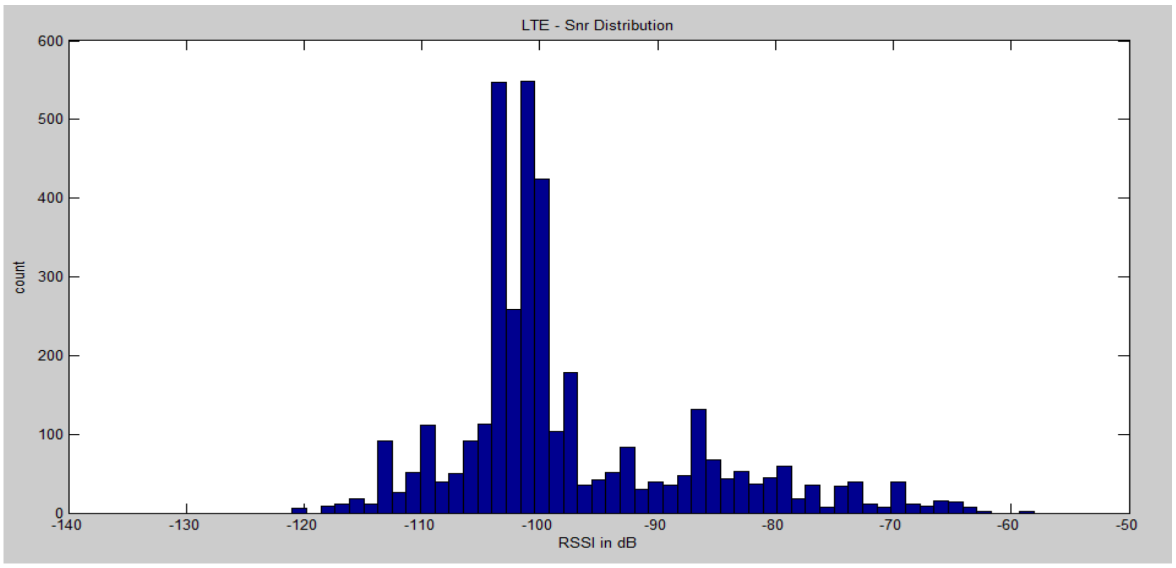


Figure 62: Signal strength variation of stationary phones for different networks.

Except for LTE, the results are encouraging for the purpose of signal strength-based mobility detection. Considering the results for EDGE and HSPA in Figure 62, it is not very likely that the signal strength will change by more than 5 dB, unless a physical movement is involved. Comparing this with some expected path loss as a function of distance we can get an idea of which level of movement that can be detected using signal strength analysis. For most macroscopic propagation models for urban areas, a 5dB change in received power would correspond to a few hundred meters' physical displacement from the base station, if the distance to the base station is not too far.

For the purpose of travel time estimation and path inference based on signal strength-based positioning it is likely that we need to rely on more observations than the currently connected base stations, i.e. observations also from neighbouring base stations, to achieve reasonable accuracy.

5 Conclusions

This deliverable presented the results of the in-lab experimental work undertaken as part of Task 5.3 "Proof-of-Concept Laboratory Experiments". Indeed, a large number of in-lab *proof-of-concept* controlled experiments have been conducted to assess both qualitatively and quantitatively the performance gains of the protocols and algorithms, as well as the individual system modules developed in work packages WP2-WP4. In total, 18 experiments were conducted by the project in a long period of continuous testing, assessment of the results, revising of the system modules and testing again the modules to ensure an optimal performance. What is presented in this document is the final version of the experimental results, with the latest version of each software module.

The experiments were split in three main categories: (i) security, (ii) networking and (iii) android application. Almost all experiments were considering (among other metrics) tests for the energy consumption and the performance of the software modules, to ensure that the modules will be able to run flawlessly on constrained devices. As shown in the detailed results in the previous sections, in almost all cases we succeeded to keep the energy consumption of the devices very low and the memory and processing usage quite low. In some cases, we were not able to run the whole experiment on the devices due to the very limited resources. For example, in the RSSI-based CS encryption, the devices are not capable of running the whole process for key generation on them, so we had to develop a different protocol to gather only the RSSI on devices and send them to a backbone server for key generation.

The execution of the experiments was done in an internal environment and mostly under controlled conditions. The goal was not to test the software modules in a real-world trial, but to have the necessary step between simulations and trials. That is to implement the software modules on real hardware and try to improve this implementation, by fixing the configuration, setting the correct parameters or changing some parts of the code in order to ensure that the experimental results in the controlled environment can be very close to simulation results. This was indeed verified every single experiment, as it was presented in detail in the previous sections. This intermediate step is crucial in order to ensure that an "optimized" version of the software modules will be used for the actual real-world trials, where re-flashing the devices with new versions of software is not always feasible (or easy to be done).

In general, the results of the many thorough experiments that were performed within the activities of Task 5.3 are very positive, since all the developed software modules work very well in the controlled laboratory environments. Furthermore, the developed hardware (the RE-Mote) was used in almost all experiments and it performed excellently. Considering these, we can be almost certain that the modules will have a comparable performance in the actual trials in both cities, which will contribute to the successful execution of the overall system trials.

References

[AG99] Abowd, Gregory D., et al. "Towards a better understanding of context and context-awareness." Handheld and ubiquitous computing. Springer Berlin Heidelberg, 1999.

[APPINS1] T. Marks, “Average Foreground CPU Usage for Google Play App Categories”, http://m2appinsight.com/average-foreground-cpu-usage-google-play-app-categories/

[APPINS2] D. Peterson, “Average Android Background CPU Usage for App Categories”, http://m2appinsight.com/average-android-background-cpu-usage-for-app-categories/

[AY07] Abbasi, Ameer Ahmed, and Mohamed Younis. "A survey on clustering algorithms for wireless sensor networks." Computer communications 30.14 (2007): 2826-2841.

[B12] Erik Bartmann. “Der A/D-wandler MCP3008 v1.3”. Technical report, adafruit, 2012. www.erik-bartmann.de .

[BCLN14] J.W. Bos, C. Costello, P. Longa, and M. Naehrig. Selecting elliptic curves for cryptography: An efficiency and security analysis. IACR Cryptology ePrint Archive, 2014:130, 2014.

[BCLN] Joppe Bos, Craig Costello, Patrick Longa, and Michael Naehrig, "Specification of Curve Selection and Supported Curve Parameters in MSR ECCLib", Microsoft Research Tech Report, available at:http://research.microsoft.com/pubs/219966/curvegen.pdf.

[BGJLSS15] Daniel J. Bernstein, Bernard van Gastel, Wesley Janssen, Tanja Lange, Pe- ter Schwabe, and Sjaak Smetsers. TweetNaCl: A crypto library in 100 tweets. In Diego Aranha and Alfred Menezes, editors, Progress in Cryptology – LATINCRYPT 2014, volume 8895 of Lecture Notes in Computer Science, pages 64–83. Springer-Verlag Berlin Heidelberg, 2015.

[BN06] Paulo Barreto and Michael Naehrig. Pairing-friendly elliptic curves of prime order. In Selected areas in cryptography, pages 319–331. Springer, 2006.

[BM93] J. Benaloh and M. de Mare. “One-way accumulators: A decentralized alternative to digital signatures”. In Proc. of Advances in Cryptology EUROCRYPT93. pages 274–285. Springer-Verlag, 1993.

[BP97] N. Baric and B. Pfitzmann. “Collision-free accumulators and fail-stop signature schemes without trees”. In Proc. of EUROCRYPT, pages 480–494, 1997.

[BPS12] C. Brzuska, H. C. Pöhls and K. Samelin. “Non-Interactive Public Accountability for Sanitizable Signatures”. In Proc. of the 9th European PKI Workshop: Research and Applications (EuroPKI 2012), pages 178, Springer-Verlag, 2012.

[C99] Coron,J.S.: Resistance against differential power analysis for elliptic curve cryptosystems. In: Koc, Cetin Kaya., Paar, C. (eds.) CHES’99. LNCS, vol. 1717, pp. 292–302. Springer, Berlin, Germany, Worcester, Massachusetts, USA (Aug 12–13, 1999).

[CC2538] Texas Instruments. “CC2538 a powerful system-on-chip for 2.4-GHz IEEE 802.15.4-2006 and ZigBee applications”. http://www.ti.com/product/cc2538 .

[FT12] Fouque, P.A., Tibouchi, M.: Indifferentiable hashing to Barreto-Naehrig curves. In: Hevia, A., Neven, G. (eds.) LATIN- CRYPT 2012. LNCS, vol. 7533, pp. 1–17. Springer, Berlin, Germany, Santiago, Chile (Oct 7–10, 2012).

[FPS12] Faust, S., Pietrzak, K., Schipper, J.: Practical leakage-resilient symmetric cryptography. In: Prouff, E., Schaumont, P. (eds.) CHES 2012. LNCS, vol. 7428, pp. 213–232. Springer, Berlin, Germany, Leuven, Belgium (Sep 9–12, 2012).

[GGLVV14] Galindo, D., Großschadl, J., Liu, Z., Vadnala, P.K., Vivek, S.: Implementation and evaluation of a leakage-resilient ElGamal key encapsulation mechanism. Cryptology ePrint Archive, Report 2014/835 (2014), http://eprint.iacr.org/2014/835.

[GHR99] R. Gennaro, S. Halevi, and R. Rabin. “Secure hash-and-sign signatures without the random oracle”. In Proc. of EUROCRYPT, pages 123–139, 1999.

[GPS08] Steven D Galbraith, Kenneth G Paterson, and Nigel P Smart. Pairings for cryptographers. Discrete Applied Mathematics, 156(16):3113–3121, 2008.

[IOTA] A. Bassi, et. al, Enabling Things to Talk, Springer, ISBN: 978-3-642-40402-3 (Print) 978-3-642-40403-0 (Online).

[K96] Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: Koblitz, N. (ed.) CRYPTO’96. LNCS, vol. 1109, pp. 104–113. Springer, Berlin, Germany, Santa Barbara, CA, USA (Aug 18–22, 1996).

[KJJ99] Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M.J. (ed.) CRYPTO’99. LNCS, vol. 1666, pp. 388–397. Springer, Berlin, Germany, Santa Barbara, CA, USA (Aug 15–19, 1999).

[KP10] Kiltz, E., Pietrzak, K.: Leakage resilient ElGamal encryption. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 595–612. Springer, Berlin, Germany, Singapore (Dec 5–9, 2010).

[L12] H. Lipmaa. “Secure accumulators from Euclidean rings without trusted setup”. In Poc. of ACNS 2012, pages 224–240, 2012.

[L15] Adam Langley. “curve25519-donna”. https://code.google.com/p/curve25519-donna/ .

[L15b] Adam Langley. “A state-of-the-art Diffie-Hellman function”. http://cr. yp.to/ecdh.html.

[LLX07] J. Li, N. Li, and R. Xue. “Universal accumulators with efficient nonmembership proofs”. In Proc. of ACNS 2007, pages 253–269, 2007.

[M05] Maurer, U.M.: Abstract models of computation in cryptography (invited paper). In: Smart, N.P. (ed.) 10th IMA Inter- national Conference on Cryptography and Coding. LNCS, vol. 3796, pp. 1–12. Springer, Berlin, Germany, Cirencester, UK (Dec 19–21, 2005).

[mECC] Tkmackay. “micro-ECC” . http://kmackay.ca/micro-ecc/

[M16] Max Mössinger. “Measurement of Elliptic Curve Cryptography Implementations for Contiki on a RE-Mote”. Master thesis at University of Passau, Cahir of IT-Security, Passau, 2016.

[MEMINF] T. Bowden, B. Bauer, J. Nerin, S. Feng, “The /proc file system”, https://github.com/torvalds/linux/blob/master/Documentation/filesystems/proc.txt

[MOP08] Mangard, S., Oswald, E., Popp, T.: Power analysis attacks: Revealing the secrets of smart cards. Springer (2008).

[MOSW15] Daniel P Martin, Elisabeth Oswald, Martijn Stam, and Marcin Wójcik. A leakage resilient MAC. In Proc. of 15th IMA int. conf. on Cryptography and Coding (IMACC 2015), 2015.

[MPPS14] H. de Meer, H. C. Pöhls, J. Posegga and K. Samelin. “Redactable Signature Schemes for Trees with Signer-Controlled Non-Leaf-Redactions”. In E-Business and Telecommunications, pages 155-171, Springer Berlin Heidelberg, 2014.

[MR04] N. Modadugu and E. Rescorla. The design and implementation of datagram TLS. In NDSS, 2004.

[MT05] Microchip Technology Inc. “MCP3004/3008. Technical report”, Adafruit, 2005. http://www.adafruit.com/datasheets/MCP3008.pdf .

[NUMS14] Internet Engineering Task Force IETF. Nothing up my sleeve (NUMS) curves for ephemeral key exchange in transport layer security (TLS) (draft), 2014. https://tools.ietf.org/html/draft-black-tls-numscurves-00.

[OSMON] Android OS Monitor, https://play.google.com/store/apps/details?id=com.eolwral.osmonitor&hl=en

[P13] H. C. Pöhls. “Contingency Revisited: Secure Construction and Legal Implications of Verifiably Weak Integrity”. In Trust Management VII, pages 136-150, Springer Berlin Heidelberg, 2013.

[P14] Oriol Piñol. “Implementation and evaluation of BSD elliptic curve cryptography.” Master thesis (pre-bologna period), Universitat Politècnica de Catalunya, http://hdl.handle.net/2099.1/24976, November 2014.

[P15] H. C. Pöhls. JSON Sensor Signatures (JSS): End-to-End Integrity Protection from Constrained Device to IoT Application. In Proc. of the Workshop on Extending Seamlessly to the Internet of Things (esIoT), collocated at the IMIS-2012 International Conference (IMIS 2015), pages 306 - 312, IEEE, 2015.

[PK14] H. C. Pöhls and M. Karwe. “Redactable Signatures to Control the Maximum Noise for Differential Privacy in the Smart Grid”. In Proc. of the 2nd Workshop on Smart Grid Security (SmartGridSec 2014), Springer International Publishing, 2014.

[PS14] H. C. Pöhls and K. Samelin. “On Updatable Redactable Signatures”. In Proc. of the 12th International Conference on Applied Cryptography and Network Security (ACNS 2014), Springer, 2014.

[PS15] Henrich C. Pöhls and K. Samelin, “Accountable Redactable Signatures”. In Proc. of the 10th International Conference on Availability, Reliability and Security (ARES 2015), IEEE, 2015.

[PSNB10] Pereira, G.C.C.F., Simplıcio Jr., M.A., Naehrig, M., Barreto, P.S.L.M.: A family of implementation-friendly BN elliptic curves. Cryptology ePrint Archive, Report 2010/429 (2010).

[R15] Noelle T. L. Rakotondravony. “Implementation of an accumulator-based Redactable Signature Scheme on a resource-constrained device”. Final year report at SUP'COM Tunis, co-supervised at University of Passau, August, 2015.

[RD2.1] T. Mouroutis, A. Lioumpas (Eds.), “Use-cases definition and threat analysis”, Dec 2014.

[RD2.2] J. Cuellar (Ed.) et al., “System Requirements and Smart Objects Model”, RERUM Deliverable D2.2, May 2014.

[RD2.3] E. Tragos (Ed.) et al., “System Architecture”, RERUM Deliverable D2.3, August 2014.

[RD3.1] D. Ruiz Lopez (Ed.) et al., “Enhancing the autonomous smart objects and the overall system security of IoT based Smart Cities”, RERUM Deliverable D3.1, March 2015.

[RD3.2] Henrich C. Pöhls and Ralf C. Staudemeyer (Eds.) et al., “Privacy enhancing techniques in the Smart City applications”, RERUM Deliverable D3.2, August 2015.

[RD4.1] E. Tragos (Ed.) et al., “Introducing CR elements into smart objects towards enhanced interconnectivity for Smart City applications”, RERUM Deliverable D4.1, March 2015.

[RD4.2] G. Oikonomou (Ed.) et al., “Advanced techniques to increase the lifetime of smart objects and ensure low power network operation”, RERUM Deliverable D4.2, September 2015.

[RD5.1] R. Munne (Ed.) et al., “Trial scenario Definitions and Evaluation Methodology Specification”, RERUM Deliverable D5.1, September 2015.

[RD5.2] M. Fabregas, A Liñán (Eds.), “Smart object and application Implementation”, RERUM Deliverable D5.2, September 2015.

[RD5.5] M. Fabregas, A Liñán (Eds.), “Smart object and application Implementation”, RERUM Deliverable D5.5, February 2016.

[RECFG] Recommended Elliptic Curves for Federal Government Use, http://csrc.nist.gov/groups/ST/toolkit/documents/dss/NISTReCur.pdf

[RERUM] EU FP7 ICT2013-SMARTCITIES-RERUM. Resilient, Robust and Secure IoT for smart city applications, http://www.ict-rerum.eu

[SHA2] SHA-2 Standard, National Institute of Standards and Technology (NIST), Secure Hash Standard, FIPS PUB 180-2. 2002.

[SF96] Standard, Federal. "1037C." Department of Defence Dictionary of Military and Associated Terms in support of MIL-STD-188 (1996).

[TI13] Texas Instruments, “CC2538 System-on-Chip Solution for 2.4-GHz IEEE 802.15.4 and ZigBee®/ZigBee IP® Applications”, Version C, May 2013.

[ZD10] Zolertia Z1 Datasheet v1.1, March 2010.