Cognitive Radio Experimentation World

Project Deliverable D5.4

Final report on demand-driven extensions and FIRE support actions (update)

Contractual date of delivery:	30-09-2014
Actual date of delivery:	30-09-2014
Beneficiaries:	iMinds, IMEC, TCD, TUB, TUD, TCS, EADS, JSI
Lead beneficiary:	iMinds
Authors:	Jan Hauer (TUB), Mikolaj Chwalisz (TUB), Daan Pareit (iMinds), Wei Liu (iMinds), Ingrid Moerman (iMinds), Michael Mehari (iMinds), Lieven Hollevoet (IMEC), Hans Cappelle (IMEC), Somsaï Thao (TCS), Paul Sutton (TCD), Carolina Fortuna (JSI), Igor Ozimek (JSI), Matevz Vucnik (JSI), Tomaž Šolc (JSI), Tomaž Javornik (JSI), Peter deValck (iMinds), Nicholas Kaminski (TCD), Luiz DaSilva (TCD)
Reviewers:	Mikolaj Chwalisz (TUB), Hans Cappelle (IMEC)
Workpackage:	WP5 – Demand Driven Extensions
Estimated person months:	23
Nature:	R
Dissemination level:	PU
Version	2.11

Abstract: This deliverable describes all the different kinds of extensions that were made. The extensions are an instrument to upgrade the CREW federation with a new set of functionality beyond the basic functionality developed in other CREW work packages. These extensions were made to support the internal use cases (see WP6) and the Open Call 2 and Open Call 3 experiments. Furthermore, FIRE support actions are listed.

Keywords:

cognitive radio, wireless networks, testbed, spectrum sensing, context awareness

Executive Summary

This deliverable describes how the federated CREW test facilities have been extended with a second set of new functionality beyond the basic functionality. It describes the extensions that were made to the portal, to the preceding work in other work packages (WP3 and WP4), to the Connectivity Brokerage and to the specific testbeds. The extensions are highlighted both for the internal use cases and the Open Call 2 experiments. Furthermore, FIRE support actions are listed.

We explained how the federated CREW test facilities have been extended with a second set of new functionality which has been defined in a demand-driven and open way based on the gaps identified, for both the internal use cases (WP6) and the experiments of Open Call 2 (WP7).

To this end, we extended the Connectivity Brokerage Framework for better usage in different scenarios and for improved usability in specific cognitive radio scenarios. A suitable database system was selected and a extension with some IEEE 1900.6 compatible parts was made. Implementation of the Connectivity Brokerage Framework was then also investigated and tested for the w-iLab.t and Log-a-tec testbed.

Furthermore, a framework (ProtoStack/CRime) that allows composing communication services in a dynamic way was proposed and different optimizations were performed within the different testbeds (e.g. OMF, IRIS, GRASS-RaPlat, USRP sensing engine improvement etc.).

To conclude the document, the specific support actions required for the Open Call 2 experiments were also described, as well as a short update on the FIRE support actions.

** The original deliverable submitted at the end of Y3 has been updated at the end of Y4 with a series of testbed extensions that were demand driven or planned and with information referring to Open Call 3. All the new sections and subsection have a * sign at the end of the name so that they can easily be identified in the table of contents and throughout the document.*

The updates of this deliverable include the following. First the updates of the CREW portal are provided, and then the benchmarking functionality has been improved as a result of internal and external demand. The Connectivity Agent has then been extended to support dynamic discovery and connection setup and the ProtoStack tool has been extended with learning functionality. Additional usability improvements have been performed for Iris and the redesign of the VHF and UHF spectrum sensing extension (SNE-ISMUHF-TV) has been completed and the new boards evaluated. Improvements to the wireless management network of the LOG-a-TEC testbed have also been performed as a result of external demand.

The CREW-GENI collaboration continued also in Y4 of the project with further work on the ontology and its integration into TaSoR and a paper preparation.

All OC2 demos needed various level of support in Y4 in view of the final review. We also provide a report on the OC3 demos and the support we provided, each in separate sections. Finally, we report on the CREW Academy, the planned participation in the OfCom trial and the participation in FIRE activities.

List of Acronyms and Abbreviations

6LoWPAN	IPv6 over Low power Wireless Personal Area Networks
ACID	Atomicity, Consistency, Isolation, Durability
ADC	Analog to Digital Converter
AGRAC	Automatic Gain and Resource Activity Controller
AI	Air Interface
API	Application Programming Interface
ASIP	Application-Specific Instruction-set Processor
ATSC	Advanced Television Systems Committee
AWGN	Additive White Gaussian Noise
BAN	Body Area Network
BWRC	Berkeley Wireless Research Center
BS	Base Station
BTS	Base Transceiver Station
CAgent	Connectivity Agent
CB	Connectivity Brokerage
CBAN	Cognitive Body Area Network
CIC	Cascaded Integrator-Comb
CoAP	Constrained Application Protocol
CompNet	Composite Network Agent
CORDIC	COordinate Rotation DIgital Computer
COTS	Commercial Off-The-Shelf
CP	Cyclic Prefix
CR	Cognitive Radio
CREW	Cognitive Radio Experimentation World
CSMA	Carrier Sense Multiple Access
DC	Direct Current
DDR RAM	Double-Data-Rate Synchronous Dynamic Random Access Memory
DFE	Digital FrontEnd
DIFFS	DIgital Front end For Sensing
DVB-T	Digital Video Broadcasting - Terrestrial
ENV	Environment
EVA	Extended Vehicular A
FCC	Federal Communications Commission
FDD	Frequency Division Duplex
FER	Frame Error Ratio
FFT	Fast Fourier Transformation
FIR	Finite Impulse Response
FIRE	Future Internet Research and Experimentation Initative
FPGA	Field Programmable Gate Array
GENI	Global Environment for Network Innovations
GPS	Global Positioning System
GUI	Graphical User Interface
HID	Human Interface Device

ID	Identifier
IO	Input/Output
IPC	Interprocess Communication
I/Q	In-Phase / Quadrature-Phase
IRIS	Implementing Radio In Software
ISM	Industrial Scientific Medical
IVuC	Increase Value until Condition
LAN	Local Area Network
LTE	Long Term Evolution
MAC	Medium Access Control
MVCC	Multiversion concurrency control
NFS	Network File System
NoEE	Number of Experiment Equals
NTP	Network Time Protocol
OFDM	Orthogonal Frequency Division Multiplexing
OFDMA	Orthogonal Frequency Division Multiple Access
OS	Operating System
PA	Platform Agent
PC	Personal Computer
PCB	Printed Circuit Board
PCI	Peripheral Component Interconnect
PD	Probability of Detection
PFA	Probability of False Alarm
PLL	Phase Locked Loop
PMD	Probability of Missed Detection
PRB	Physical Resource Block
PSD	Power Spectral Density
PUB	Publish
RELT	Relative Error of performance is Less Than
RF	Radio Frequency
RFIC	Radio Frequency Integrated Circuit
ROC	Receiver Operating Characteristics
RSSI	Received Signal Strength Indication
RTT	Round Trip Time
Rx	Receiver
SDR	Software Defined Radio
SFTP	Secure FTP
SIMD	Single Instruction, Multiple Data
SIR	Signal Interference Ratio
SNR	Signal to Noise Ratio
SSRuC	Step Size Reduction until Condition
SUB	Subscribe
SUMO	Surrogate Modeling
SUT	System under Test
TCP	Transmission Control Protocol
TDD	Time Division Duplex

TSMC	Taiwan Semiconductor Manufacturing Company
TVWS	Television White Spaces
TWIST	TKN Wireless Indoor Sensor network Testbed
Tx	Transmitter
UDP	User Datagram Protocol
UML	Unified Modeling Language
UniNet	Unified Network Agent
US	Usage Scenario (see D2.1 for definition)
USB	Universal Serial Bus
USRP	Universal Software Radio Peripheral
VCC	Virtual Control Channel
WARP	Wireless Open Access Research Platform
WHIPP	The WiCa Heuristic Indoor Propagation Prediction
WInnF	Wireless Innovation Forum
WLAN	Wireless Local Area Network
WSAP	White Space Access Point
WSD	White Space Device
WSN	Wireless Sensor Network
ZMQ	ZeroMQ

List of Figures

Table of Contents

1 Introduction

In the CREW project demand-driven extensions are the tools to dynamically adapt the CREW federation to needs identified during the course of the project. While some of these extensions were envisioned during the project planning, not all needs were evident at that time. In addition, since external experimenters were/will be joining the project as a result of open calls during the project course (M12/M24/M36) a mechanism to extend the project with new functionality to better support these experimenters was required. This was the 'raison d'être' for Work Package 5 (WP5).

In deliverable D5.2 the extensions were described that were developed in Year 2 to support the internal use cases and the experiments of the Open Call 1 experimenters. Within this deliverable, we describe the further enhancements to support internal use cases (WP6) and we list the extensions which were needed for the Open Call 2 experiments (WP7).

We explained how the federated CREW test facilities have been extended with a second set of new functionality which has been defined in a demand-driven and open way based on the gaps identified, for both the internal use cases (WP6) and the experiments of Open Call 2 (WP7).

To this end, we extended the Connectivity Brokerage Framework for better usage in different scenarios and for improved usability in specific cognitive radio scenarios. A suitable database system was selected and an extension with some IEEE 1900.6 compatible parts was made. Implementation of the Connectivity Brokerage Framework was then also investigated and tested for the w-iLab.t and Log-a-tec testbed.

Furthermore, a framework (ProtoStack/Crime) that allows composing communication services in a dynamic way was proposed and different optimizations were performed within the different testbeds (e.g. OMF, IRIS, GRASS-RaPlat, USRP sensing engine improvement etc.).

To conclude the document, the specific support actions required for the Open Call 2 experiments were also described, as well as a short update on the FIRE support actions.

** The original deliverable submitted at the end of Y3 has been updated at the end of Y4 with a series of testbed extensions that were demand driven or planned and with information referring to Open Call 3. All the new sections and subsection have a * at the end of the name so that they can easily be identified in the table of contents and throughout the document.*

The updates of this deliverable include the following. First the updates of the CREW portal are provided (Section 2.1), and then the benchmarking functionality has been improved as a result of internal and external demand (Section 2.2.3). The Connectivity Agent has then been extended to support dynamic discovery and connection setup (Section 2.3.6) and the ProtoStack tool has been extended with learning functionality (Section. 2.4.6). Additional usability improvements have been performed for Iris (Section 2.5.5) and the redesign of the VHF and UHF spectrum sensing extension (SNE-ISMUHF-TV) has been completed and the new boards evaluated (Section 2.5.6). Improvements to the wireless management network of the LOG-a-TEC testbed have also been performed as a result of external demand (Section 2.5.7).

The CREW-GENI collaboration continued also in Y4 of the project with further work on the ontology and its integration into TaSoR and a paper preparation (Section 2.6.2.2).

All OC2 demos needed various level of support in Y4 in view of the final review (Section 3). We also provide a report on the OC3 demos and the support we provided, each in separate sections (Section 4 and 5). Finally, we report on the CREW Academy (Section 6), the planned participation in the OfCom trial (Section 7) and the participation in FIRE activities (Section 8).

2 Demand-driven Extensions Derived From Internal Use Cases*

This section describes the extensions that were developed during the third year of the CREW project as a result of the shortcomings identified in the CREW internal use cases (WP6). The section has then been updated at the end of the fourth year of the project with results of the development planned for that year (see starred * subsections).

2.1 CREW Portal & CREW Repository*

The portal (http://www.crew-project.eu/portal) and the repository (http://www.crew-project.eu/repository) were kept up to date during CREW's third year. Some of the notable changes are the following:

- Linking to static files for processing the CREW Common Data Format (CDF) has been replaced by linking to a github repository which has always the latest scripts available (see also 2.5.1.2).
  http://www.crew-project.eu/repository/scripts
- Information on running an experiment on iMinds' w-iLab.t testbed has been updated to reflect the new way to request an account and to use the Emulab environment
  http://www.crew-project.eu/portal/wilab/basic-tutorial-your-first-experiment-w-ilabt
- The description of the WARPs and the mobile robots in iMinds' w-iLab.t testbed has been added
  http://www.crew-project.eu/content/configuration
  http://www.crew-project.eu/content/warp-usage

* The following updates to the CREW portal have been made or are underway in Y4:

The portal has been or will be updated as follows. First, the video material from the training days, from the core partner and OC demos at various events as well as the CREW Twitter and YouTube channels have been added. The documentation regarding the Imec sensing engine and the documentation the new type of sensing recently developed will be added. The TCD facility is being upgraded towards cloud-based access and info on the way of using the new system and link to the new access portal will be added. Also information and code about the new radar module under development will be added. The testbed description will be updated to reflect the new changes of the components in TWIST. TUD will provide an update regarding the new hardware and the remote desktop connection. A link to the WInnF XCVR API hosted on the Wireless Innovation Forum will be added, and the LOG-a-TEC portal with info about the new extension board and the new software stack corresponding to LOG-a-TEC 2.0 will be updated. Finally, we are looking whether promoting the EADS aircraft mock-up on the portal is possible since, due to company policy, the access to it is subject to very strict rules.

2.2 Continuation of Other Work Packages

2.2.1 Creating the Federation (WP3)

2.2.1.1 Maintenance and update of the WInnF Transceiver API source code

CREW attended the 73rd Wireless Innovation Forum (WInnF) Working Meeting, from 31st October to 2nd November 2012, held by Harris Corporation in Melbourne, Florida. CREW presented there the initial software upload realized, then discussions were held on the real-time limitations of the solution based on a widely available and used platform such as Ettus Research USRP2 board. The current solution requires indeed about 3 ms of anticipation in bursts creation commands and the Absolute Timing programming mode implementation is difficult.

Mr Matt Ettus put CREW in contact with Mr Balint Seeber who is Ettus Research Expert in our issue. Direct interactions between CREW and him took place in January and February 2013, as a follow-up of the WInnF findings. Those exchanges enabled to identify promising perspectives concerning the degree of real-time performance achievable using USRP solutions, by taking advantage of digital frequency tuning, that enables, provided useful bandwidth is smaller than the digitized bandwidth, to have very fast frequency switching that can mitigate the relatively slow control reactivity from PC to USRP board. Evaluations concluded that **some 100 µs** would be a target for a minimum required anticipation for a burst creation.

2.2.1.2 Finalize the automatized support for common data format in the LOG-a-TEC testbed

To enable collaboration with other testbeds in the CREW federation, support for the CREW Common Data Format (CDF) has been developed for the LOG-a-TEC testbed. Spectrum sensing experiments that have been described in CDF files can be re-run on the LOG-a-TEC testbed without any additional programming. Results of such experiments can also be saved back to a CDF file.

As described in the deliverable D3.2 under "VESNA interfaces", the high-level interface to wireless sensor nodes deployed in the LOG-a-TEC testbed is a HTTP-like protocol that is accessible through a gateway on the Internet.

An open source Python module library[1] called vesna-alh-tools has been developed that provides a convenient interface to the testbed through a HTTP-like protocol. By using it, a program written in the Python programming language running on the experimenter's computer can remotely control the sensor nodes in the LOG-a-TEC testbed. At the base of this library is the vesna.alh module that provides convenience functions for communicating with the sensor nodes through the LOG-a-TEC Internet gateway. On top of this module a CDF support layer vesna.cdf provides abstraction over the raw resources that are exposed by sensor nodes in the testbed and adapts the LOG-a-TEC specific interface to the common CREW testbed schema. The vesna.cdf.xml module then provides CDF parsing and serialization utilities. Finally, a run¬_cdf_experiment executable uses this infrastructure to directly run an experiment described in a CDF file without the need for any additional programming. This layered approach allows the experimenter to choose between the level of testbed independence and LOG-a-TEC specific features that best suit his needs. Specifically, the Python interface allows for a level of automatic control over the experiment that is not directly describable within the CDF format.

Because of the specific implementation details of the LOG-a-TEC testbed, several metadata fields needed to be added to the CDF in order for an experiment description to be applicable to VESNA devices. Specifically, in order for a CDF file to be directly usable with the run_cdf_experiment tool, it must contain information about device access like the gateway URL to use and VESNA management network addresses. Due to the rigid nature of the CDF's XML schema, these metadata fields have been added as JSON-encoded strings (prefixed with "Additional VESNA metadata follows") within free-form string sections of the CDF file. Should the CDF format be expanded to support these fields in the XML schema itself, these additional JSON-encoded fields could later be replaced in favour of native XML tags.

[1] https://github.com/sensorlab/vesna-alh-tools

2.2.2 Benchmarking the Federation (WP4)

2.2.2.1 The WHIPP tool extension for REM visualization

The WiCa Heuristic Indoor Propagation Prediction (WHIPP) tool is a heuristic network planner, developed and validated for indoor environments [1]. The model has been constructed for the 1.8 - 2.6 GHz band and its performance has been validated with a large set of measurements in various buildings.

In the third year of CREW, the WHIPP tool is extended from a simple predictor to a monitor, which can be used to visualize the Radio Environment Map (REM) in the w-iLab.t testbed. The backbone of this tool relies on the distributed spectrum sensing system, as shown on the left side of Figure 1. We use several sensing devices, including USRP, IMEC sensing engines and regular WiFi devices together for monitoring the w-iLab.t environment and collect the measurements into a central database. Before combining the measurements from different devices, we need to first calibrate the devices so that they don't measure signal with different offset. Apart from that, the sensing devices have different measurement capabilities, some devices produce measurement faster than others; therefore we apply certain data fusion mechanisms to ensure the speed of data production by different sensing engines are comparable.

The REM interface is part of the original WHIPP tool, a new "Monitor Wilab" tab is added on the web interface. When clicking on it, the w-iLab.t Zwijnaarde testbed topology is loaded. The tool reads the RSSI of the selected nodes from the central database regularly. The discrete RSSI results are interpolated to have a realistic view of the REM on the testbed. A snapshot of the final result of the interpolation is presented on the right side of Figure 1.

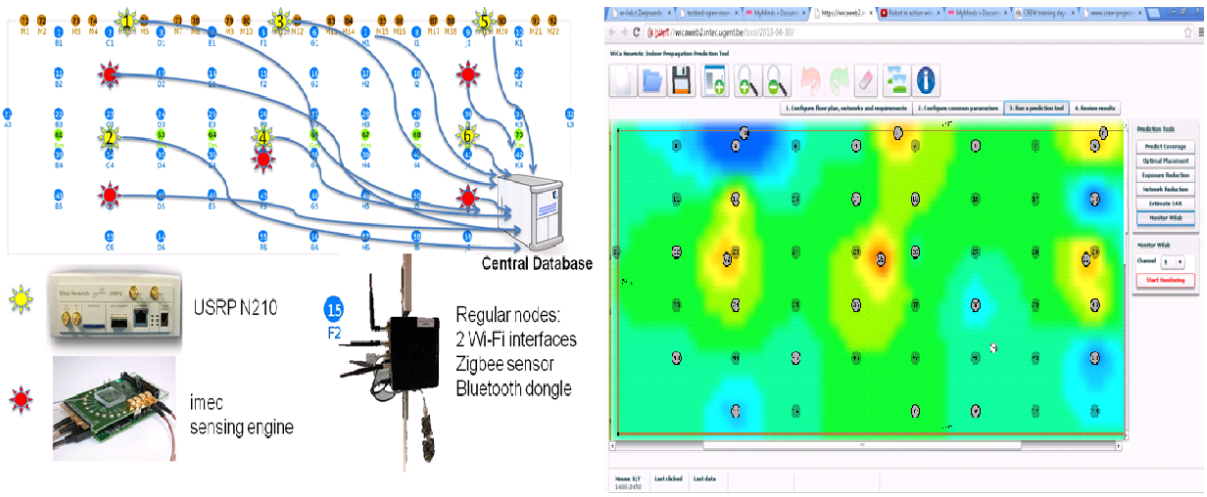


Figure 1 Distributed spectrum sensing and the monitor interface on WHIPP tool

2.2.2.2 SUMO tool

Out of the box, Surrogate Modelling SUMO toolbox [2] is used as complete multi-dimensional optimizers. It is targeted to achieve accurate models of a computationally intensive problem using reduced datasets. From the reduced datasets, the tool generates accurate Surrogate Models to evaluate design objectives.

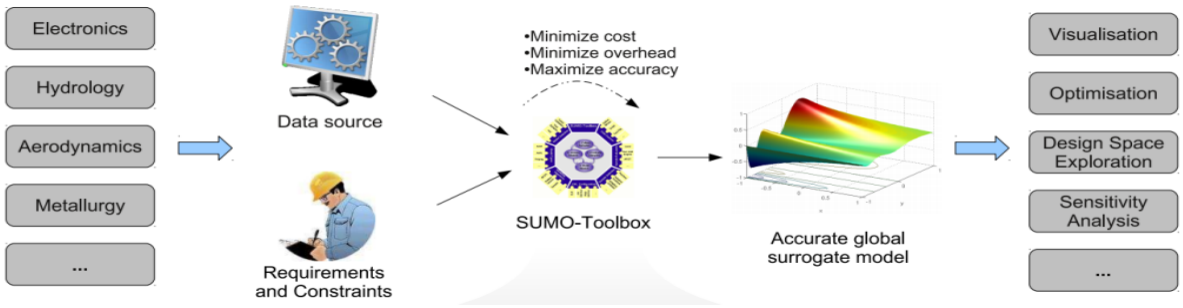


Figure 2 The process of generating accurate surrogate model

The SUMO toolbox bundles both the control and optimization functions together. The control function sitting at the highest level manages the optimization process with specific user inputs. The figure below describes SUMO tools in a nutshell highlighting the control and optimization functions together.

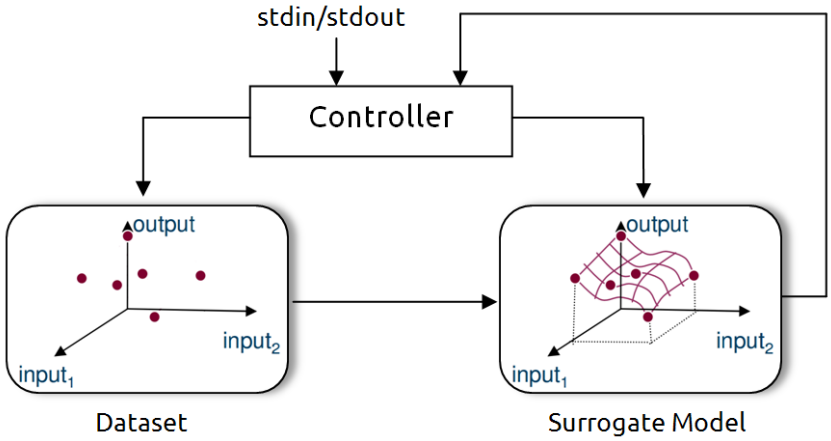


Figure 3 Out of the box SUMO tools in a nutshell view

From the figure above it can be seen that the controller manages the optimization process starting from a given dataset (i.e. initial samples + output performance) and generates a surrogate model. The surrogate model approximates the dataset over the continuous design space range. Next, the controller predicts the next design space element from the constructed Surrogate model to further meet the optimization's objective. Depending on the user's configuration, the optimization process iterates until conditions are met.

The aim is to use SUMO tools as a standalone optimizer and put them inside an experimentation framework. This means from out of the box SUMO tools, the loop is broken, the control function is removed and clear input/output interfaces are created to interact with the controlling framework. The figure below shows how modified SUMO tools are integrated in the wireless testbed.

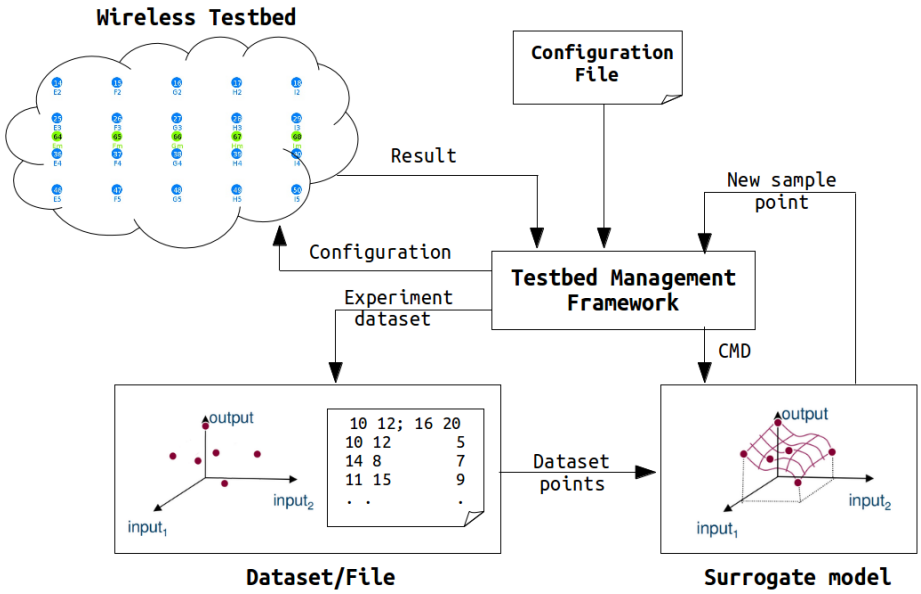


Figure 4 Integration of modified SUMO tools in the wireless testbed

The figure above shows the use of the testbed management framework instead of the default controller. Suppose we are in a context of a wireless experiment, concerning two parameters: transmit power and the node location. The goal is to find the optimum combination of those two parameters to achieve the maximum throughput. There is a format agreement between SUMO toolbox and the testbed management framework. After the initial datasets are achieved, SUMO toolbox determines the parameter set for the next experiment, and writes the parameters in the dataset file, as shown in the figure above. The testbed framework performs the experiment, and appends the result on the same row in the data file (in this case it is the throughput result). This iteration goes on until the stopping criteria are reached. More details of this example is described in Figure 5 and the paragraph below, and a more complex experiment on the w-iLab.t testbed with SUMO optimization is described in D6.3, Section 2.3.

This framework performs the same tasks which were already implemented by the previous controller except additional tasks like experimentation on the wireless testbed, storing the dataset on a separate file, and reading experiment configuration from a file. It should be understood very well that the operation of SUMO tools has not changed at all except replacement and addition of a few working blocks. A more general pictorial presentation on how SUMO tools work is also presented on the next figure.

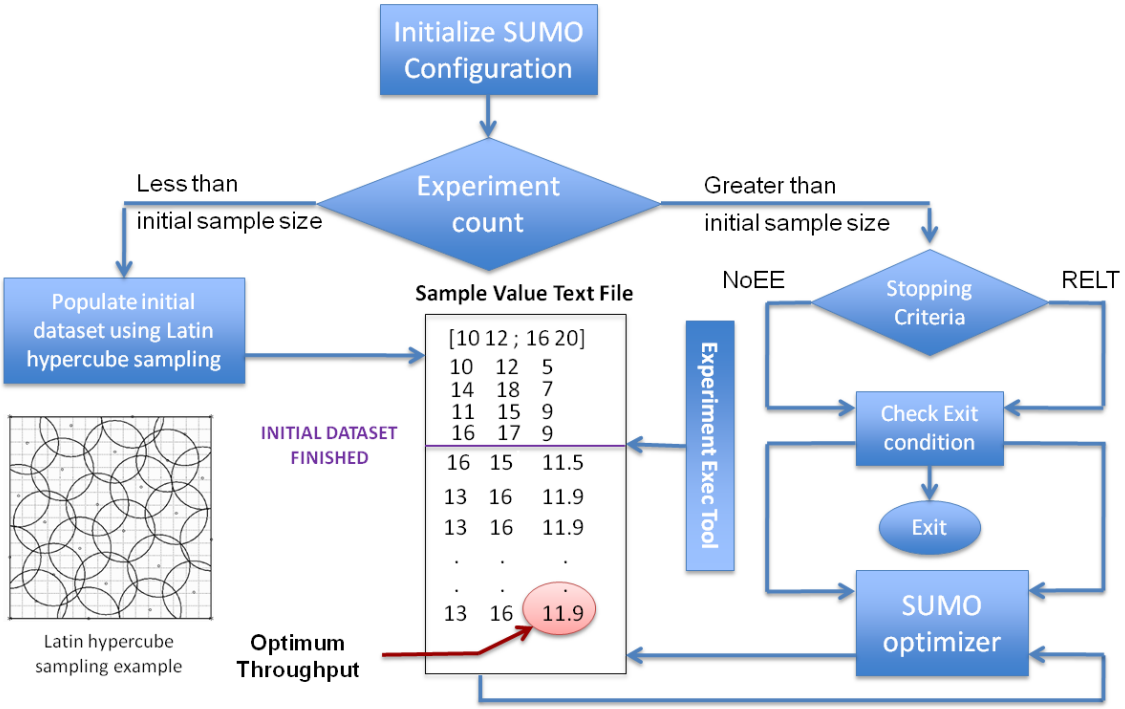


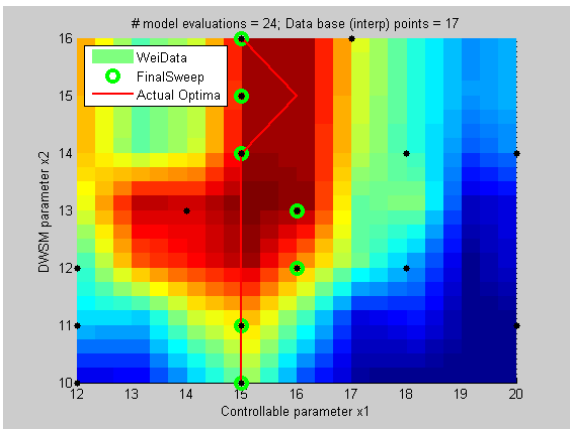
Figure 5 Complete modified SUMO toolbox optimization over two dimensional design space

In the figure above, a complete optimization using the modified SUMO tools is presented. The SUMO tools optimize a two dimensional design space (i.e. transmit power 10dbm to 16dbm and transmitter location id 12 to 20) problem.

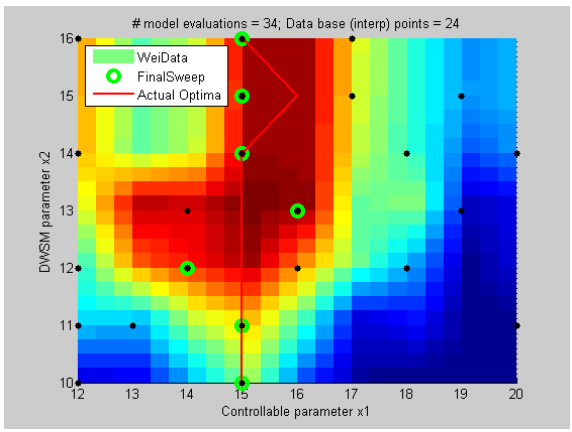
Before SUMO tools start the optimization process, it requires a minimum number of initial samples which is used to generate the first surrogate model. The figure above shows four selected initial dataset pairs (i.e. [10 12], [14, 18], [11, 15], [16, 17]). Latin hypercube sampling is used to generate the initial samples which verifies that the samples are equally distributed across the design space range.

Next, the experiment is conducted at each initial sample and the dataset (i.e. initial samples + output performance) is fed to the optimizer. The SUMO tools first generate the surrogate model and next calculate a new sample point to reach the global optimum. Using the calculated sample point, we do a new experiment and update the dataset. As optimization progresses, the SUMO tools approach the global optimum point. The figure below shows the different steps during SUMO optimization process.

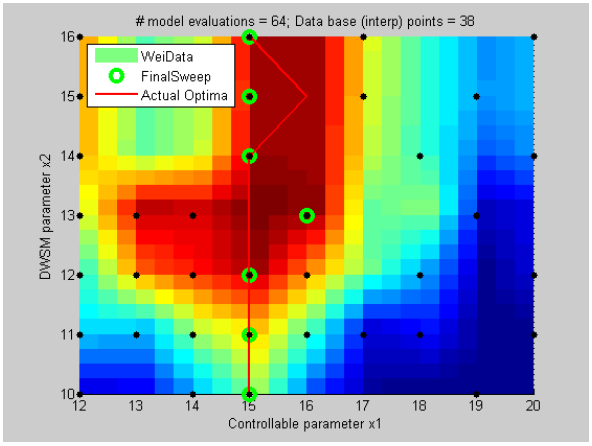
Optimum throughput = 11.9 Mbps



Optimum throughput = 12.184 Mbps



Optimum throughput = 12.37 Mbps



Optimum throughput = 12.419 Mbps

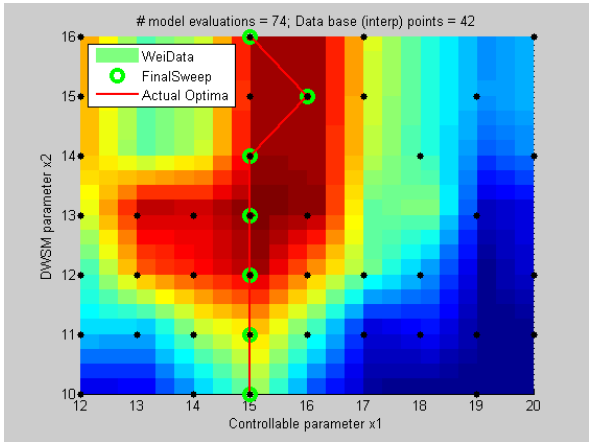


Figure 6 The different steps during SUMO optimization

From the above figures we see that as the number of iterations increases the simulation optimum coincides with the global optimum. The first figure shows 17 experiments and optimum performance of 11.9 Mbps. The second figure shows 24 experiments and optimum performance of 12.184 Mbps. The third figure shows 38 experiments and optimum performance of 12.37 Mbps. The last figure shows 38 experiments and optimum performance of 12.419 Mbps. However useful the above example was, it was not elaborate enough to fully describe SUMO tools. Thus a detail analysis on a "wireless press conference" optimization problem is presented in Deliverable 6.3 "US3-Horizontal Resource Sharing in ISM bands".

We end the optimization after a stopping criteria is met. Two possible stopping criteria are when the Number of Experiment Equals (NoEE) a given value and when the Relative Error of performance is Less Than (RELT) a given threshold and both are implemented in "experiment execution" tool.

2.2.2.3 Benchmarking framework extension

Benchmarking in the scope of wireless networks focuses on the methodology and approach of conducting the experiment. In the scope of CREW, a benchmarking framework to meet this demand is designed with a number of tools. The benchmarking tools developed are experiment definition, experiment execution, and result analysis. Each separate tool plays a role in the process of wireless network benchmarking. A brief summary of the CREW benchmarking framework is included here; more details are elaborated in section 2.2 of deliverable D4.3.

The experiment definition tool is used to define, configure, and make changes to wireless experiments that are to be conducted on the testbed. The second tool, the experiment execution tool, schedules and executes the already defined experiment. This section also defines the optimizer selection and different configurations that are also passed along. The last tool, the result analysis tool, is used to conduct performance comparison of an experiment to a reference solution. In sum, an experimenter defines an experiment description, schedules and executes his experiment, and finally analyzes its performance.

The set of benchmarking tools that were developed in previous years are further improved with a number of functionalities.

Updates on the "experiment definition" tool

The first update made in the "experiment definition" tool is the merging of the Solution Under Test (SUT) and Environment (ENV) configuration files. Now the XML configuration file contains both the SUT and ENV configuration files together and the experiment definition tool accepts one configuration file instead of two.

Second, the project name and experiment name sections are included in the "experiment abstract" section. This modification is connected to a recent change in the w-iLab.t Zwijnaarde testbed development. This is related to the control and management framework of wireless testbeds. Most of the European research projects involving wireless testbeds have previously implemented the cOntrol and Management Framework (OMF). However, recently the OMF loading image functionality is disabled, and handled by *Emulab* [9]. The reason for this migration is that Emulab is more advanced for loading images on a node. Besides, other parts of the the iLab.t testbed (which w-iLab.t is part of) already use Emulab as its primary management framework and a unified interface internally was envisaged. Note that the normal experiment flow is still managed by OMF, as is desired. Emulab is a network testbed, giving researchers a wide range of environments in which to develop, debug, and evaluate their systems. Emulab testbed management follows a structure where each experiment is described by a unique project name and experiment name and hence the need for the changes.

The last update made on this tool is on the "Included Nodes" section. In the previous implementation, a group could only configure one node but the current implementation supports more than one. Furthermore, two important functionalities added in this section are "Send Message" and "Execute Program". The Send Message functionality gives the option to send dynamic messages to a running process. A typical example is a running process waiting for user input to get dynamic messages. The second functionality, Execute Program, lets the experimenter execute a program at a given time. The combined functionality of the two helps create more advanced scenarios.

Updates on the "experiment execution" tool

In this tool, we did the most important changes over the whole range of benchmarking tools. The first change made was the categorization of optimization problems. It is a fact that a single optimization method cannot solve all types of problems. Experimenters depend heavily on specific optimization methods for their specific problems. To this end, three optimizers are selected and plugged into the "experiment execution" tool. These are "Step Size Reduction until Condition" (SSRuC), "Increase Value until Condition" (IVuC), and "Surrogate Modeling" (SUMO) tools.

SSRuC is already covered in deliverable D4.3 and we only discuss IVuC and SUMO tools optimizers. IVuC optimizer is designed to solve problems which show either increasing or decreasing performance along the design parameter range. A typical example could be bandwidth optimization of a WiFi experiment. By setting datagram error rate as a performance parameter, the highest bandwidth is searched and optimized for certain datagram error rate threshold.

The algorithm used by IVuC optimizer is similar to the SSRuC optimizer. However, the main difference between the two is that the SSRuC optimizer performs a complete experimentation cycle before locating the local optimum value whereas the IVuC optimizer performs a local optimum performance check after the end of each experiment. Later on, both approaches refine their design parameter range and restart the optimization process to further tune the design parameter. The figure below shows the different steps involved in IVuC optimizer. In the context of one-dimensional optimization, the IVuC means linear searching for an optimum area and zoom in on this area in the next iteration. Three iteration of one-dimensional optimization is illustrated in Figure 7.

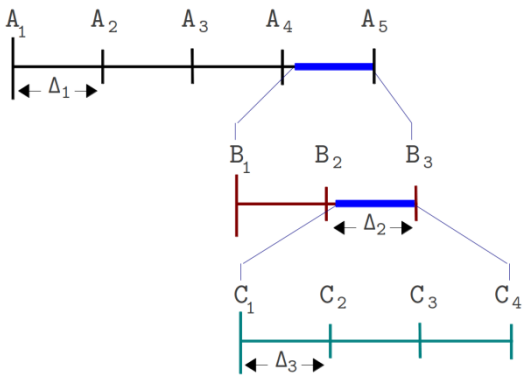


Figure 7 The different steps of an IVuC optimizer over a design parameter range A1 to A5.

The last optimizer supported is SUMO tools and unlike the previous two cases it performs multi-dimensional optimization. Usually researchers optimize problems that contain more than one parameters and their complexity urges the use of multi-parameter optimizers. SUMO tools in particular are targeted to achieve accurate models of computationally intensive problems using reduced datasets. A complete explanation of the SUMO tools, however, is vast so we devoted a separate section for it. Please refer to the previous subsection for a detailed explanation of SUMO tools.

The current needs of most researchers are covered by the three optimizer sets. However, when the need for a new set arrives, it is as simple as plugging it in and integrating it with the benchmarking tools.

Next, upon selection of a specific optimizer, unique configuration settings are automatically populated. A configuration example of the SUMO tools optimizer is shown below.

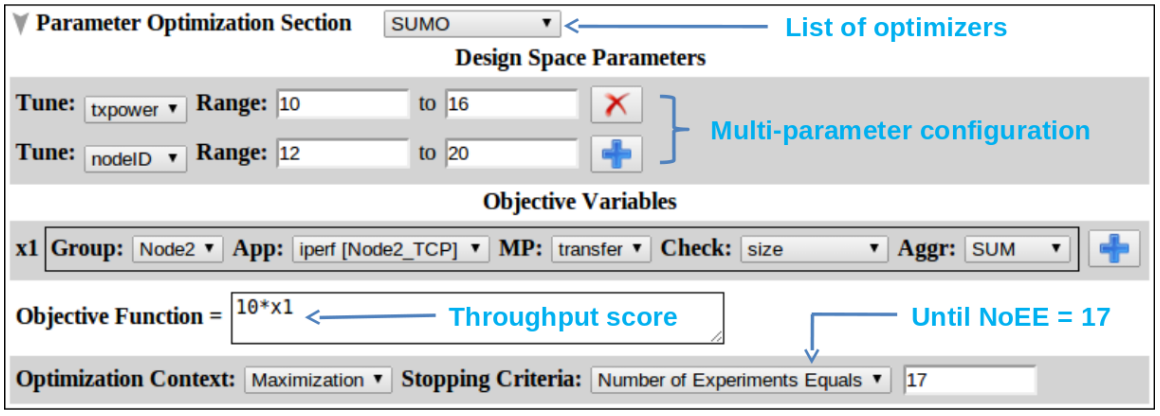


Figure 8 SUMO optimizer configuration at glance

In the figure above, a two dimensional parameter is optimized to maximize throughput of a TCP communication. The experiment is iterated within design parameter space (i.e. TxPower 10 to 16 and nodeID 12 to 20) and after NoEE equals 17, the optimization is stopped. Another addition to this section is the "objective function text to expression" interpreter. The objective function is defined using text description and parsed into an expression for objective function evaluation.

Update on the "Result Analysis" tool

The "Result Analysis" tool is a new addition, which was conceptual in previous implementation. It contains graphical analysis and score calculation and comparison sections. The graphical analysis section compares multiple experiment data, displaying it graphically and thus analyzing group performance. The figure below shows a graphical comparison of datagram vs. time for six consecutive experiments.

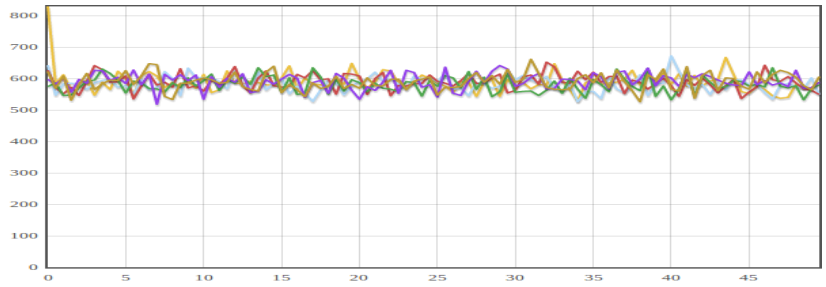


Figure 9 Datagram vs. time plot for six consecutive experiments.

Moreover, this section allows users to define a number of performance metrics such as sample average, sample standard deviation, etc. In the "Result Analysis" section of the benchmarking framework, score calculation and comparison, is used for objective comparison of experiment data. A score is defined as an arithmetic combination of performance metrics. A performance metric is a measure where a physical meaning is given to an experiment data. Mean Opinion Score (MOS), throughput and transmission exposure are examples of performance metrics. An optimization problem to maximize audio quality and minimize transmission exposure, for example, defines a score out of MOS and transmission exposure metrics.

2.2.3 Improve the benchmarking functionality*

Wireless experimenters often find themselves in need of knowing what is happening in the wireless environment when running an experiment on the w-iLab.t. A common way to do so is to log packet statistics on individual devices and then manually inspect the logged information. However, the device may fail to detect the packets from an interferer using another technology, which eventually affects the user's experiment. By far, unknown interference is actually the major difficulty imposed on a wireless network solution. Through this extension that improved the benchmarking functionality of the wireless environment, we make monitoring much easier and reliable for the users. This helps the experimenters to understand what is happening in the wireless environment during the experiment, and explain the results or re-run the experiment as needed.

2.2.3.1 Demo scenario*

While a network solution is often experimenter-specific, the need to have a stable and clean wireless environment for trials is universal. Therefore, our demo will show the benchmarking of the wireless environment in which an experiment has been carried out, rather than benchmarking the solution itself – of which the facility operator OR the benchmarking tool is not aware.

Our demo scenario is composed of four nodes. Two of which are part of the solution under test (SUT) and two are creating the background environment. In our SUT, one node (i.e. client) sends an increasing and continuous iperf stream to a second node (i.e. server) in order to check the limit of its TCP bandwidth on a specific channel. While the SUT is undergoing, the other two nodes create interference.

2.2.3.2 Metrics describing the wireless environment during the experiment*

The extended bechmarking tool generates the following metrics during the experiment and provides them together with a visualization tool to the experimenter:

- **Channel Occupancy Ratio** (COR): Measures the amount of energy percentage of a wireless channel above a certain threshold.
- **RSSI visualization**: It uses the measured discrete RSSI at particular nodes to construct a continuous radio environment map. It enables the experimenter to (re-)visualize experiments.
- **Packet count**: A metric that keeps track of the amount of packets that are not part of the Solution Under Test (SUT). Therefore the larger the packet count, the more interference on SUT. Moreover, this metric is technology dependant as one needs to count the number of fully decoded packets.

The tool can visualize traces from experiments and compute the above metrics either in real time or in play back mode.

2.2.3.3 Implementation*

*2.2.3.3.1 General architecture**

The general architecture of the tool is shown in **Figure 10** below. It consists two parts: a web interface for visualization as the frontend, and a set of tools controlled by to monitor the wireless environment as the backend.

The components in our wireless monitoring system can be classified in two categories: front-end components interact directly with the experimenter while back-end components are responsible for the actual monitoring and processing. Figure 10 provides a general overview of this system with the different components highlighted in red. These components will be further discussed in the following sections.

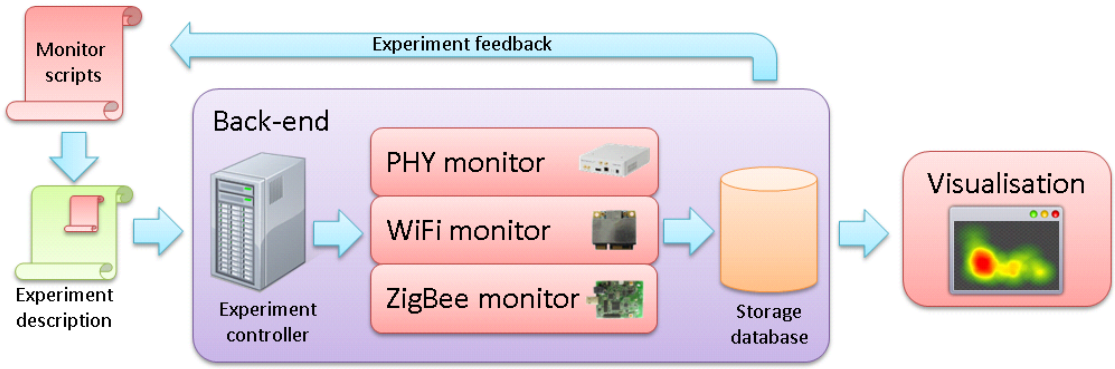


Figure 10 General architecture

*2.2.3.3.2 Frontend**

The frontend components are responsible for the interaction between the monitoring system and the experimenter. The monitoring components are controlled by several monitor scripts and their output can be visualized or used to provide feedback to the experiment itself.

2.2.3.3.2.1 Measurement scripts*

To enable the wireless monitoring components, the experimenter must include the relevant monitoring script in his experiment description, and provide a couple of configuration parameters:

- **Monitor nodes**: the experimenter must specify the nodes that will be used to perform the monitoring. Depending on the experiment topology, the experimenter is free to choose the optimal monitor locations.

- **Monitor channels**: with the commodity hardware used by our monitor components not all wireless channels can be monitored simultaneously. Therefore the experimenter must specify the channels of interest.
- **Experiment nodes**: in order to filter out traffic generated by the experimenters' nodes themselves, a list of nodes involved in the experiment must be provided.

These parameters allow the scripts to configure the monitoring nodes and start the monitoring process. Monitor data is stored in a database and can be queried to provide real-time feedback during the experiment or used to visualize the experiment afterwards.

*2.2.3.3.3 Backend**

The backend tools use **OMF (cOntrol and Management Framework [11])** as their control interface. Three types of backend tools are developed towards improving the benchmarking functionality. These are 1) Physical layer WiFi spectral scanning and USRP/imec sensing engine, 2) Link layer WiFi and 3) Zigbee packet sniffing.

2.2.3.3.3.1 WiFi spectral scanning*

A special feature present in Atheros WiFi chipsets (i.e. AR92xx and AR93xx) is the ability to report spectrum samples to the user space [31]. A normal WiFi chipset provides the bits of a decoded WiFi transmission but the spectral scanning feature of Atheros chipsets also provides the FFT bins of a single WiFi channel. The FFT bins provided are the absolute magnitude |i|+|q| present on the subcarriers of an OFDM signal and depending on the mode of operation (i.e. 20MHz or 40MHz band), 56 and 128 FFT bins are returned per single spectral sample. This feature involves a different operation in hardware and thus both operations (i.e. normal and spectral scanning) cannot be achieved at the same time.

Principle of operation

Looking at the complete spectral scanning vertical, we see four layers namely Hardware, ath9k, debugfs and userspace. Figure 11 shows the details of the spectral scanning stack.

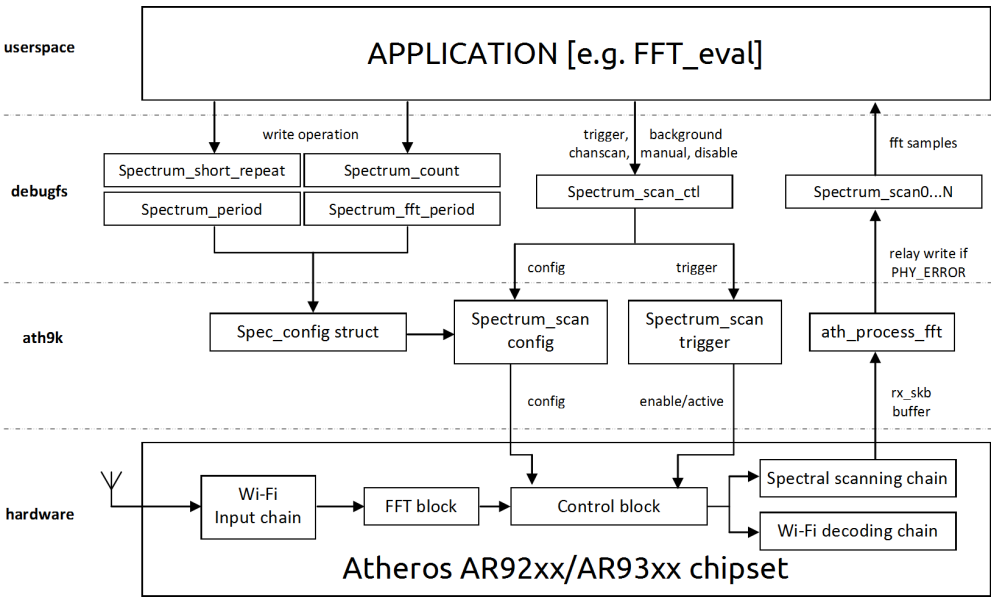


Figure 11 Atheros AR92xx/AR93xx spectral scanning stack

Starting from the Hardware layer, the Atheros chipset samples the ISM spectrum and pre-process the data to get the FFT samples. The FFT samples at this point can choose two different paths depending on the mode of operation. The first mode is the normal WiFi operation and the second mode is the spectral scanning mode. The second mode stops further post-processing on the FFT samples (such as equalization, constellation mapping, demodulation, ...) and forwards it to the ath9k kernel space device driver. The ath9k declares the packet as a PHY-ERROR since it does not comply with a standard WiFi preamble sequence pattern. This time, ath9k does not discard the packet since the spectral scanning flag (i.e. spectral_scan_ctl) is set and it stores it to one of the debugfs files (i.e. spectral_scan**x**). This file actually corresponds to circular buffer that is stored in the random access memory (RAM) and is created and configured with in the ath9k kernel space. After that, applications can make use the spectral samples after mapping the debugfs file to the userspace. One other thing the ath9k does every time it receives a spectral sample is update the spectral frame counter. The spectral frame counter can be useful when checking the amount of spectral samples read by the userspace and dropped by the kernel space [32].

Going back to the higher layer in Figure 11, an application passes special configuration commands in order to start the scanning and collection of spectral samples. These commands are written to specific debugfs files and finally trigger the scanning operation.

Spectral scanning steps [31]

1. Configure spectral parameters by writing appropriate values to debugfs files (i.e. spectral_short_repeat, spectral_period, spectral_count and spectral_fft_period, spectral_scan_ctl)
   - spectral_short_repeat: controls whether the chip is in spectral scan mode for 4 usec (enabled) or 204 usec (disabled)
   - spectral_period: when active, time period between successive spectral scan entry points (period*256*Tclk). Tclk = 44MHz for HT20 operation, 88MHz for HT40 operation
   - spectral_count: number of scan results requested.
   - spectral_fft_period: when active and triggered, PHY passes FFT frames to MAC every (fft_period+1)*4uS
2. Select spectral mode of operation by writing to the spectral_scan_ctl debugfs file. Supported modes of operation are
   - disable: spectral scan is disabled
   - background: spectral scans samples are returned endlessly from the currently configured channel. It is running while the hardware is not busy with sending/receiving
   - manual: as many spectral scan samples as configured in spectral_count are returned from the current channel.
   - chanscan: as many spectral scan samples as configured in spectral_count are returned for each channel when performing a scan.
3. Trigger spectral scanning by writing "trigger" command to the spectral_scan_ctl debugfs file.
4. Read spectral fft samples from spectral_scan**x** debugfs file

2.2.3.3.3.2 Zigbee packet sniffing*

Zigbee is a specification for a suite of high-level communication protocols using small, low-power digital radios based on an IEEE 802.15.4 standard for personal area networks. Zigbee operates in the industrial, scientific and medical (ISM) radio bands; 868 Mhz in Europe, 915 Mhz in the USA and Australia and 2.4 GHz in most jurisdictions worldwide. Data transmission rates vary from 20 to 250 kbps.

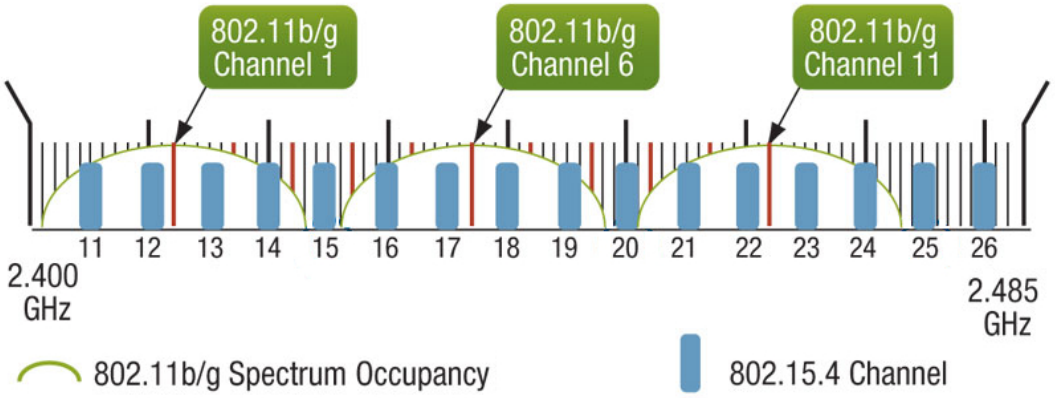


Figure 12 Zigbee 2.4GHz ISM band channel allocation overlapping with WiFi 802.11b/g, adapted from [34]

Looking into Figure 12, 16 Zigbee channels are allocated at 2.4GHz ISM. These are numbered from 11 up to 26. Each channel occupies a 5MHz bandwidth and the total bandwidth coverage spans from 2405 MHz up to 2480 MHz.

Principle of operation

Since Zigbee packet sniffing tools ate not widely available, we created a custom Zigbee packet sniffing module used in the w.ilab.t testbed as shown in Figure 13.

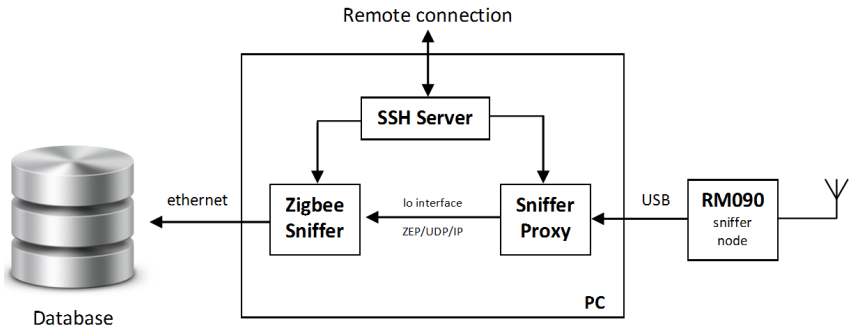


Figure 13 Zigbee packet sniffing path

The Zigbee packet sniffing starts from the sensor/sniffer node at the right hand side which the RM090 sensor nodes are used for this purpose. Having the baudrate set to 500kbps on the RM090 devices, all 802.15.4 traffic bound to 250kbps are captured without a problem. The sniffing application installed on the RM090 sensor node captures all Zigbee traffic and sends it to the PC via a USB connection. The sniffer proxy running on the PC bridges the sensor node with the Zigbee sniffer by creating a virtual serial port to collect the data from RM090 and forwards the data to the loop back (lo) interface. The Zigbee sniffer in return waits for Zigbee Encapsulated Packet (ZEP) frames from the loopback (lo) interface and stores per frame statistics into the database. All control commands go via a remote connection from external users through the SSH server. The frames arriving at the Zigbee sniffer application are encapsulated inside a ZEP version 2 format and its header structure looks as follows.

Preamble	Version	Type	Channel ID	Device ID	CRC/LQI Mode	LQI value	NTP timestamp	Seq no.	Reserved	Length
2 bytes	1 byte	1	1 byte	2 bytes	1 byte	1 byte	8 bytes	4 bytes	10 bytes	1 byte

		byte								

From the ZEP header above, channel ID and LQI fields are stored into database. Following the ZEP header, the 802.15.4 frame data follows which contains a maximum of 127 bytes. The 802.15.4 frame format is presented below.

Frame Control	Sequence number	Addressing fields	Variable data	Frame Check Sequence
2 bytes	1 bytes	0 to 20 bytes	0 to 102 bytes	2 bytes

Again from the header, the sequence number and addressing fields are stored into the database. The device ID and timestamp information from the PC device are also stored into database.

2.2.3.3.3.3 WiFi packet sniffing*

A Typical of-the-shelf WiFi chip can be used to collect PHY and MAC layer parameters. PHY layer parameters include, but are not limited to, transmission rate, frequency, received signal strength and modulation types. MAC layer parameters also include, but not limited, to MAC address, sequence number, Frame Check Sequence and frame length.

MAC layer parameters are included in the transmitted packet, whereas PHY layer parameters are generated at the receiver end, and it depends on the driver support, which is made available to the end user. If it is supported, these PHY layer parameters are injected into the received frame and brought as one whole packet to the application.

The additional support we can get from wireless drivers is the ability to work in monitor mode. In this mode, an interface can intercept packets coming from all the nodes within a single Wireless Distribution System (WDS). More to that, a promiscuous mode interface can intercept all packets in the air including packets that are external to the WDS. This mode by default embraces the functions of a monitor mode interface and therefore it suffice to create a promiscuous mode interface in order to sniff all packets on the air.

In w.ilab.t, the WiFi card and device driver used, Sparklan WPEA-110N/E/11n mini PCI 2T2R and ath9k, support RadioTap injection and by enabling a promiscuous mode interface, we can capture all packets, extract the PHY and MAC layer parameters and store it into database.

At the data collection or database end, each record has a unique Frame Check Sequence. There is a limited chance that two consecutive packets on the air will have identical Frame Check Sequences. However the case, there are special cases where this happens. Sometimes, senders can send multiple copies of identical packets on the air or during retransmission of packets where the exact replica has to be transmitted. Certain control packets like ACK, CTS and RTS can be identical although sent at different moments in time. In all these cases, replicated packets are modified by changing their Frame Check Sequence (FCS) and that avoids identical packet post processing.

2.2.3.3.3.4 USRP/imec sensing engine*

The USRP and imec sensing engines were developped in earlier years of CREW. Readers are refferred to D6.3 from Year 3 and [15] for further details. Compared to the energy based spectrum sensing using normal WiFi interfaces, the advantage of USRP or imec sensing engine cover the following aspects:

- Seamless spectrum sensing using USRP
- Spectrum sensing out of normal WiFi channels is possible with imec sensing scaldio front end or other daughter board from USRP

- Both imec sensing engine and USRP offer more flexibility in terms of frequency resolution, while the solution using normal WiFi interface has the frequency resolution fixed to 392 KHz.

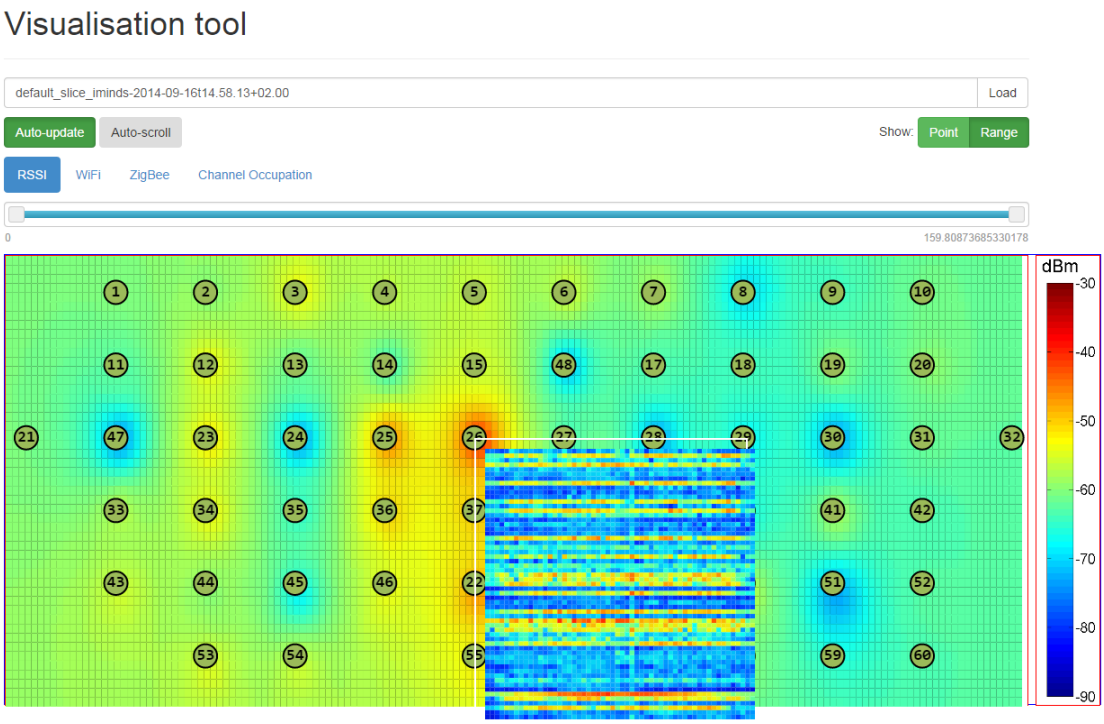
However, as there are more regular WiFi devices deployed in the testbed, the detection using normal WiFi interface can offer more accurate result across the entire testbed.

2.2.3.3.3.5 Visualisation tool*

The visualization tool can be used to visualize the captured data during or after the experiment. This allows the experimenter to visually assess the environment under which the experiment takes/took place. Depending on the enabled monitor components, several views are available.

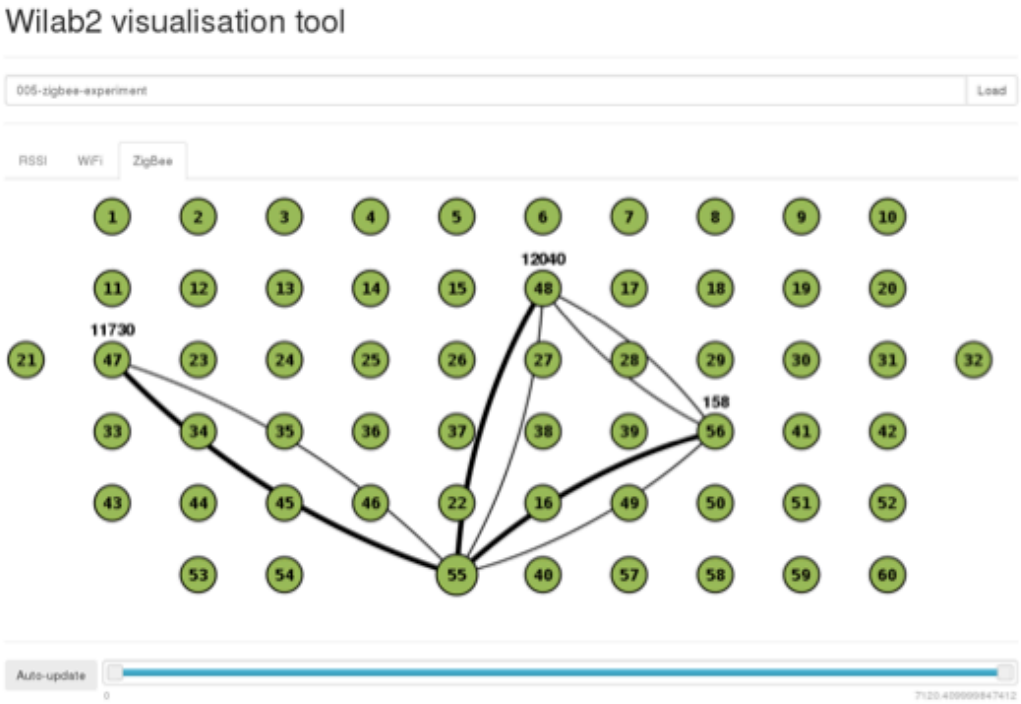
Heath map and spectrogram*

If the PHY monitor was enabled, a heatmap can be shown that interpolates the measured RSSI values between the different monitoring nodes. The more monitor nodes are enabled, the more realistic this map will be. By hovering over a monitor node the experimenter can view a spectrogram measured by that node. In addition, when user wants more information at a specific node, a spectrogram can be generated for more information within the selected band.



Link map*

Using WiFi or ZigBee link layer data, the communication links between different nodes can be visualized. By filtering out traffic produced by the experiment itself, it is possible to see which nodes could have interfered with the experiment. This visualization is technology will only show interference sources coming from WiFi and ZigBee devices.



2.3 Connectivity Brokerage Framework Extensions

During the second year of the CREW project a collaboration between CREW and UC Berkeley had been established through various bilateral discussions between CREW and Jan Rabaey from Berkeley Wireless Research Center (BWRC). The goal of this collaboration was to cooperate towards prototyping the *Connectivity Brokerage* (CB) framework [3] and applying it within the CREW infrastructure. The CB framework is a means to enable cooperation between devices in order to achieve spectrum sharing that *"enables diverse wireless technologies to exchange information and to collaborate in a seamless fashion, making a joint optimization of the scarce spectrum resources possible."* [3]. As described in D5.2, the *Connectivity Brokerage* (CB) framework allows experimenters to focus their investigations on the subset of cognitive radio approaches of interest and establishes a means for collaboration among different CR entities at different levels of abstraction (please refer to section 2.1 of deliverable D5.2 for a description of the CB framework).

The benefits for a CREW experimenter are that the CB framework describes a structured functionality set that should be supported by each entity in the framework, called Connectivity Agent (CAgent). This allows better structuring an implementation and focus on individual functionality. Furthermore, experimenters may use the CB framework to "fill up" the missing parts of their CR system-under-test (SUT) with generic CB framework software components developed in the CREW project, for example a repository that stores spectrum sensing results. In addition to providing a first implementation of the CB framework during the second year of the CREW project [3] the inter-CAgent communication via the so-called virtual control channel (VCC) and [4] the serialization of messages was specified.

In the third year, the collaboration between CREW and UC Berkeley has continued. In the following we report on the tasks that have been carried out in the third year of the CREW project, among other things, a redesign of our initial CB implementation, a performance evaluation of various database systems for their suitability to store spectrum sensing data and a new mechanism that allows CAgents to specify the type of data contained in their repositories to facilitate discovery of relevant data.

2.3.1 Revised Software Design

The first CREW implementation of the CB framework had to a large extent been used for CR experiments in the TWIST testbed. Once the implementation was applied to other platforms it became clear that it included some unwanted dependencies. For example, the Discovery class included a function and definitions tailored to a specific sensing platform (TelosB platform); also the Repository class included a reference to a specific MySQL database to store spectrum sensing data. In order to decouple the framework from these platform- and testbed-specific functions a redesign was undertaken during the third year of the CREW project.

Our revised design tries to better separate the CB framework from the scenario and involved platforms. The revised software approach is based on two levels of abstraction, a set of generic modules and derived ones. The decoupling is realized via object-oriented inheritance. The generic modules are the CB core and the derived modules are scenario-dependent as shown in the following figure.

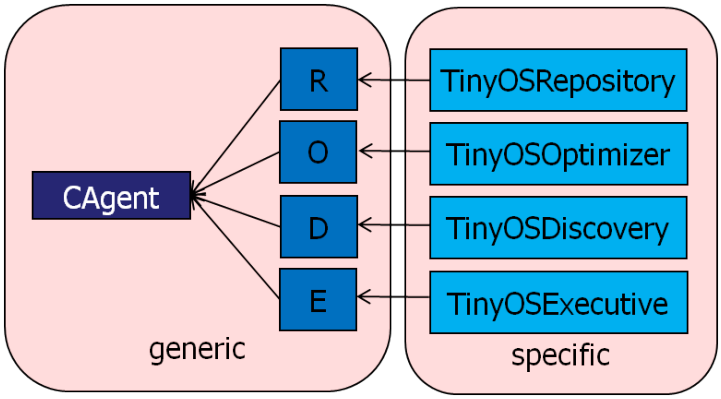


Figure 14: Revised class diagram

The generic functions are grouped in the basic classes: CAgent, Discovery, Repository, Optimizer and Executive which interact according to the following UML diagram.

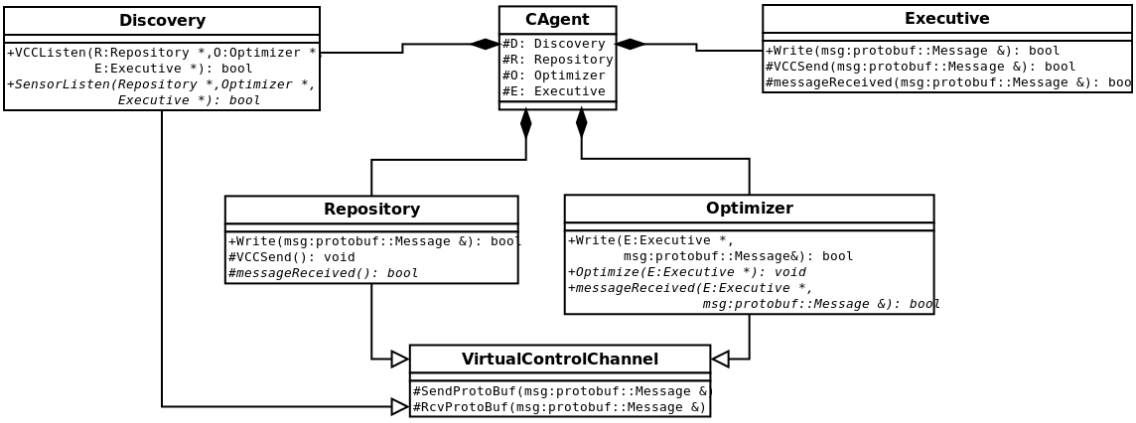


Figure 15: Revised UML diagram

The *CAgent* class is the basic block of the CB framework. It only has the reference to the four blocks that forms a CAgent now. The *Discovery* class has a similar function as a dispatcher. It has generic function in order to listen the VCC and a virtual function to listen the sensor data. This virtual function should to be implemented by the derived class, because it is technology dependent.

The *Repository* class has a method Write that receives intra-CAgent communications from other modules, and a virtual method to manage the received messages. Also, it has a method to send messages to the VCC. The new *Optimizer* class has a method that receives messages from other blocks and a virtual method to manage them. Another virtual method that has been included is Optimize(). This function should be implemented in the specific scenario because each application will need a different optimization algorithm. Like the *Optimizer* and the *Repository*, the *Executive* module has two functions to manage the received messages. Also, it provides a function for sending different kinds of messages over the VCC.

In an instantiation of this generic implementation the specific blocks are derived from the base classes. The four blocks have a derived implementation that inherit the generic methods and implement other specific functions. For example, derived classes for the TelosB platform have been developed as shown in the following figure.

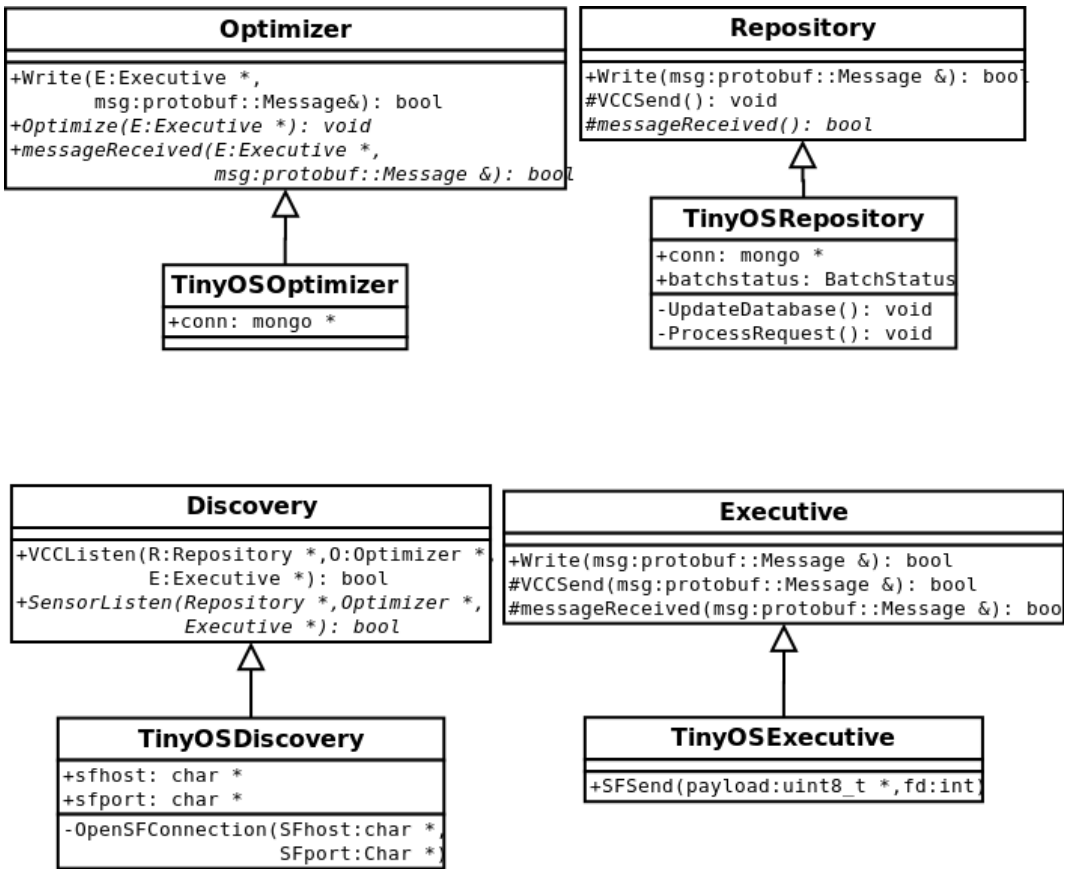


Figure 16: Revised part class diagram for TelosB platform

The *TinyOSDiscovery* class includes the data, pointers and functions to receive the spectrum sensing data from the TelosB platform over a serial port. The *TinyOSRepository* class has a function to update the database when it receives new data. For example, the revised MongoDB database introduced below can be pointed to from this class. Finally, *TinyOSRepository* has a function to manage the request message sent from others blocks or CAgents.

The *TinyOSOptimizer* class has a pointer to the database which it may need to access while executing the optimize algorithm. Finally, the *TinyOSExecutive* class maintains the function *SFSend* (c.f. previous implementation described in D5.2) that enables the communication with the TelosB nodes.

The last part of the revised design facilitates communication among CAgents. The spectrum sensing data and metadata (described below) are going to be queried by other CAgents that have no direct access to the database. Therefore, it is important that the framework supports CAgent to Cagent

communication, independently of their types. This communication is abstracted by the VCC. Now, a Broker class is inserted into the framework to improve the possibilities of the VCC: while in our previous design N (NodeCAgents) to 1 (TestbedCAgent) communication was supported, the added broker component now supports N to N communication.

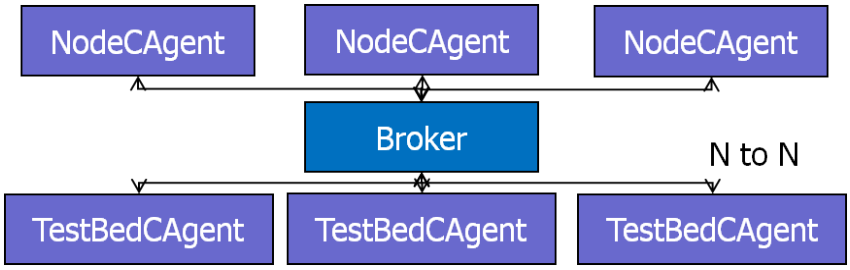


Figure 17: Broker design and CAgent communication.

The task of the *Broker* is to forward the message from the NodeCAgents to all TestbedCAgents, and in the opposite direction.

2.3.2 Comparison of Databases

Within the Cognitive Brokerage framework the task of the *Repository* module is to store the environment information collected by the sensors. It is expected that there may be a massive amount of such spectrum sensing information that needs to be stored. In our previous design (c.f. deliverable D5.2) a MySQL database was selected to store spectrum sensing information. The choice was motivated by the popularity of the database rather than its performance or suitability for our particular use cases. In the third year of the CREW project we revisited the choice of the database within the Repository component more thoroughly. We conducted a study to understand the suitability of four popular databases for our scenarios: MySQL, PostgreSQL, SQLite and MongoDB. The selection of these four databases is based on the fact that all of them are quite popular and easily available. Three of them use a compatible language, SQL. MongoDB is an NoSQL database that has been selected to test a different database approach.

One of the requirements for the database is to be accessible from everywhere and everyone. In the following we compare the local and remote access. Then, the most important programming languages used to access to the database are described.

MySQL: The database host should be installed on a mysql server. This is the only requirement in order to access the data locally or remotely. The access function is the same and supports any IP address. MySQL supports multiple transactions at the same time, multithreading and multiuser. Each user can have different privileges for different databases. The most important programming languages in order to access MySQL databases are: C, C++, C#, Pascal, Delphi (via dbExpress), Java, Lisp, Perl, PHP, Python, Ruby or Tcl.

PostgresQSL: As in MySQL, the only requirement is to install a postgreSQL server on the host. The remote and local access works in the same way. PostgreSQL supports multiple transactions at the same time, multiproccess and multiuser. Each user can have different privileges for different databases. The most important programming languages in order to access postgreSQL database are: C, C++, Java, PHP, Tcl, Python, Ruby or Perl.

SQLite: SQLite database store the information in a simple file, that is blocked for each transaction. This characteristic makes it unsuitable for remote access. However, it can be achieved via NFS or Samba. SQLite does not support multiuser or multiaccess, only multiple reading access. This is because of the database file based architecture. Moreover the SQLite database does not have users and passwords. The most important programming languages in order to access postgreSQL database are:

C, C++, Java, PHP, Tcl or Python. The three databases selected in this benchmark are multi- platform and can be installed in Windows, Linux and OS X.

MongoDB: Instead of storing data in tables as is done in a "classical" relational database, MongoDB stores structured data as JSON-like documents, using dynamic schemas (called BSON), rather than predefined schemas. An element of data is called a document, and documents are stored in collections. One collection may have any number of documents. Compared to relational databases, we could say collections are like tables, and documents are like records. MongoDB supports local and remote connections. The installation on the host is simple. MongoDB's design philosophy doesn't care much about users and passwords but it supports them. The most important programming languages in order to access MongoDB databases are: C, C++, Java, Perl, PHP, Ruby or Python.

2.3.2.1 Buffering and Caching Concepts

Multiple agents will access the database when the system will grow. Therefore, it is important to reduce the time that the agents spend reading or writing data (concurrency). We have studied how the three databases work when the agent wants to introduce or read multiple data and if there are some buffering libraries in order to simplify this mechanism.

MySQL uses InnoDB as an ACID (Atomicity, Consistency, Isolation, Durability) compliant, transactional storage engine using Multiversion concurrency control (MVCC) technology. The InnoDB engine has an insert buffer that caches updates to secondary index entries and applies them in the background. This can significantly speed up inserts, reducing the number of physical writes required by combining many updates. If a secondary index page has outstanding updates when it is needed for a query the updates will be merged first. As of version 5.5 the insert buffer is also used as a buffer for other types of writes, improving the performance of UPDATE queries as well. Prepared statements are a method to accomplish the task of executing a query repeatedly, albeit with different parameters in each iteration. C and C++ APIs allow the use of prepare statements which improve the security of the MySQL server.

PostgreSQL supports a data storage motor, its default Postgres storage system (Postgres Storage System). It uses caching in the same way as MySQL does. The modified and written data is moved to cache in order to speed up the following queries. A C API for PostgreSQL implements Prepared Statements in the same way as MySQL. This allows to increase the security and the usability when the system executes queries repeatedly.

SQLite uses caching in order to speed up the SELECT queries. If a query only involves cached pages, it can execute much faster since no disk access is required. As in the previous two systems, a SQLite C API implements Prepared Statements. That system, described in the previous sections, provides the opportunity to insert or read multiple data sending in a more flexible and secure way. Despite implementation of prepared statements, the speed in the inserting queries is not improved. Only the security and the usability of the system are improved.

The MongoDB approach is completely different to that of SQL databases. Each document that is inserted in MongoDB could have a different structure; so, prepare statements make no sense here. MongoDB keeps all of the most recently used data in RAM. If you have created indices for your queries and your working data set fits in RAM, MongoDB serves all queries from memory.

2.3.2.2 Performance Tests

There are several benchmarks and experiments that compare between MySQL, PostgreSQL and SQLite, but the most reliable experiment is to reproduce the most common interactions between the CB system and the three kinds of database. The most common queries that will be executed in our scenarios are:

- INSERT: the Repository saves data from the sensors in the database. This query is used most often.
- SELECT by TimeStamp: some agent wants the data from a certain past time interval.
- SELECT by NodeID: the agent is interested only in a specific node.
- SELECT by Threshold: The agent only gets the data that satisfies a threshold.

In the following we report on the results of several experiments, one per each possible query and three in order to test the behavior with threads and multi-access. The tests try to mimic the usage patterns that we envision in the CREW scenarios, i.e. we utilize actual spectrum sensing data structures. The experiments have been prepared to have the same characteristics, including a database with the same data, the same host, the same host load and the same programming language, C. The database is composed of 100 rows with 19 columns in the case of SQL and 100 entries in MongoDB. Each column has an integer data. The node id and the timestamp columns are filled with controlled values, meanwhile the last 16 columns have random values between 0 and 80. Each experiment has been repeated 100 times. The experiments include the connection and the close time in each iteration.

2.3.2.2.1 Test 1: Insert

This experiment shows how a client can introduce different numbers of rows with different numbers of queries. This test measures the speed of inserting data and the efficiency of including multiple data in each query. The results are shown in the following table and visualized in the figure thereafter.

	100 rows 100 insert [us]	100 rows 1 insert [us]	1000 rows 1 insert [us]	10000 rows 1 insert [us]
mysql	840,3147	561,9536	104,44588	51,385199
sqlite	132942,772	1666,5772		
postgresql	730,7761	763,5081	123,32322	40,75623
mongodb	236,0683	48,8913	17,96169	16,304648

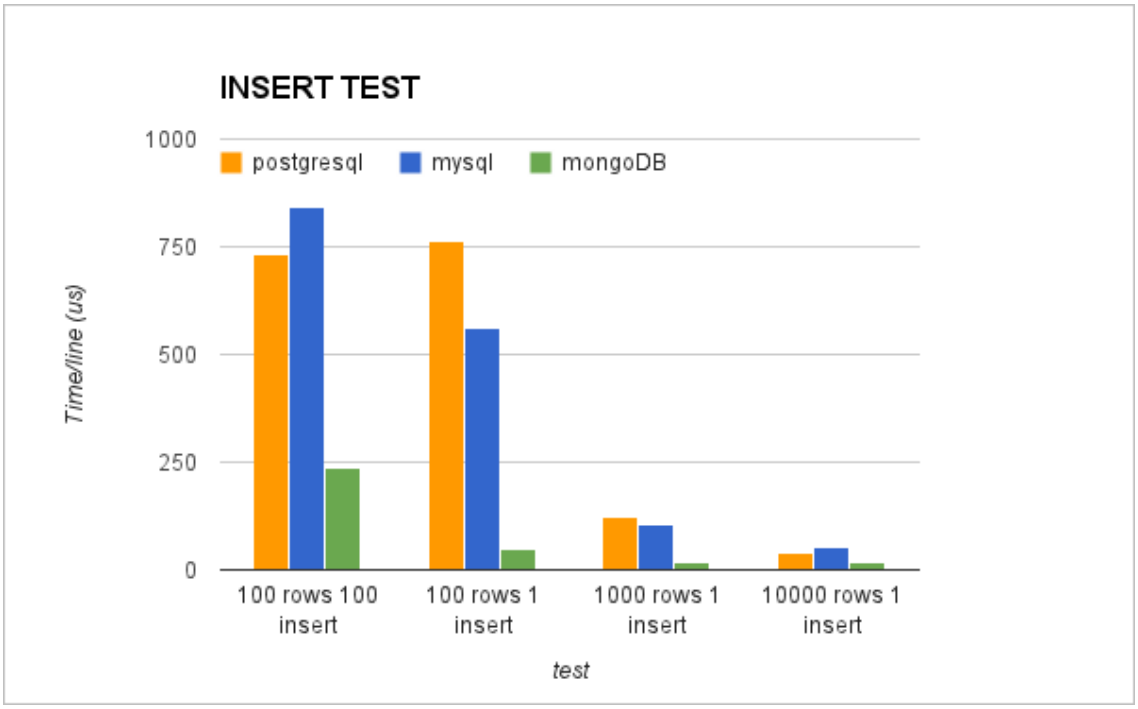


Figure 18: Insert test results. Comparison among PostgreSQL, MySQL and MongoDB

As we can see in the table, the introduction of multiple data in an INSERT sentence reduces the time for row by three when more than 1000 rows are included at the same time. However this requires more memory and a higher refresh latency. For example, if we have 100 sensors that each send data once a second, we only refresh the database once every 10 seconds. Depending on the application, the maximum buffer size should be configured in order to balance latency with the speed of insertion.

The second conclusion that can be extracted is that SQLite performs poorly compared with the rest of the databases. The performance of MySQL and PostgreSQL is quite similar and the best one is MongoDB. Inserting a batch of 100 objects in MongoDB is 10 times faster than inserting a query of 100 objects in MySQL. Three additional important factors that have been found during this test are:

1. PostgreSQL with the buffer of 10000 lines consumes lot of CPU and results in failures in the execution.
2. SQLite with the buffer of 1000 and 10000 lines results in errors. SQLite INSERT queries do not support natively more than 1 row per INSERT but it can be done using others commands, but only to 100 rows.
3. The use of prepare statements has been completely discarded because of the execution time. The insertion of 100 rows in 100 INSERT queries using prepare statements takes 41,6ms per row instead of 840us using normal string queries. This indicates that the problem with the insertion speed was caused by the prepare statements.

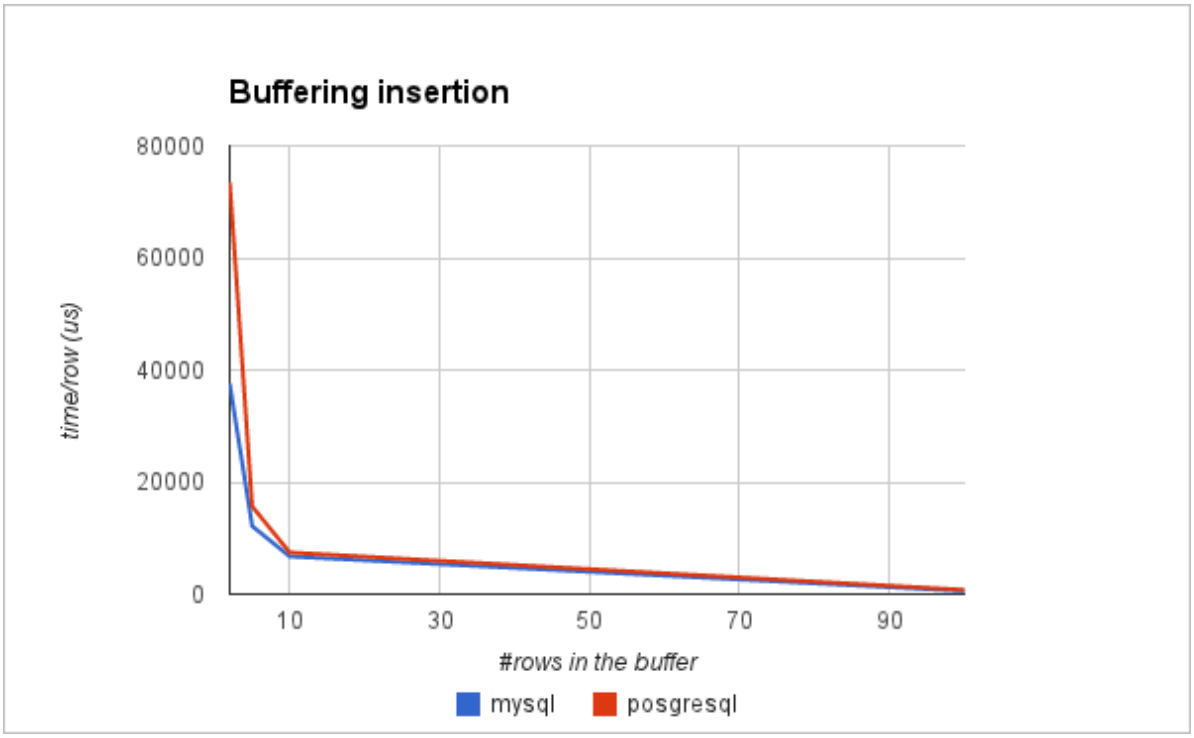


Figure 19: Buffering insertion. Comparison between PostgreSQL and MySQL

The previous figure shows the time that a client needs to insert a row when multiple rows are combined in one insert. The y axis represents the time per row in microseconds. As can be seen, with only with 5 or 10 rows in the same query the time is reduced by more than a 75%.

2.3.2.2.2 Test 2-4: Select

In order to test the performance of reading data from the database we performed the following tests:

Test 2: In this test, the clients request the data that satisfies a query of the type "Time > Threshold, i.e. reading the inserted data from an specific time. In our case the query should return 10 rows. The test consists of executing the SELECT query and accessing the data of each row. The code saves each data in a int variable. This test has been design in this way in order to compare the speed of SELECT queries but also the speed of access to the data returned.

Test 3: Select by time and node ID. The third test executes a SELECT query with two different conditions: a threshold time and a specific node ID. In this case, the database must return 2 lines. As in test 2, the application accesses the data of each row.

Test 4: Select by threshold. The difference between this example and the previous one is that the clients request data over a threshold but these data are random. This forces the database to make a more complex search, ignoring possible keys.

The results for these tests are summarized in the following table / figure.

	Test 2 [us]	Test 3 [us]	Test 4 [us]
mySQL	455,39 (148,84)	355,97 (121,96)	666,13 (293,49)
postgreSQL	70765,9 (2516,62)	65402,18 (1568,05)	66973,1 (5104,49)

SQLite	388,95 (299,01)	375,8 (273,89)	638,41 (631,80)
mongoDB	1017,36 (513,22)	1020,19 (507,89)	1135,3 (656,54)

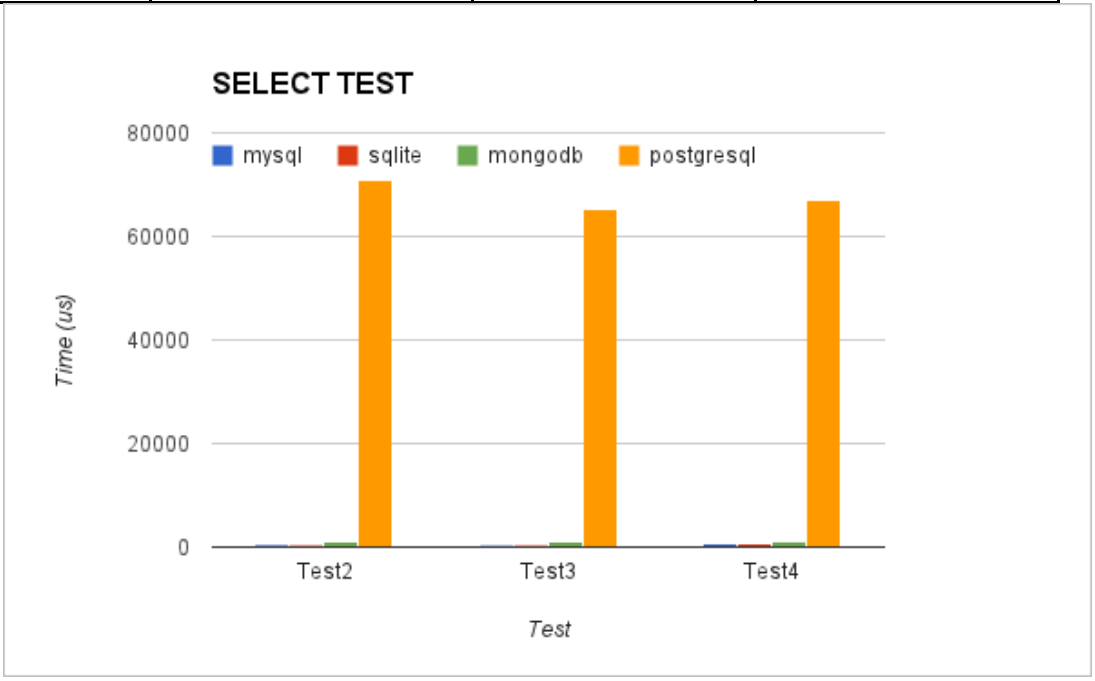


Figure 20: Select tests results. Comparison among PostgreSQL, MySQL, SQLite and MongoDB

The first figure represents the three performance results for MySQL, SQLite, PostgreSQL and MongoDB. The y axis represents the time in us to execute the SELECT sentence and to access each element of data.

The performance of PostgreSQL in all the tests is significantly worse than the performance of MySQL, SQLite or MongoDB. The reason is the overhead cost required to connect to the database. It is important to remember that each experiment includes the connection and the disconnection time. If this time is not included, the performance of PostgreSQL improves (results shown in previous table between brackets), but it still is worse than the other databases. The following figure allows to better understand the difference between the other three databases:

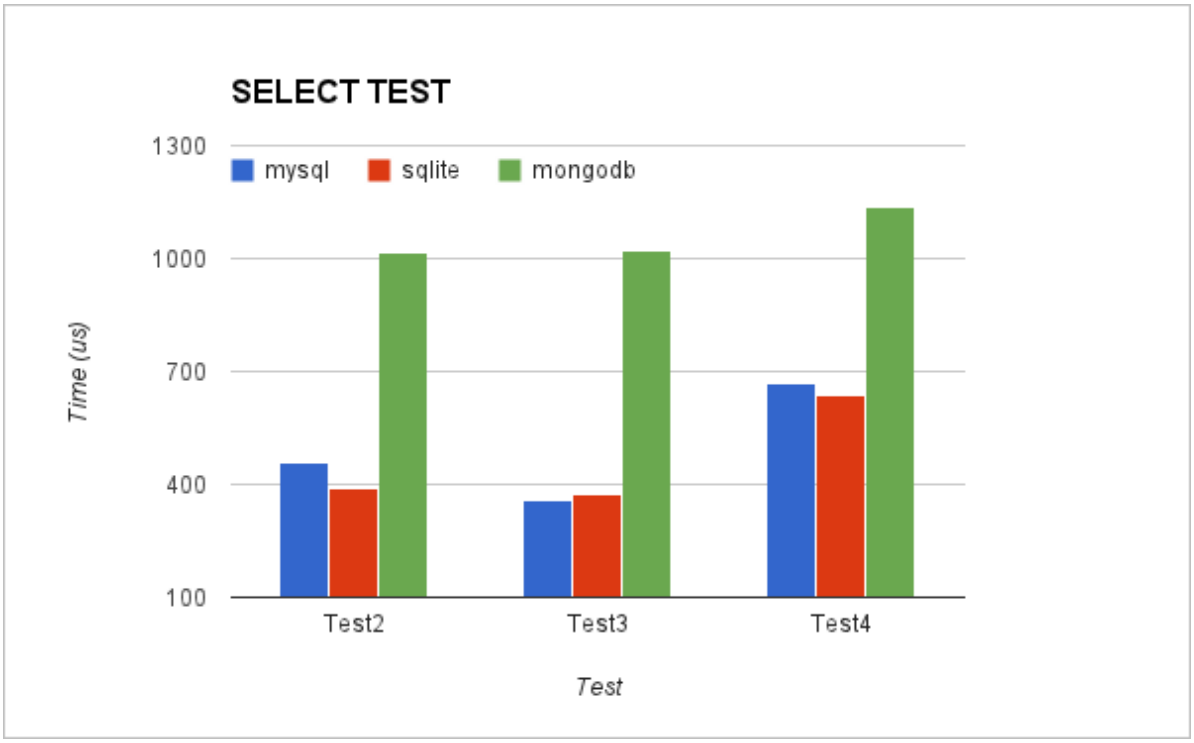


Figure 21: Select tests results. Comparison among MySQL, SQLite and MongoDB

The performance for MySQL and SQLite in all the tests is very similar. If we compare the results without including the connection time, MySQL is twice as fast as SQLite. MongoDB has good performance too, but it still needs double the time to consume data.

2.3.2.2.3 Test 5: Insert - concurrent access

These tests have been developed in order to test the performance of the database under multiple access. Each case creates different number of threads that inserts rows in the database:

- Case 1 - 10 threads and 100 rows each thread
- Case 2 - 100 threads and 100 rows each thread
- Case 3 - 100 threads and 1000 rows each thread
- Case 4 - 1000 threads and 1000 rows each thread

In the following figure the four thread-based test results are shown. The y axis represents the average time that a thread needs to insert the corresponding number of rows.

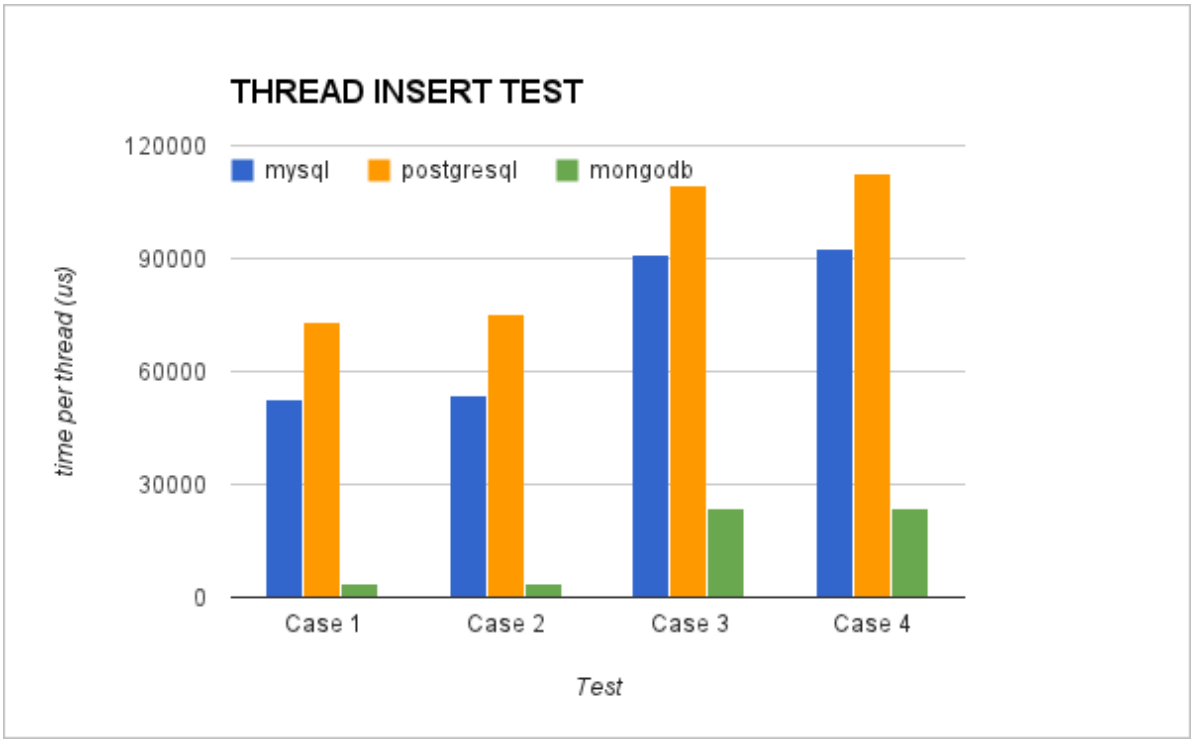


Figure 22: Insert tests with threads results. Comparison among PostgreSQL, MySQL and MongoDB

The results show that SQLite has the worst performance. They are excluded from Figure 22, as they would prevent comparison of the other solutions. Due to this reason it will also be excluded from the further investigations. The reason is again that SQLite blocks the file where the database is stored each time that a thread writes. MongoDB is the fastest database in the tests but MySQL and PostgreSQL have a good performance, too. This test shows that the number of objects in the batch affects the time more than the number of threads.

2.3.2.2.4 Test 6: Select - concurrent access

The test 6 has the same goal as the previous test, but in this case the threads are consumers. Each thread sends a query to the database in order to get the last 10 data elements saved. The results are summarized in the following figure.

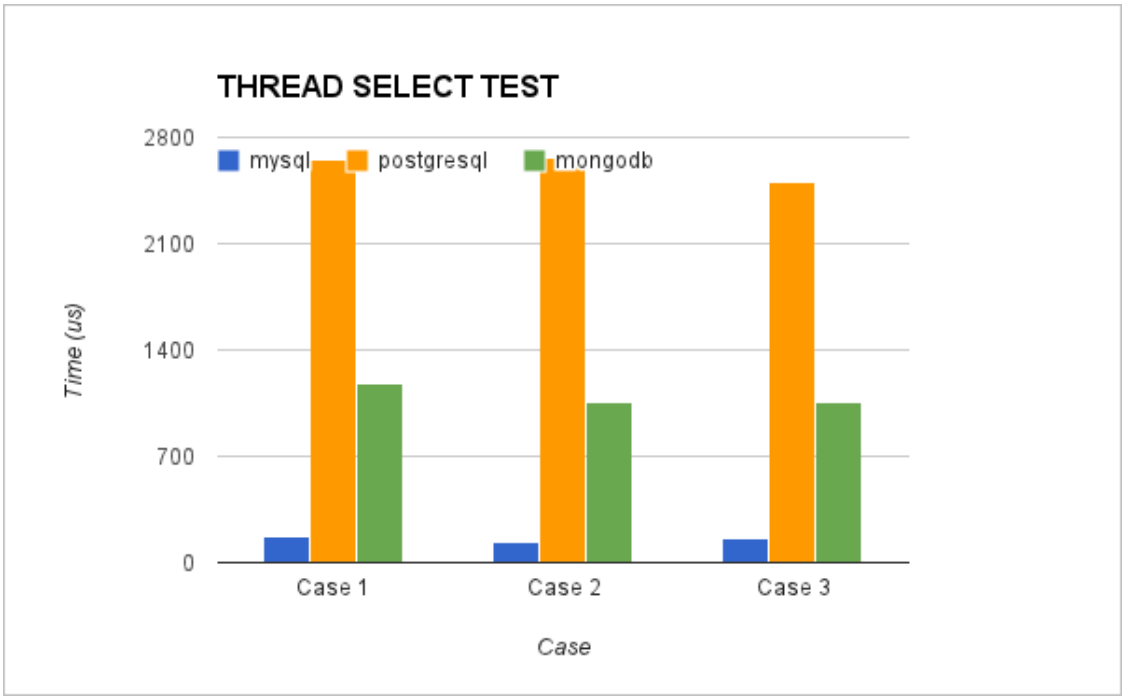


Figure 23: Select tests with threads results. Comparison among PostgreSQL, MySQL and MongoDB

MySQL is again the quickest when the application executes SELECT queries. MongoDB takes more time, but it still obtains reasonable results. Finally, PostgreSQL takes more time than the other databases.

2.3.2.2.5 Test 7: Insert/Select - concurrent access

The last test tries to generate a common situation in the database, where multiple connections (producers and consumers) access the database at the same time. The test creates 10 producers and 10 consumers. Each producer connects to the database 100 times and each time saves 100 rows or objects. The consumers connect to the database 100 times and request the last 10 rows or objects.

The test is executed in two variants ("cases"). The first case creates all the producers first and then the consumers. The second case creates alternatively (interleaved) a producer and then a consumer, etc. The results are shown in the following figure.

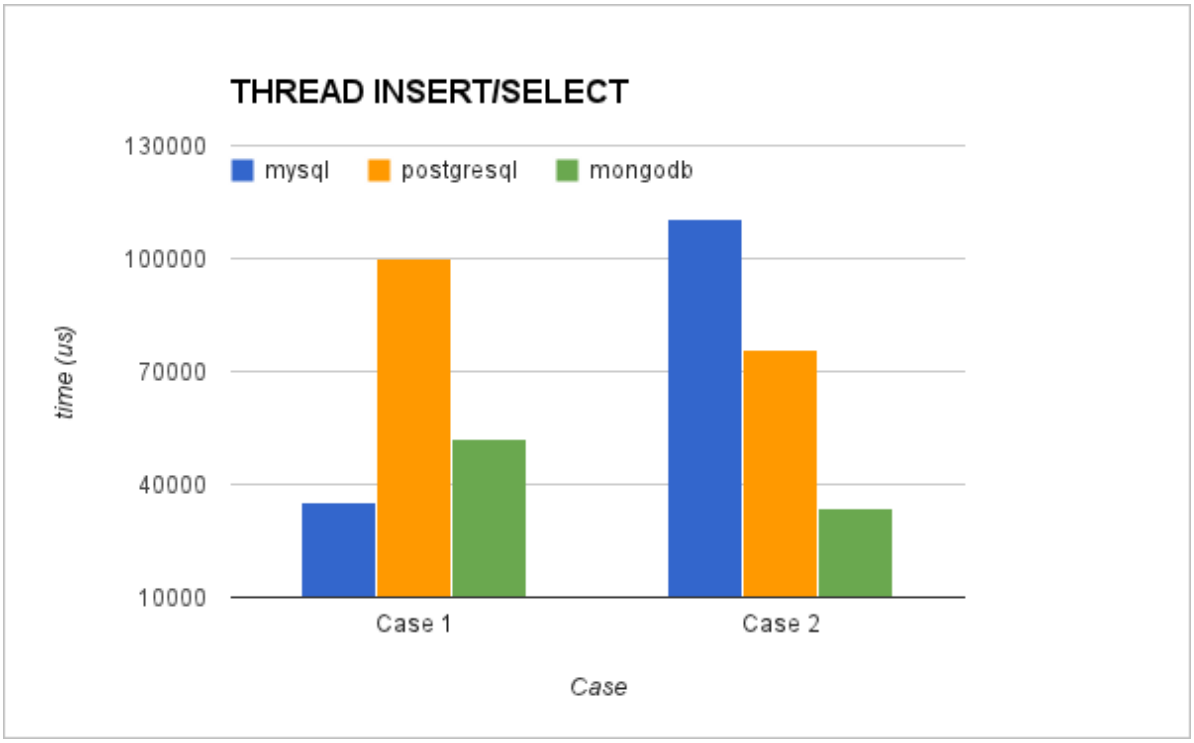


Figure 24: Multiple thread tests results. Comparison among PostgreSQL, MySQL and MongoDB

The results show that MySQL is the best option when producers and the consumers access at different times. However, when the producers and consumers operate interleaved MongoDB has the best performance.

2.3.2.2.6 Usability

The use of a C library and the installation of the three SQL technologies was simple, efficient and intuitive. The installation and the use of MongoDB are also simple, but this database uses a completely different usage model. If the user is used to SQL languages, the first steps may be slower. Our usability experiences are summarized in the following table.

	Strengths	Weaknesses
MySQL	Simple fastest SELECT queries Most used Well known by developers	Simultaneous producers and consumers few data types supported
PostgreSQL	Scalability Complex features	Speed CPU usage with high buffers
MongoDB	Speed INSERT queries Flexibility	Less used Not well known

2.3.2.2.7 Conclusions from the Performance Tests

We have presented the investigation and a benchmark among three well known databases. The test has been modeled according to the requirements of the CB system in CREW scenarios and is biased to those requirements. Other benchmarks try to assess the performance of the databases in a general and usage independent way. We have showed a pretty specific type of use where the multiple access to the newest data in the database is very important. According to the results the use of SQLite should be discarded because of many restrictions: it does not support users and passwords, does not support remote connections and does not support multiple insertions in a single query.

MySQL and PostgreSQL have a lot of features in common. The performance in many tests has also been similar, but there are a few differences in the SELECT tests, where MySQL shows better results. MongoDB often features superior results in comparison with MySQL and PostgreSQL, but the selection of "best database" is test-specific.

According to our results we can determine the scenarios where the use of each database is indicated:

1. **Scenario with more producers than consumers**. In this scenario the use of **MongoDB** is recommended, because MongoDB shows a very good performance when inserting data.
2. **Scenario with more consumers than producers**. Here, **MySQL** is the best solution. MySQL SELECT queries have had the best results and the SQL language is easier than MongoDB in order to build SELECT queries.
3. **Complex scenario**. When the scenario requires high requirements, a high number of data and complex features there is some indications that **PostgreSQL** *may* be the optimum solution. However, we have not been able to run very large-scale tests which show the threshold when PostgreSQL starts to be more efficient than MongoDB or MySQL.

Since in the CREW scenarios we envision many data producers (spectrum sensing devices) in our implementation we have opted for MongoDB. This database is scalable, high-performance, open source and NoSQL. MongoDB uses a database and collections in order to save the entries. The collection can be compared with the tables in MySQL. The basic elements of information in MongoDB are the BSON objects. The BSON objects are basically binary JSON objects. Each BSON object includes some key-value pairs, similar to a row in MySQL. The next code represents an example of BSON object showing an entry of spectrum sensing data collected with the TelosB platform:

```
{ "_id" : ObjectId("50ebf4b245c03d4f2f12a230"), "node_id" : 1, "TimeStampSec" : 1357640882, "TimeStampMicroSec" : 297237, "ch11" : -93, "ch12" : -92, "ch13" : -86, "ch14" : -93, "ch15" : -92, "ch16" : -93, "ch17" : -92, "ch18" : -93, "ch19" : -94, "ch20" : -94, "ch21" : -82, "ch22" : -93, "ch23" : -73, "ch24" : -88, "ch25" : -70, "ch26" : -93, "capabilitiesid" : 1, "profileid" : 1 }
```

In order to migrate our previous database implementation from MySQL to MongoDB, multiple functions had been implemented. Some are similar, for example, the function retrieve_rssi_by_timestamp(), which retrieves the data that fills a time restriction. In addition, other functions have been developed in order to support the metadata included in the database. The metadata are explained in the next section.

2.3.3 Metadata and Discovery Functions

Our first implementation of the CB framework lacked some important information and functions: the CAgent saved the RSSI data in a table or collection but for external clients it was not possible to query / discover the structure (data types) in another CAgent's repository. If an external CAgent wanted information from a node it would have to know the structure of the database a priori. In the third year

of the CREW project we solved this problem by introducing a well-defined set of metadata and API that CAgents can use to discover each other's repository content.

The new data and functions have been selected from the IEEE 1900.6 standard that specifies procedures, protocols and message format specifications for the exchange of sensing related data, control data and configuration data between spectrum sensors and their clients. The standard is very extensive and only two primitives have been adopted. However, these two primitives include most of the relevant information for a spectrum sensing application. The two primitives define the API to access the metadata. Moreover, IEEE 1900.6 also defines the data types of the information included in the metadata.

The first primitive is used to request information about the spectrum measurement capabilities. These capabilities are static and associated with a platform. This primitive returns information about the frequency range, the sensing mode, frequency resolution, sweep time, etc. The first primitive includes the following request() and response() function pair:

Get_Supported_Spectrum_Measurement_Description.request

(MeasurementCapabilityTransID)

Get_Supported_Spectrum_Measurement_Description.response

(MeasurementCapabilityTransID, Status, MeasuRange, SensingMode, DataSheet.ADDAResolution, DataSheet.AngleResolution, DataSheet.FrequencyResolution, DataSheet.LocationTimeCapability, DataSheet.LoggingFunctions, DataSheet.RecordingCapability, DataSheet.SweepTime, LockStatus)

The second primitive is related to the configuration of the platform. This type of configuration is variable, it is specific for a particular experiment but may change between different experiments. The primitive retrieves information like the confidence level, sensing mode, report rate, time stamps, lower threshold, measure bandwidth, etc. The second primitive includes the following request() and response() function pair:

Get_Sensor_Measurement_Profile.request

(SensorMeasurementProfileID)

Get_Sensor_Measurement_Profile.response

(SensorMeasurementProfileID, Status, ConfidenceLevel, ReportMode, SensingMode, ReportRate, TimeStamp, Scan.LowerThreshold, MeasuBandwidth, LockStatus)

For both primitives we have implemented MongoDB functions and respective protobuffers that support the request (recall that we use Google protocol buffers for serialization). The protobuffers are filled with the respective information. The google protobuffers designed for the primitives are used as in the following example:

```
message CAgentResMessage {
  optional int32 cagentsrcid = 1;
  optional int32 cagentdestid = 2;
  optional CAgentResType type = 3;
  enum CAgentResType {
```

```
  MEASUREMENT_CAPABILITIES = 0;
  MEASUREMENT_PROFILE = 1;
 }
 optional MeasurementCapabilities mcapabilities = 4;
 optional MeasurementProfile mprofile = 5;
}
```

The next message sequence diagram visualizes a common operation of a new CAgent when it wants to filter the information, i.e. a CAgent selecting the data from some platform or configuration.

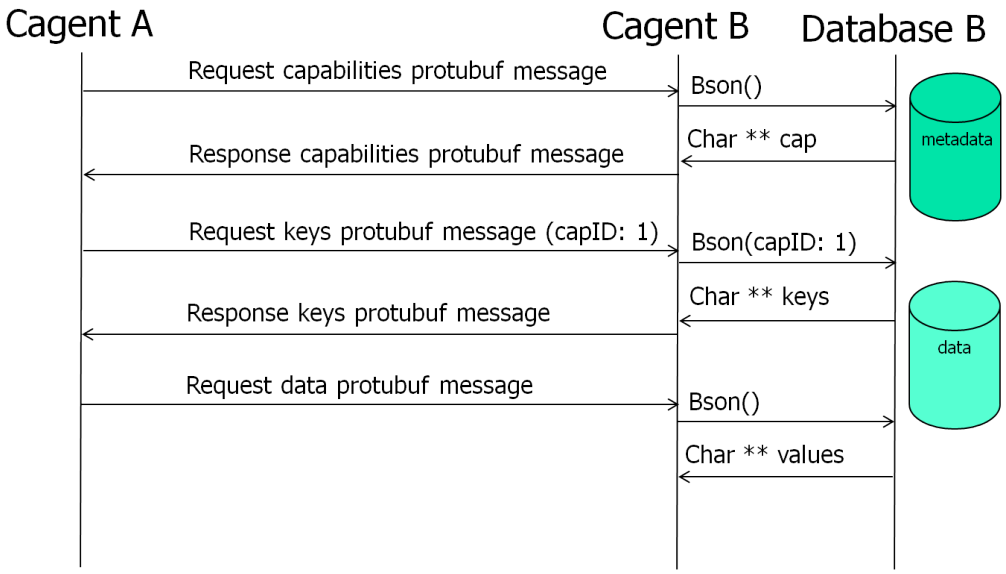


Figure 25: Example of the metadata access method.

The CAgent requests the available capabilities in the metadata. It first receives a list of different capability IDs and their properties. With this information the CAgent can find out which parameters the platform has and what the capabilities of that CAgent are. Afterwards the CAgent has all the information necessary to select the data it wants in a subsequent query (not shown).

2.3.4 Summary

In the third year of the project the collaboration with the Berkeley Wireless Research Center (BWRC) on prototyping the Connectivity Brokerage framework has continued. The main outcome is summarized in the following:

- We redesigned our software framework: the CB framework implementation is now decoupled from the scenario and can thus be instantiated more easily in different experiments.
- We have performed a detailed performance analysis of various databases to better understand their suitability for a spectrum sensing repository and extracted guidelines that help in selecting a database for a specific CR scenario.
- For our internal use case MongoDB has been selected and integrated with our software framework.

- We have adopted parts of the IEEE 1900.6 standard to extend the CB framework with metadata and provide an API that allows discovery of repository content among CAgents.

2.3.5 Hardware specific implementations

2.3.5.1 CB implementation on w-iLab.t

We further extend the CB implementation in a simplified manner for different sensing platforms, more specifically the USRP and IMEC sensing engine. Those extensions are tested on the w-iLab.t testbed. Within the context of CB, the extension focuses on the virtual control channel functionality, by utilizing Google protocol buffer and ZMQ libraries. Two types of CAgents are implemented: the USRPCAgent and the IMECCAgent. Furthermore, dedicated applications on an individual NodeCAgent are developed to query sensing information from the USRPCAgent and the IMECCAgent respectively. The USRPCAgent is directly coupled to the USRP hardware and the same goes for IMECCAgent., which produces sensing data. NodeCAgent has no direct link to the sensing hardware, it queries sensing data from the corresponding sensing engine CAgent via the virtual control channel.

2.3.5.1.1 Define the message structure

Two types of messages are defined on the virtual control channel, one contains the configuration of the sensing engine and one contains the measurement result. We defined those two types via the Google protocol buffer. First, a very concise ".proto" file is defined, which has exactly what item the message needs to contain. Secondly, based on the ".proto" file, Google protocol buffer generates more comprehensive ".cc" and ".hp" file. When opening the generated files, we see that every message we defined in the ".proto" file is a corresponding class in the ".cc" file, which not only contains all the message fields, but also the functions to prepare the message and interpret the message. The relevant part in the ".proto" file of the IMEC sensing engine and USRP are shown in Figure 26 and Figure 27 respectively. Both sensing engines have the "se_mode" in its configuration message. This field is used to specify the type of channel the sensing engine should be configured to measure, for example to choose between WiFi channel and Zigbee channel. A couple of fields ("first_channel", "last_channel") are used to specify the request channel range. And a couple of fields, like the "fe_gain", "bandwidth", are used to configure the radio front-end. Apart from the comment fields, USRP sensing engine has the "DetectMode" field, to choose between maxhold, averaging or minhold; IMEC sensing engine has two output formats, you can choose to read the raw output or the output converted to dBm.

```
/*-------------------------------------------------------------------------*/

message Configuration {
  // Keep values in sync with the enumeration 'se_mode_t' defined in 'sensing.h'
  enum SeMode{
    FFT_SWEEP          = 0;
    WLAN_G             = 1;
    WLAN_A             = 2;
    BLUETOOTH          = 3;
    ZIGBEE             = 4;
    LTE                = 5;
    DVB_T              = 6;
    ISM_POWER_DETECT   = 7;
    TRANSMIT           = 97;
    ADC_LOG1           = 98;
    ADC_LOG2           = 99;
    STDBY              = 100;
  }
  enum OutputFormat {
        RAW = 0;
        DBM = 1;
  }
  required SeMode      se_mode              =   1;
  optional uint32      fe_gain              =   2;
  optional uint32      first_channel        =   3;
  optional uint32      last_channel         =   4;
  optional uint32      bandwidth            =   5;
  optional uint32      fft_points           =   6;
  optional uint32      dvb_nr_carriers      =   7;
  optional float       dvb_guard_interval   =   8;
  optional float       threshold_power      =   9 [default = 10.5];
  optional OutputFormat output_format       =   10;
}
```

Figure 26 The configuration message for IMEC sensing engine

```
message Configuration {
  enum SeMode{
//    FFT_SWEEP          = 0;
    WLAN_G             = 1;
    WLAN_A             = 2;
    BLUETOOTH          = 3;
    ZIGBEE             = 4;

  }

  enum DetectMode{
        AVERAGE = 1;
        MAXHOLD = 2;
        MINHOLD = 3;
  }
  required SeMode      se_mode              =   1;
  optional uint32      fe_gain              =   2;
  optional uint32      first_channel        =   3;
  optional uint32      last_channel         =   4;
  optional uint32      fft_points           =   5;
  optional string      usrpip               =   6;
  optional uint32      spb                  =   7;
  required DetectMode  det_mode                      =   8;
}
```

Figure 27 The configuration message for USRP

2.3.5.1.2 TestbedCAgent

When looking at the different roles of the two CAgents, it is clear that the TestbedCAgent needs to listen to the request from the NodeCAgents. Therefore the TestbedCAgent is implemented as a server, which contains a loop to wait and process incoming requests. The detailed structure of TestbedCAgent

is illustrated in Figure 28. Before entering this loop, there are a few things to set up. Upon starting up, the IMECCAgent establishes a connection with its sensing hardware. In case of the IMEC sensing engine, this means we first configure the the SPIDER platform (loading the firmware for the USB interface chip and configuring the FPGA on the device) and then load the DIFFS chip with its firmware. In case of USRP, the server tries to establish connection with USRP and allocate buffers for receiving samples. Apart from the device-specific hardware initialization, the CAgent also initialized the ZMQ libraries to ensure that the socket is successfully created and binded. For Google protocol buffer we need to do a version control to make sure that the binary version used to generate the ".cc" and ".h" files are exactly the same as the library version we are using for serializing and de-serializing the message. Finally a stop handler is registered. In case of emergency shutdown, the stop handler makes sure that the sensing engine hardware is shut down properly as well.

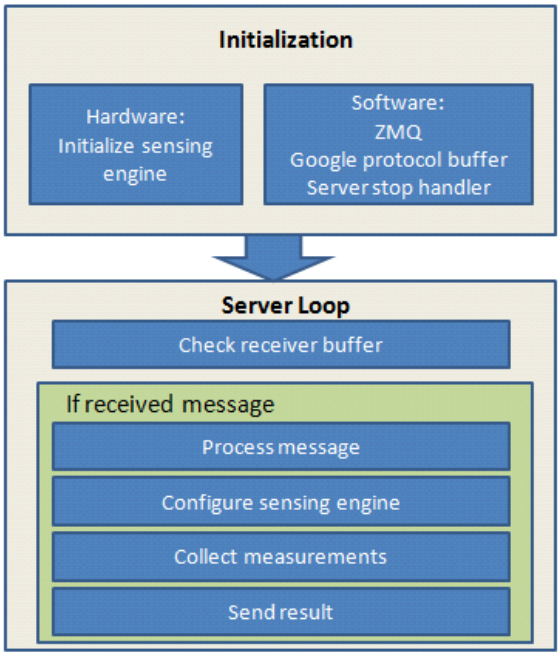


Figure 28 TestbedCAgent diagram

After the initialization phase, the program enters a loop. It keeps on polling the receiving buffer until a message is present. Then it will parse the message to understand what kind of measurement is required and configure the sensing engine to perform measurements. Finally the collected result is sent back to the client.

2.3.5.1.3 The sensing query applications on the NodeCAgent

The application to query sensing information on a regular NodeCAgent is less complex than the sensing engine CAgent. It has a couple of input options used to describe the configurations to be passed tothe sensing engine. The most important option is the "-server", to specify the address and port of the sensing engine CAgent. The rest of the options are used to configure the sensing engine, for instance, it can choose the "se_mode" among different wireless standards; it can fine tune the sensing engine's sensitivity via the "fe_gain" option, etc. As an example, the help menu of the NodeCAgent application to query from the USRP sensing engine is shown in Figure 29. The message structure used in the NodeCAgent is exactly the same as in the sensing engine CAgent, see the previous section for more information.

```
liu@craig:~/Documents/workspace/usrpcb/build$ ./usrpse_cb_client -help
linux; GNU C++ version 4.4.3; Boost_104000; UHD_003.005.003-78-g49a4929b

Usage:

        $ ./usrpse_cb_client [OPTIONS]

        Options are:
                -help
                        Print this help page.
                -level LEVEL
                        Print messages of specified and lower level.
                          Level 0: No messages, except messages from the UHD library.
                          Level 1: Panic/Error messages.
                          Level 2: Warning messages.
                          Level 3: Information messages (default).
                          Level 4: Logging messages.
                          Level 5: Verbose messages.
                -silent
                        Disable printing messages.
                -server
                        Server address of the sensing engine server.
                          ZeroMQ format: protocol://host:port
                          Default:       tcp://se-server:3607
                -output
                        Output filename storing the measurement data.
                          Default: ./data.out

                -mode MODE
                        Sensing mode. Supported modes with arguments are:
                          WLAN_G    [fe_gain,first_channel,last_channel]
                          WLAN_A    [fe_gain,first_channel,last_channel]
                          BLUETOOTH [fe_gain,first_channel,last_channel]
                          ZIGBEE    [fe_gain,first_channel,last_channel]
                -fe_gain GAIN
                -first_channel CHANNEL
                -last_channel CHANNEL
                -fft_points FFT size, must be a positive integer and a power of 2
                -detectmode DETECT_MODE
                        The detect mode:
                          1: Average
                          2: Maxhold
                          3: Minhold
```

Figure 29 The help menu to query from USRPCAgent

2.3.5.2 CB implementation on Log-a-TEC

In this section we report on our investigation about the feasibility to implement the CB framework for LOG-a-TEC.

The cognitive radio experimentation part of the LOG-a-TEC testbeds has the following main components:

- The cognitive radio experimentation hardware in the form of the VESNA node with the SNE-ISMTV extension installed outdoor on the public light poles
- The Lightweight Client Server Protocol (LCSP) which is a protocol for controlling and retrieving data from the hardware nodes.
- The Python and JS tools that are wrapped around the testbed functionality.

Additionally, all the meta-data about the nodes is managed by the Sensor Management System (SMS) that also periodically polls all the nodes for monitoring purposes. The SMS uses MongoDB and NodeJS, an event-driven I/O server-side JavaScript environment, for interacting with the database and the nodes.

Comparing this basic infrastructure and functionality offered by LOG-a-TEC with the CB framework, it can be seen that the Repository and Discovery functionality is already present, however the abstractions and implementation are different than the approach taken with the CB. For instance, the CB relies on primitives from the IEEE 1900.6 standard for discovery while the API used by LOG-a-TEC is less biased by IEEE wireless standards since it covers a broader scope of sensor descriptions (including energy consumption and air quality monitoring). Additionally, in the CREW-GENI collaboration an ontology based on the CREW common data format (CDF see CREW D3.1) that will be used for device meta-data description for spectrum sensing experiments is being developed. With respect to the Optimizer and Execution part of the CB, LOG-a-TEC already has implementations of power allocation games based on the Python tools and the LCSP protocol.

Due to several parallel developments within the CB framework, the LOG-a-TEC testbed and the CREW-GENI collaboration during the 3rd year of the CREW project it has been hard to take a final decision on whether the CB framework is suitable for LOG-a-TEC. This investigation will be continued during the 4th year of the project. In the 4th year of the project, we focused on upgrading to LOG-a-TEC 2.0. This upgrade was requested by internal and external users and lead to API changes. We will further consider the suitability of the CB in the light of the new API.

2.3.6 The WiFi Connectivity Agent*

Continuing the work on the Connectivity Brokerage framework developed in the third year of the project, in Year 4 we have also extended the framework to support dynamic discovery and connection setup between Connectivity Agents. This is an extension in view of WiFi devices that can also be applied to all previous scenarios supported in year three. We have developed a prototype that is able to use WiFi beacon packets to embedd CAgent presence information and use it to connect two agents over the Internet.

In the urban environments a high density of small access points can be observed, each have own internet access, each of them is managed by different people or different management systems. Still all those APs have to compete for free spectrum. It is possible to optimize the usage of the spectrum by APs separately, but without the knowledge about other APs in the neighbourhood a lot of the effort needs to be spent just to discover the interference conditions. Much better results are expected when the APs, that are in different management domains, could exchange information about the spectrum and perform joint optimization for the spectrum usage. Although it is a very specific scenario, it can be shown that this case can be generalized to most other WiFi environments in terms of optimization of radio resource usage. For example, if it is possible to collect and exchange the data from home APs, it should be even easier with corporate networks. In this case, it would not be necessary to discover connection possibilities, as this task should be already solved by the corporate management system that manages large amount of access points and clients.

There are a number of possibilities how two AP could connect to each other to exchange spectrum information. First, a centralized solution where all APs connect to the known database and announce their presence there. The interested APs could include GPS position to be able to find neighbour stations. Or, the interested device could scan for beacons from other AP's to register them as own neighbours in the database. Such database would be responsible for collecting the information about active AP and distribute it to the interested devices. Such a solution rises several security and privacy issues. It is also not a trivial task to design such a database. So the other solution would be, that the AP would form an open ad-hoc network that would allow other APs to join just for the purpose of spectrum usage optimization. This provides much better localization of the service and it is easier to solve on the local basis but would require a lot of overhead from the APs that would like to support such solution. Much simpler solution would be if the AP could announce its presence by beaconing the information on how to connect to its own CAgent in the Connectivity Brokerage context. The latter one is the most promising solution and will be described in more details.

It is assumed that each Access Point will have its own Connectivity Agent that is able to optimize the performance of the device. It is relatively easy to connect each client's CAgent to the access points CAgent as they are in the same network and can directly exchange information over the Internet. In order to be able to connect two CAgents together, they have to fulfil three requirements. First at least one of them has to be available on the public Internet for connections from other CAgents. Second it has to announce it's IP and port tuple on the WiFi interface. The other CAgent then can listen for this beacon packets to discover what CAgents are in the neighbourhood. Chandra et.al in [39] have presented that it is possible to modify the standard WiFi beacon packets to include own information. With so called beacon stuffing it is possible to change for example vendor-specific Beacon Information Elements (BIEs), into own information in the beacon packet. The working example provided in [39] is using Windows operating system and Atheros based wireless card. It can be presented that it is also possible with Linux ath5k driver [40]. This idea is reused to add CAgent information into the beacon. This approach will allow to announce the CAgent presence without breaking any of the current WiFi functionalities. This is also the only change that needs to be made in

the driver to make the solution work. Additionaly the change is mainly done on the AP side as the driver needs to be modified to change standard beacon. On the other hand, in the client device the change is less severe and can be done in the userspace. In this case only the analysis of the beacon packet needs to be done.

This solution has also one more advantage over the centralized database with CAgent information. It also provides intristic localization information. Only the devices that are close enough to eachother would be able to receive such beacon. This might not be enough to find all neighbours, especially the hidden node problem will be importand in this case, but once the communication is established with some of the neighbour information about futher nodes can be obtained in much easier way from them also. It is expected that CAgents to form two hop neighbour graphs. It is also importand to note here that legacy devices using WiFi and not supporting Connectivity Brokerage still can be detected and analysed by the CAgents in the proximity. This information can still be propagated to the all neighbouring CAgents over the control channel.

In the Snippet 1 the CAgent announcement message based on the Google Protocol Buffers is presented. There are two required fields: uuid – which is an unique identifier of the CAgent and the class type of the CAgent. Class as also described in the snipped corresponds to the CAgent class type. Other very important element in the message is uri, it is the address under which this given CAgent can be reached. It is not necessary for a CAgent to have this element, but on the other hand it can also have more than one uri addresses pointing to it. For example and Air Interface CAgent would rather connect to other CAgents and not wait for connections to itself. On the other hand a Composite Network CAgent would need probably couple of interfaces to be able to serve multiple heterogeneous networks.

```
message CAgent {
      optional string name = 1;
      required string uuid = 2;
      repeated string uri = 4;
      required Class type = 3;
      enum Class {
            AIR_INTERFACE = 1;
            PLATFORM = 2;
            UNIFORM_NETWORK = 3;
            COMPOSITE_NETWORK = 4;
      }
}
```

Snippet 1 Protocol Buffer CAgent beacon

The sequence graph of discovery and CAgent connection protocol can be seen in Figure 30. The first of the CAgents needs to discover the presence of the other one, as it was described before. Next the discovery module will pass (publish) this information to the interested (subscribed) access manager module. This module is responsible for managing the list of connected CAgents and the specific proxies to those agents. It is also the entry point for new connections. It is important to note here that it is not required that the access manager and beacon discovery modules would reside in one CAgent. It is possbible for example that the new beacon would be discovered by a CAgent responsible for a client station in the network and it will pass this information to the more capable AP CAgent. It is not marked in Figure 30 for simplicity reasons.

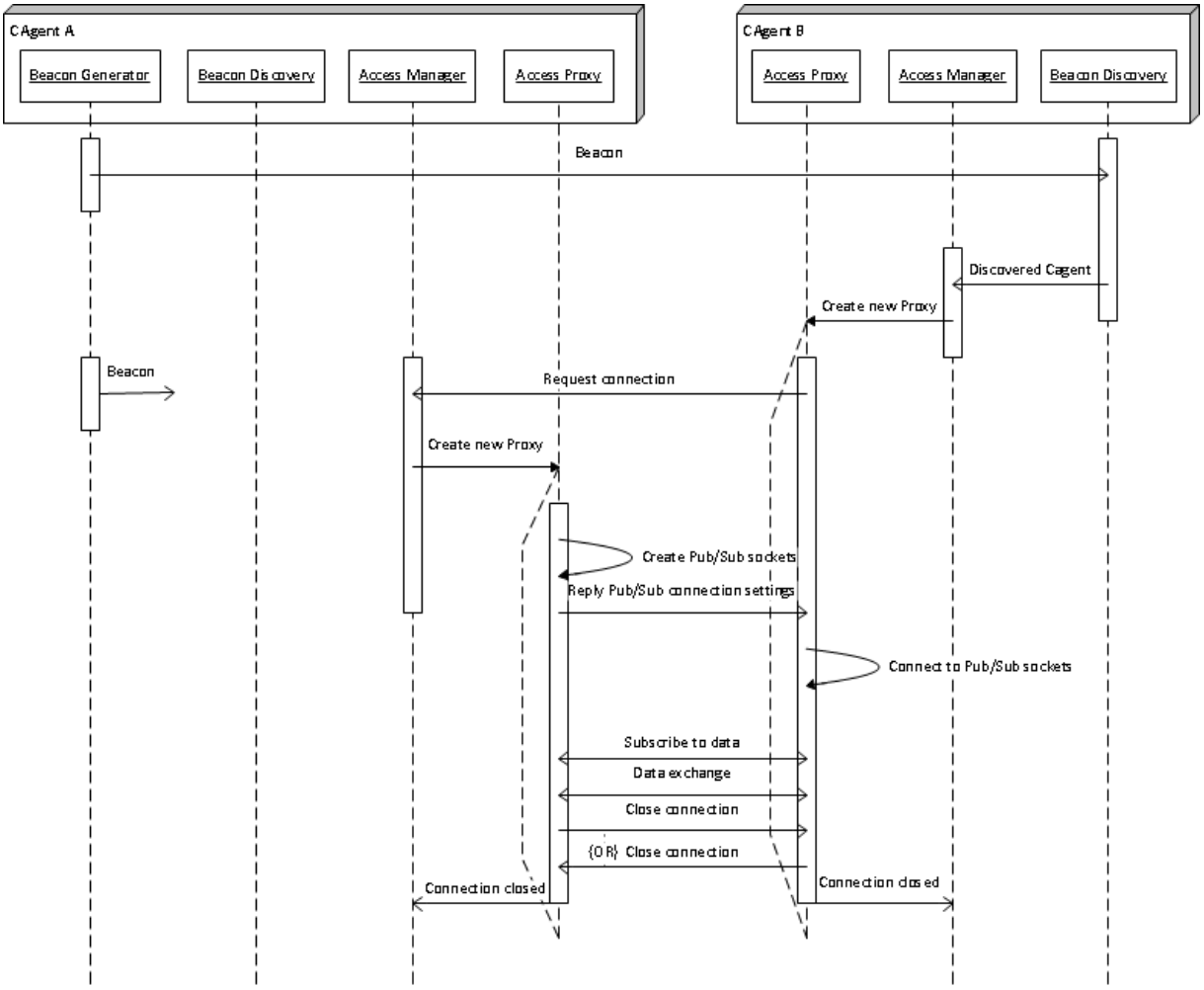


Figure 30CAgent connection establishment

The main task of the Access Manager is to keep track of all CAgents that are connected to it. It is responsible for spawning new Access Proxy modules if there is a new beacon message. It will pass the beacon information to the proxy so it will be able to initiate new connections. It will also store a unique identification of the other CAgent for further reference. The Access Manager is also responsible for reception of the new connection requests.

The Connectivity Brokerage framework is trying to use a Publish/Subscribe type of messaging for communication inside and between CAgents. In the current implementations we have been using the ZeroMQ library for all data exchange. One of the many communication patterns in the library is of course publish-subscribe and that is used for the communication between CAgent modules and between different CAgents. This messaging pattern is however not suitable for the establishment of the new connections. It is by design focused on one way communication and doesn't allow for any reply to the message. It would be hard to implement any advanced connection establishment protocol with publish-subscribe. That is why the Request-Reply messaging pattern will be used for this purpose. The request socket allows for sending one message and expects an answer. This process can be repeated but not changed, e.g. it is not possible to receive a message twice in a row. The idea is to use Request-Reply in the connection establishment phase.

The main goal of Access Proxies is to establish two way Publish-Subscribe connection between two CAgents. Although Access Proxy will send a connection request message to the manager with the connection request, it should talk to another proxy on the second CAgent. To solve this issue the Router socket provided by ZeroMQ can be used. As the name indicates it allows forwarding the request message to another Router or Reply socket, without directly answering with reply message.

Once the Access Manager gets the message on the Router socket it will create the new Access Proxy module and forward the request to it. The Router socket is the one that is announced in the beacon. It can also be bound to multiple addresses and ports at the same time, which greatly improves the flexibility of the system. The Access Proxy receiving the request message, containing beacon from the source CAgent, will create and bind to both publish and subscribe sockets. It is assumed that if the other proxy was able to send a request it will also be able to connect to those sockets. It doesn't have to be true in other direction, due to different routing or network address translation mechanisms. After the sockets creation, it will send a reply message containing both beacon messages and additional pub/sub socket addresses (can be repeated). The Protocol Buffer message prototype is shown in Snippet 2. If the connection between two CAgents is no longer necessary, or the connection gets broken, both proxies will terminate notifying Access Managers about that.

```
message CAgentConnect {
        required CAgent src = 1;
        required CAgent dst = 2;
        repeated string pub = 3;
        repeated string sub = 4;
}
```

Snippet 2 Connection message

Although the whole protocol is relatively simple it allows for easy extensions. For example it could be improved by the security mechanisms authenticating CAgents and encrypting the communication between them. It would only require changes in the Access Proxies extending request-reply scheme of communication by additional authentication mechanisms. Actually no other module needs to be changed.

To test the protocol and later further integrate it into the Connectivity Brokerage framework; the working prototype has been prepared. It is developed in Python, as it much more efficient prototyping language compared to C/C++. Python also allows for easy use of C libraries if there would be a need for faster processing. In the current state of the prototype the structure of the communication and concurrency in the processing is much more important than performance. The prototype uses Python bindings to ZeroMQ library and Google Protocol Buffers. That means it is possible to interconnect with CAgents developed in other languages, both libraries as already mentioned in other deliverables provide a big range of supported languages. The prototype is fully functional according to the protocol specification. The only and biggest limitation is that it does not interface to the beacon stuffing functionality. It is however able to send and receive beacons based on UDP broadcast packets, and thus is possible to demonstrate the functionality over the local network.

In this sectiona discovery and connection protocol for WiFi based CAgents has been showed. This scheme can be however adopted by any other CAgent type. In general it can be adopted by any other CAgent type. For example the beacon doesn't have to be a part of the WiFi packet. It can be exchanged in any other way. New beacon can even be transported inside the existing CAgent pub/sub system to inform other nodes about new device popping up in the neighbourhood. It would allow any agent to make a new connection if required.

It is also easy to replace each module adding required functionalities if necessary. For example the current discovery module can be replaced by any other discovery protocol. Like for example Link Layer Discovery Protocol (LLDP), which allows finding all devices connected directly to local network, like switches, routers, VoIP phones etc. The main problem is, however, that this discovery protocol usually is designed to work on the local network and requires devices to be already connected with each other. This is not the case in our example, where the two APs don't really have any means to communicate.

The flexibility of the solution is very important aspect for the Connectivity Brokerage framework. In the presented scheme the only requirement for connection is a public IP address. That means that any CAgent that is connected to the internet would be able to connect. This is true not only for almost all WiFi devices but also many sensor networks with 6LoWPAN support and Internet of Things applications. The whole solution can be easily adapted and extended if new needs arise, without breaking the whole idea of the solution.

2.4 Modular protocol architecture

We have recently proposed [10] a framework that allows composing communication services in a dynamic way. The framework is based on a fine-grained modular protocol stack architecture. The reference implementation of the framework is ProtoStack[2] while the reference implementation of the modular protocol stack architecture is Composable Rime[3] (CRime).

2.4.1 The components of the proposed framework

The overall framework has four functional components: the physical testbed, the module library, the declarative language, and the workbench as depicted in Figure 31.

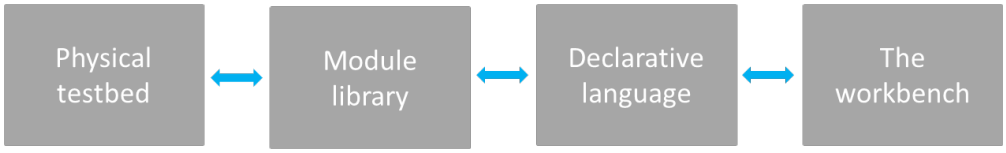


Figure 31 The four components of the framework for the composition of services.

The physical testbed

By physical testbed we refer to a set of machines on which the stack built by the composition of services is deployed and tested. The machines need to support the module library and any additional software that is planned to be deployed. When implementing the framework, the type of machines will determine the selection of the module library, or vice versa. To better represent the likely future deployments, it is desirable that the supported machines of the physical testbeds are as diverse as possible (i.e. heterogeneous). This implies that the module library should be as portable as possible. Further, depending on the location and configuration of the testbed, procedures for resetting the machines in case of fatal errors may be challenging, therefore it is desirable that the deployed binary image is fault proof.

The module library

The module library consists of the source code of the modules used for composing communication services. Besides the code for the modules it also contains additional code necessary for compiling and linking the binary image. Depending on the programming language, the modules are implemented as classes or as a set of functions, each in its own file. The modules provide services to each other through interfaces. One module may correspond to a basic service such as routing (e.g. shortest path routing) or may correspond to composite services such as an entire protocol (e.g. IP).

[2] https://github.com/sensorlab/ProtoStack

[3] https://github.com/sensorlab/CRime

The declarative language

The declarative language is used to instantiate and configure modules from the module library. Subsequently, tools that are able to perform validity checking, error detection, compilation of binary images and their deployment in the physical testbed can be used. The declarative language is a natural intermediate level of abstraction between a user interface such as the workbench and the program code. There is a correspondence between elements of the workbench and the elements of the language. A translation tool is employed to translate from the workbench's elements to the declarative language. In some cases, the user may want to bypass the user interface and directly use the language for describing and configuring the modules in a stack prior to the experiment. As a consequence, the language typically also needs to be human readable, possibly easy to learn and should use intuitive code words.

The workbench

The workbench is thought of as a control panel which allows the experimenter to configure, start, run, retrieve and visualize the results of an experiment. Therefore the workbench should first and foremost contain functionality that would allow the experimenter to intuitively compose a stack and provide initialization parameters. This is typically achieved by having a region where available modules are listed in graphical and/or textual form (e.g. shortest path routing, transmission control protocol). The modules can then be dragged to a workspace, connected and specific parameters initialized (e.g. time to live, maximum number of retransmissions). Some error checking mechanism should be implemented to ensure that incompatible elements are not wired together and that parameters are inside the permitted ranges. Additionally, the workbench can contain an area where the experiment can be visualized while running (e.g. number of dropped packets, delay) and a summary of the completed experiment can be provided (e.g. total time per operation).

2.4.2 Requirements for the proposed framework

The framework for dynamic composition of services has to support design and experimentation of new communication services and modular protocol stacks. In order to achieve this, we identify a set of requirements which help fulfill this objective:

- **Modularity** – the communication services have to have a modular design and implementation to allow composeability of more complex services which can then achieve end to end communication.
- **Flexibility** – the components of the workbench should be designed and implemented in a way that allows interacting with the resulting tool at different levels of abstractions (e.g. at the module library level, at the workbench level). The components should also be easy to extend and upgrade.
- **Easy programming** – users with various levels of programming skills should find it easy to use the tools appropriate to their level of experience resulting from the implementation of the framework.
- **Reproducibility of experiments** – the framework should support re-running and reproducing experiments in an easy way for instance by saving and reloading an experiment description.
- **Remote experimentation** – remote users should be able to define and perform experiments and download the result. This can be most easily achieved through a web portal.

Reference implementations of the framework for dynamic composition of communication services should take this set of requirements as guidelines.

2.4.3 Reference implementation: the ProtoStack tool

In this section we briefly introduce a reference implementation of the framework for the dynamic composition of services called ProtoStack. We discuss the implementation of the framework and we identify the communities which may find such a tool useful.

The implementation of ProtoStack was triggered by a wireless sensor network testbed and is used for experimentation with cognitive radio and cognitive networking in the framework of the CREW project. As such, ProtoStack is designed in a way to ease research and experimentation with communication networks, particularly with cognitive networks. The system was designed so that

i) an advanced user such as the component developer needs to focus on developing the component and make it work with Contiki[4] and

ii) a novice user needs only to focus on composing services in a stack using the workbench.

The physical testbed is based on VESNA sensor network platform[5] to which the Contiki OS has been ported, partly due to the adaptive Rime architecture which comes with it [12]. This physical testbed posed constraints that determined the selection of Composeable Rime (CRime) as the module library.

The CRime module library in the reference implementation is a purpose-built set of protocols influenced by and based on the Rime [12] architecture. The declarative language we selected for ProtoStack is based on the Resource Description Framework (RDF)[6], a standard semantic web language. The implementation uses the Turtle[7] syntax together with existing standardized vocabulary and a custom ontology. The workbench is tightly integrated with the language and is implemented using WireIt[8], an open source javascript library which enables the creation of full web graph editors.

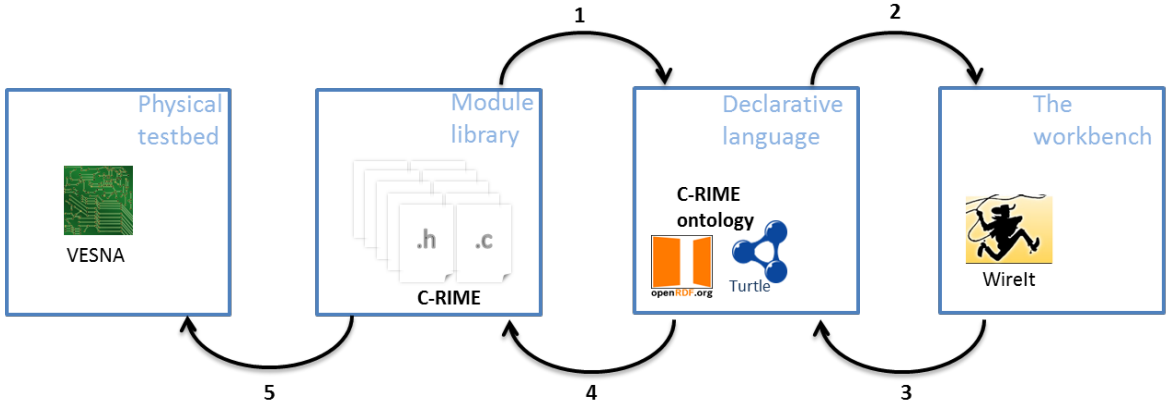


Figure 32 ProtoStack: an implementation of the framework for composing communication services. The implementation addresses the sensor networks domain.

In Figure 32 we illustrate the steps for dynamic composition of services using the ProtoStack tool. The component developer develops a module, manually tests it and makes sure everything works as intended, and, at the end he/she needs to write few lines of Turtle statements (i.e. triples) which specify basic characteristics of the new module (i.e. the name of the module, how many and what type of primitives it implements, etc.). Once this is done, ProtoStack parses the Turtle triples from the new module and stores them in the triple store (arrow 1 in Figure 32). When the user starts using the

[4] http://www.contiki-os.org/

[5] http://sensorlab.ijs.si/hardware.html

[6] http://www.w3.org/TR/PR-rdf-syntax/

[7] http://www.w3.org/TeamSubmission/turtle/

[8] http://neyric.github.com/wireit/

system, the workbench will be automatically populated with modules based on the statements stored in the triple store and rendered (arrow 2 in Figure 32).

The user will then compose the desired stack, insert the required parameters and press a button to run the stack on the physical testbed (arrow 3 in Figure 32). When such a command is received, the system first checks for consistency by making sure the composition of modules is valid and that the input parameters are in a valid range. If all is fine, some C code is automatically generated based on what the user composed (arrow 4 in Figure 32). This code configures the CRime stack. Finally, the source code is compiled into a binary form representing an image that is uploaded on VESNA (arrow 5 in Figure 32).

The experiment description resulting from the stack composed and configured by the user is saved and can be re-used at a later time for re-running the same experiment. All this can be done remotely thanks to the web based workbench.

2.4.4 CRime abstractions

CRime is a new architecture designed to support the composition of communication services which is inspired by and built upon the Rime [12] architecture. CRime introduces three abstractions the *amodule*, the *pipe* and the *stack*.

The *amodule* (short from abstract module) is a generic building block of the CRime stack. Behind each instance of an amodule hides a communication service [13] such as broadcast or multihop. The communication service is an implementation of a network function such as protocol or algorithm and contains only the execution logic of that function. Several amodule instances can be arranged in a pipeline to form a communication stack.

The *pipe* is a vertical structure which can be accessed by any of the modules in a composed stack. The pipe contains only data structures corresponding to parameters that are used by the stack. Pipes are uniquely identified by the channel number they are assigned to, therefore a single channel can only have one associated pipe at a time. This implementation, while not the most efficient approach from a software engineer's perspective, nor the most resource efficient in terms of memory, instantiates the concept of vertical layer and is the first building block in the implementation of the knowledge plane required for experimentation with cognitive networks. The approach is also a compromise between memory footprint and the complexity required by the autogenerated C code based on user input.

The *stack* is a structure which contains a meaningful sequence of amodules and a pipe. It behaves as a container for these elements and enables the composition of more complex communication services which use more than a single channel at a time. Using the stack abstraction, an independent communication stack can reside on each channel. These stacks merge at the application layer or below it. Figure 33 depicts the three abstractions in an example of a 1 channel - 1 stack and an example of a 3 channel – 3 stack communication system.

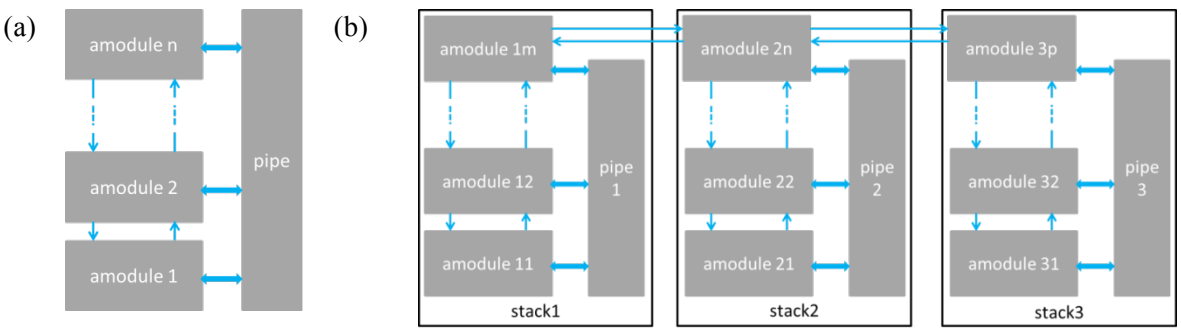


Figure 33 Example of CRime stacks: (a) 1 channel – 1 stack example and (b) 3 channel – 3 stack.

2.4.5 The cost of composeability

Rime was one of the first communication architectures for sensor networks to provide a set of abstractions in order to support adaptive communication. Compared to non-adaptive architectures, Rime incurred higher execution time [12]. CRime is, to the best of our knowledge, the first architecture that supports dynamic composition of services for sensor networks. Through dynamic composition of services, CRime helps to speed up the design, prototype and evaluation of new communication services. Compared to Rime, CRime introduces new abstractions which reflect in overhead in terms of code size, execution time and power consumption. In this section we assess the costs that CRime's overhead introduces versus Rime. In the comparison we focus on a relevant subset of modules and the example applications that include them. The selected Rime primitives against which we compare are abc, broadcast, polite, unicast and multihop, while the Rime applications examples are example_name_of_the_application.

Experimental setup

In order to compare the components of the Rime stack against the components of the CRime stack in terms of the image size we use the VESNA sensor node[9], a hardware platform developed in our lab and used in several sensor testbeds in Slovenia. The VESNA sensor node is equipped with a ST ARM Cortex M3 32 bit microcontroller running up to 72 MHz, 1 MB of FLASH, 96 kB of SRAM and 128kB of fast (2,25 MB/s) non-volatile MRAM memory (NVRAM). The hardware has a fully modular design and can be programmed via RS-232 compatible interface or standard JTAG providing debug capabilities.

For the numbers provided in this chapter, we used the following experimental setup: VESNA sensor node with TI CC1101 radio module connected via serial line to a Lenovo X200 machine (Intel Core Duo CPU @2.53 GHz with 4GB or RAM) with a Contiki 2.5 OS port. The notebook runs Windows 7 Enterprise on the computer on which we installed the open source development environment consisting of Cygwin, Codesourcery tool-chain, OpenOCD, Eclipse Helios. For energy consumption measurements, we connected the hardware to a Tektronix TDS5104B oscilloscope.

Image size

We first compare the components of the Rime stack against the components of the CRime stack in terms of the image size. We looked at the size of the code (.text), the size of the initialized (.data) and non-initialized (.bss) static variables and list the results in

[9] http://sensorlab.ijs.si/hardware.html

Table 1. It can be seen that the Rime and CRime components differ just in the size of the code.
The c_abc code size is 36 bytes larger than the abc code size. This is mainly due to the fact that the c_abc module implements some functionality, namely the c_abc_recv function, the correspondent of which in the Rime version is implemented by the module above or by the application. The multi-hop and mesh communication primitives are noticeably larger in CRime compared to Rime. This is due to several calls to a function implemented by the amodule block which sets values in the pipe structure. The code footprint could be reduced if we optimized only for that, however, we also optimized for maintainability and easy debugging.

Table 1: Comparison of Rime and CRime components with respect to code size (.text), initialized static variables (.data) and uninitialized static variables (.bss). All values are in bytes.

Rime components				CRime components			
Name of component (.o)	.text [B]	.data [B]	.bss [B]	Name of component (.o)	.text [B]	.data [B]	.bss [B]
abc	192	0	0	c_abc	228	0	0
broadcast	248	0	0	c_broadcast	196	0	0
polite	604	0	0	c_polite	580	0	0
unicast	252	0	0	c_unicast	236	0	0
multihop	468	0	0	c_multihop	652	0	0
stbroadcast	348	0	0				
ipolite	712	0	0				
stunicast	512	0	0				
				amodule	2572	0	4
				stack	3760	24	0

For the broadcast, polite and unicast communication primitives, the CRime version results in smaller code size than then Rime version. This is due to the fact that in the case of CRime the functions do not directly call corresponding functions from the modules below. In the CRime case this overhead is solved by the amodule component. If we add in the c_broadcast module explicit calls to the c_abc module, we pay 8 bytes in the size of the code for each call we add.

The stbroadcast, ipolite and stunicast modules from Rime have no direct equivalent in CRime. As discussed in the architectural comparison between the modules, the functionality of these is replaced by triggers and by the modular composition of the stack. The two CRime specific modules, which enable modularity, are the amodule and the stack. The code footprint of these is relatively large if we compare them to the other modules. They also use some static variables as can be seen in the corresponding .data and .bss columns of

Table 1.

Some applications using the Rime modules are already available with the Contiki code and we created a similar application (e.g. dummy sending of packets) entitled example-crime which we use with the composed stacks. The evaluation of the application memory footprint for the two stacks is listed in zed data sections is below 0.1%.

Table 2 and in *Table 3*, while the difference is listed in *Table 4*. It can be seen that the size of the code of the applications which use CRime stacks is about 16% larger (~13.000 bytes) while the absolute size difference of the initialized and uninitialized data sections is below 0.1%.

Table 2 The code size (.text), initialized static variables (.data) and uninitialized static variables (.bss) of five Rime example applications. All values are in bytes.

Application name	Bin fsize [B]	.text [B]	.data [B]	.bss [B]
example-abc	81188	79056	1636	5112
example-broadcast	81260	79128	1636	5116
example-polite	83116	80984	1636	5152
example-unicast	81500	79368	1636	5116
example-multihop	83260	81112	1652	5796

Table 3 The code size (.text), initialized static variables (.data) and uninitialized static variables (.bss) of five CRime example applications (the applications are compatible and do the same thing as the Rime applications from). All values are in bytes.

Application name	Bin fsize [B]	.text [B]	.data [B]	.bss [B]
example-crime (c_abc)	94932	92800	1636	5112
example-crime (c_broadcast)	94976	92840	1640	5108
example-crime (c_polite)	96123	94000	1636	5112
example-crime (c_unicast)	95364	93224	1642	5108
example-crime (c_multihop)	97040	94880	1664	5816

Table 4 The cost of CRime example application in terms of code size (.text), initialized static variables (.data) and uninitialized static variables (.bss). Data compiled based on the values in Table 2 and Table 3. All values are in bytes.

Difference between application using the CRime stack and the equivalent application using the Rime stack	Bin fsize		.text		.data		.bss	
	[B]	[%]	[B]	[%]	[B]	[%]	[B]	[%]
c_abc - abc	13744	16.9	13744	17.3	0	0.00	0	0.00
c_broadcast - broadcast	13716	16.8	13712	17.3	4	0.02	-8	-0.01
c_polite - polite	13016	15.6	13016	16.0	0	0.00	-40	-0.07
c_unicast - unicast	13864	17.0	13856	17.4	8	0.04	-8	-0.01

c_multihop - multihop	13780	16.5	13768	16.9	12	0.07	20	0.03

Processing speed

We expect that CRime is slower than Rime in terms of processing speed due to the overhead the stack and amodule abstractions add. In *Table 5*, we list the evaluation of the processing speed for opening and closing a connection as well as for sending and receiving packets with the two stacks. We used the abc and c_abc communication primitives for the evaluation.

Table 5:CRime vs Rime processing speed (results averaged over 100 runs).

Rime operations			CRime operations		
Name	Duration [µs]	Duration [%]	Name	Duration [µs]	Duration [%]
open	59.0	23.0	open	107.0	17.7
send	104.0	40.5	send	380.0	61.0
recv	71.5	27.8	recv	96.5	15.5
close	22.0	8.50	close	67.0	5.70
Total	256.5	99.8		622	99.9

It can be seen that in both stacks, the most time consuming operation is sending a packet. Rime needs on average 104 µs to send one packet while CRime needs 380 µs, an increase by a factor of 3.5. If we consider the sequence of operations open→send→recv→close, it can be seen that Rime spends 40% of the time sending the packet while CRime uses 61% of the time for the same task. In Rime, the second most time consuming operation is processing the received packet throughout the stack. The CRime open, close and recv operations are relatively simple; the overhead comes mostly from the code necessary to propagate through the tree. The CRime send operation is more complex and incurs higher processing time mostly for the following two reasons. First, the operation is typically sent over one stack. This means that some checking needs to be done so that the operation does propagate on the path of the tree corresponding to that stack and not over the entire tree. Second, this operation needs to handle triggers, for which additional instructions need to be executed.

The total amount of time needed by Rime to execute the sequence of operations open→send→recv→close is ~256 µs. CRime needs 622 µs for the same sequence; this is an execution time which is a factor of ~2.4 higher.

Next, we take a closer look at CRime and the overheads introduced by the amodule and stack abstractions in terms of processing speed.

Table 6 lists the name of the four basic operations in the first column and the overall duration in the second column. In the third column, the names of the CRime functions called in order to execute the functionality of the c_abc stack are listed. For instance, for the open operation, the sequence of functions c_abc_open and c_channel_open are called and their cumulative duration is 73 µs, as listed in the fourth column. The last column lists the overhead introduced by the amodule and stack abstractions (i.e. walking through the tree, performing checks, etc.). This last column is the difference between the second and the fourth, namely between the duration of the overall operation and the duration of the executed functions.

Table 6:The cost of stack and amodule abstractions in terms of processing speed.

CRime operations					
Name of operation	Duration of the overall operation [µs]	Name of the executed functions	Duration of the executed functions [µs]	Amodule and Stack overhead	
				µs]	[%]
open	107.0	c_abc_open, c_channel_open	73.0	34.0	31
send	380.0	c_abc_send, c_rime_output	76.0	304.0	80
recv	96.5	c_abc_recv, c_abc_input	12.0	84.5	87
close	67.0	c_abc_close, c_channel_close	26.0	41.0	61

It can be clearly seen in this breakdown in *Table 6* that the overhead for the send operation where checks for walking through the tree and trigger handling are quite high, representing 80% of the total duration of the operation (304 µs out of 380 µs). The overheads for receiving a packet and closing a connection are also high in relative terms representing 87% and 61% of the total duration of the respective operations. The smallest relative overhead occurs for the open operation, representing 31% of the total duration. The differences between the overheads of the four operations are justified by the different implementations of the tree walking algorithms (for the send operation the trigger handling also has to be accounted for).

Power consumption

In order to evaluate the cost of CRime in terms of energy consumption, we performed measurements using a Tektronix TDS5104B oscilloscope. We measured the power consumption of representative Rime and equivalent CRime applications. Each application was run 10 times for 100 ms and the results are summarized in *Table 7*.

It can be seen that on average, CRime consumes 1.6 % more energy than Rime. In other words, if a battery powered node could run for 365 days sending messages using the Rime stack, the same node could only run for about 360 days sending messages with the CRime stack.

Table 7:CRime vs Rime energy consumption.

Rime power consumption		CRime power consumption		$\Delta P/(P_R)$
Application name:	Consumed power (P_R) [mW]	Application name: example_crime	Consumed power (P_{CR}) [mW]	
example_abc	89.96	(c_abc)	91.84	0.020
example_broadcast	88.96	(c_broadcast)	90.21	0.014
example_polite	88.70	(c_polite)	89.58	0.010
example_unicast	90.09	(c_unicast)	91.84	0.019
example_multihop	91.84	(c_multihop)	93.59	0.019

2.4.6 Introduce learning functionality in the ProtoStack tool*

The ProtoStack tool was designed and developed to enable implementing the cognitive cycle in sensor networks. *Learning* is one of the states of the cognitive cycle along *sensing*, *planning*, *deciding* and *acting*. By introducing learning functionality in ProtoStack, we aim to enable the following use case. The sensors forming the network are sensing the link by observing the instant LQI (link quality indicator) and RSSI (received signal strength indication) values provided by the radio transceivers and/or snooping into all the messages received from the neighbors and monitoring sequence numbers (see Figure 34). In a more advanced scenario where a cognitive radio performing more advanced energy detection or channel occupancy detection can also be used to sense. The *planning* and the *learning* states can materialize in models, estimators or predictors of the link quality. For instance, a moving average of the sensed properties can be maintained to be able to tell more about the link than provided by an instant measurement. Additionally, simple prediction models such as linear regression can be learnt over time from the sensed parameters. The *decide* state can materialize in the routing algorithm that selects the best route. With traditional wired networks, a decision can be made based on a shortest path (SP) policy which is performed by the SP routing algorithm. However, in wireless networks, the decision can be made by the SP routing algorithm taking into account sensed (i.e. instant RSSI) or learned (i.e. prediction provided by the linear regression) information or by other algorithms such as minimum transmission (MT) that may give less weight to the hop count and more to the scores provided by the link quality estimators. The *act* state consists of sending the packet on the route selected by the decider.

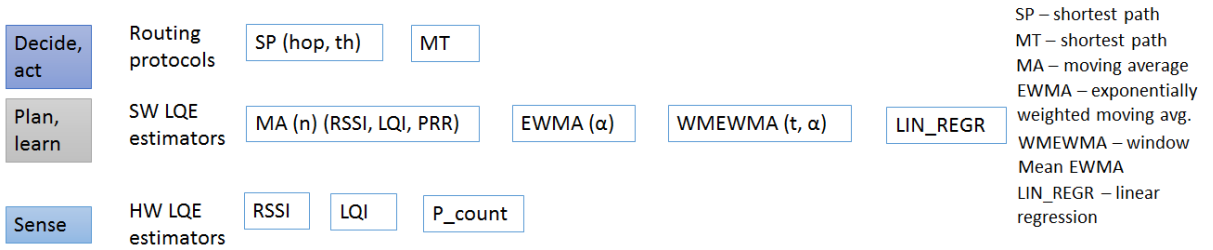


Figure 34 Example LQE and routing protocols relevant for a cognitive networking scenario.

The cognitive networking use case, on which we show the advantage of using tools such as ProtoStack, considers running extensive experiments in which the options for the *plan*, *learn* and *decide* phases of the cognitive cycle are extensively explored to learn and understand more about the multihop performance of changing network topologies determined by varying link quality. As shown in the literature [41][42][43][44][45][46], these kind of explorations need to be performed frequently as new transceivers and more powerful microprocessors supporting more advanced features are used by sensor nodes. Often, the resulting implementations reflect in standardization efforts such as IETF drafts [42][44].

The design space for understanding and optimizing advanced routing in wireless networks along the stages of the cognitive cycle is large and labor intensive. Figure 34 illustrates this space and possible combinations for a selected set of instantiations of the cognitive cycle. Sensing of a choice of link properties such as RSSI, LQI or received frames can be coupled with any *planning* and learning techniques that estimate the link quality and with any of the decision techniques. More choices that further increase the design space can be found in [47][41] etc. ProtoStack decreases the overhead and required effort for exploring this space as follows. First, it implements 5 link quality estimation (LQE) modules (see Figure 35): c_link_stats, c_lqe_ma, c_lqe_ewma, c_lqe_wmewma and c_lqe_linregr which correspond to link statistics of average and standard deviation of RSSI and LQI, link quality estimation using moving average (MA), estimation using exponentially weighted moving average (EWMA), link quality estimation using window mean with EWMA and link estimation based on linear regression, respectively. These modules include adjustable parameters such as time window t size for the moving average estimator and α for the EWMA estimator. Each of these modules fills in a

data structure named *neighbor_list* located in the common pipe structure. The neighbour list contains a list of all neighbours from which the current node N heard messages. Besides the neighbour's Rime address, the data structure holds time averaged RSSI and LQI and a cost. This cost is computed differently, depending on which software LQE is being used. CRime's c_multihop service extends the routing table with a cost, additionally to the already existing hop count. This routing table is also contained in the pipe structure. Besides supporting the shortest path routing, CRime's c_multihop can thus be configured to use shortest path based on thresholding where instant or time averaged LQI and RSSI values are used from the pipe structure's neighbor_list that is filled in by the LQE modules and shared with all the other modules. Additionally, the cost from the route_table can be set based on values provided by the LQE estimators according to the strategy of the decider.

Figure 36 lists all the possible combinations of a data stack using the elements that instantiate the cognitive cycle as discussed so far in this section.

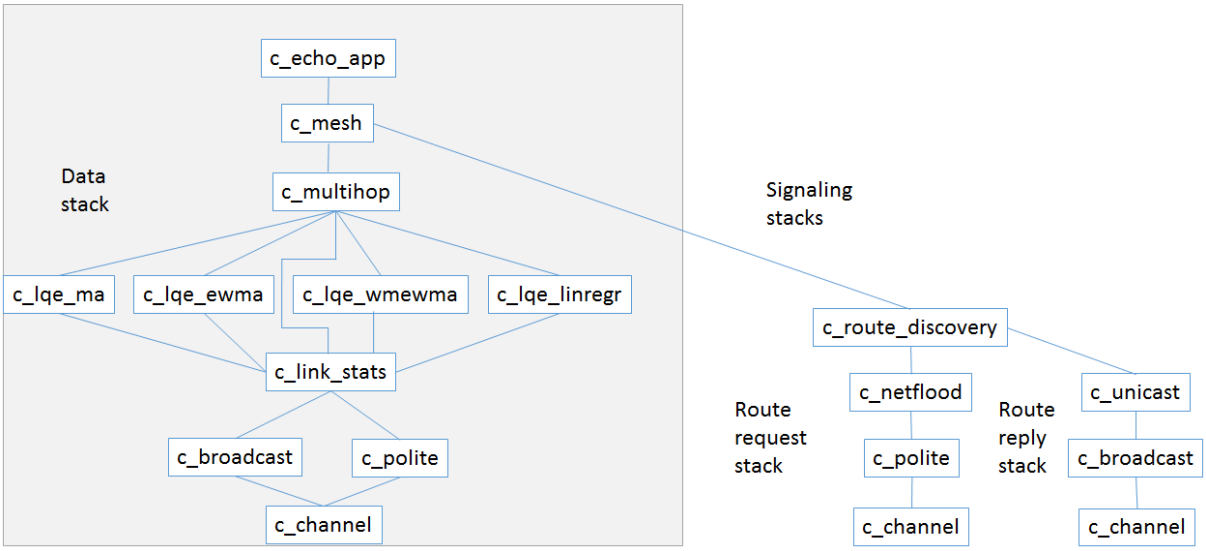


Figure 35 Dependency graph of a stack achieving multihop communication using ProtoStack with focus on the cognitive networking options for the data transmission functionality represented by the Data stack.

c_channel, c_broadcast, c_link_stats, c_multihop, c_mesh, c_echo_app

c_channel, c_broadcast, c_link_stats, c_lqe_ma, c_multihop, c_mesh, c_echo_app

c_channel, c_broadcast, c_link_stats, c_lqe_ewma, c_multihop, c_mesh, c_echo_app

c_channel, c_broadcast, c_link_stats, c_lqe_wmewma, c_multihop, c_mesh, c_echo_app

c_channel, c_broadcast, c_link_stats, c_lqe_linregr, c_multihop, c_mesh, c_echo_app

c_channel, c_polite, c_link_stats, c_multihop, c_mesh, c_echo_app

c_channel, c_polite, c_link_stats, c_lqe_ma, c_multihop, c_mesh, c_echo_app

c_channel, c_polite, c_link_stats, c_lqe_ewma, c_multihop, c_mesh, c_echo_app

c_channel, c_polite, c_link_stats, c_lqe_wmewma, c_multihop, c_mesh, c_echo_app

c_channel, c_polite, c_link_stats, c_lqe_linregr, c_multihop, c_mesh, c_echo_app

Figure 36 List of possible composition options for the data stack using selected services offered by the ProtoStack tool.

As a reference scenario to explain the benefits of using the ProtoStack tool, we consider [41] in which the authors were looking for a simple and reliable link estimator (or predictor in machine learning terminology) that would be used by the routing protocol in deciding which route to select. Since the search space for finding and evaluating the estimator and the performance of the routing algorithm is very large, the authors take a four stage approach: 1) determine empirical link characteristics, 2) investigate link estimation techniques, 3) design a neighbour management policy and 4) design and implement the routing framework. This approach can be carried out using the ProtoStack tool as follows.

2.4.6.1 Empirical link characteristics determination*

Empirical link characterization stands at the basis of any work related to multihop wireless network design and is performed in several works such as [41][45][46] to name a few. This stage is carried out using real hardware and the results vary across transceivers and deployment environments, so it has to be redone frequently. The result of this step is a statistical model of the channel on which the design and evaluation of LQE estimators is based on. ProtoStack can be used in this stage of work in two ways. First, it can be used to ease the configuration and deployment of the experiment where a <c_channel, c_broadcast, c_link_stats, c_echo_app> stack would achieve the required function. Second, and most relevant for research with cognitive networks, the same stack can be used to enable each node to build automatically a statistical model for the link quality. Rather than manually computing the link model and subsequently use the same model on all nodes, each node can build its own model.

2.4.6.2 Link estimation techniques*

Link estimation techniques are typically designed using a statistical model of the link that is often based on purely theoretical model and sometimes in empirical observation such as in the works considered in this paper. Since the number of link estimation techniques tends to be large and each has at least one tuning parameter, their evaluation is performed in a simulator using a statistical model. With ProtoStack, the cost for empirically evaluating these techniques is lowered because 1) a new link estimator is not significantly harder to implement then it would be in a simulator and 2) once implemented, this estimator can be even remotely added to a stack and configured via a graphical user interface. With the increasing number of available sensor testbeds and their increasing size [48], and with a tool such as ProtoStack that adds a simulation-like interface to them, empirical evaluations should become more prevalent also yielding more realistic results compared to those obtained by simulations. Since link estimation techniques embody stages of the cognitive cycle, and ProtoStack already includes advanced link quality estimators such as the one based on linear regression (c_lqe_linregr), also empirical experimentation with self-adaptive and self-learning strategies should become easier.

2.4.6.3 Neighbour management policy*

Neighbour management policies aim at finding the best way for a node to determine, over time, how large a neighbour table should be and which nodes should be added to this table. The size of the table as well as the periodic refresh time are parameters in ProtoStack and can be varied across experiments to have an empirical validation of its effect on routing. The currently available LQE estimation modules are in charge of filling the neighbour list of the pipe structure, each of them using a certain strategy. These modules would need to be extended or new ones added with various approaches to neighbour addition and eviction to support experiments concerning neighbour management policies.

2.4.6.4 Design and implement the routing framework*

The design and implementation of the routing framework has to take into account the previous three stages and also to deal with routing specific aspects such as routing strategy and algorithm, table management, cycle detection, etc. A small set of routing protocols using the most promising link estimation techniques is typically evaluated empirically. Using any of the stacks listed in Figure 35, ProtoStack can support easy experiment configuration, deployment, execution and monitoring of shortest path, shortest path considering link quality and minimum transmission protocols.

2.4.1 Conclusions*

We have recently proposed in [10] a framework that allows composing communication services in a dynamic way. The framework is based on a fine-grained modular protocol stack architecture. The reference implementation of the framework is ProtoStack[10] while the reference implementation of the modular protocol stack architecture is Composable Rime[11] (CRime). In this report we briefly described ProtoStack and CRime and provided a quantitative evaluation of the overhead introduced by composeability by comparing CRime against Rime.

More details on ProtoStack and CRime are available in [10] where we show, through feedback collection from first time users, that the ProtoStack tool can significantly speed up prototyping and testing of new stacks and is friendly to novice and advanced users. The initial feedback shows that the tool can speed up design and prototyping of new protocol stack by at least a factor of 2. The cost of increased flexibility and prototyping speed of the protocol stack is paid in terms of increased memory footprint, processing speed and energy consumption. The CRime library used by ProtoStack has a 16% larger footprint, it takes 2.4 times longer to execute an open->send->recv->close sequence and consumes 1.6% more power in doing so. Even though with ProtoStack more resources are consumed by the node, the tradeoff in terms of prototyping speed seems to pay off.

*As part of Year 4 activities, ProtoStack has been extended with LQE functionality including some based on machine learning.

2.5 Testbed-specific Optimization

2.5.1 OMF

The TWIST testbed has very stable and well-established architecture for experimenting with wireless sensor network applications. It also has a web interface for experiment control. However during the time of the project we have extended the functionalities to new devices. In this section we describe the improvements to testbed with respect to the devices other than sensor nodes.

The current TWIST web interface is sensor node oriented. That is the reason to keep it only for this purpose and install new framework of experiment control for other devices. We have tried to follow iMinds in this case and install the OMF framework for testbed control. There where however couple of issues that prevent direct clone of the OMF in the TUB testbed. Currently the iMinds testbed uses OMF in version 5.4 which is very focused on the usage with embedded PCs. This implementation was very hard to adapt to other devices that are part of the testbed such as the spectrum analyzer or signal generator. It also requires the Aggregate Manager to control the experiments. It is used to store the inventory, disk images and measurement collections. It also introduces additional overhead.

In the recently released OMF version 6.0 the whole system was greatly simplified. The new architecture is shown in Figure 37. The main focus is put at the Resource Controller (RC). In contradiction to previous versions of the, OMF it can run on the resource itself (like PC) or run

[10] https://github.com/sensorlab/ProtoStack

[11] https://github.com/sensorlab/CRime

separately and control other resources (like sensor nodes) [11]. This is a big improvement from the TWIST testbed point of view. It is now possible to install new RC entity that will be able to control TWIST nodes, Spectrum Analyzer or TWISTbot without major changes in the currently existing infrastructure. It is also said that OMF 6.0 is backwards compatible which means we still will be able to reuse the experiment scripts from other partners.

2.5.1.1 Spectrum sensing tools

In this section we describe the integration of spectrum sensing devices into the OMF framework. We introduce the motivation for such an improvement, then the details of implementation are explained and finally we also demonstrate the benefit for such approach.

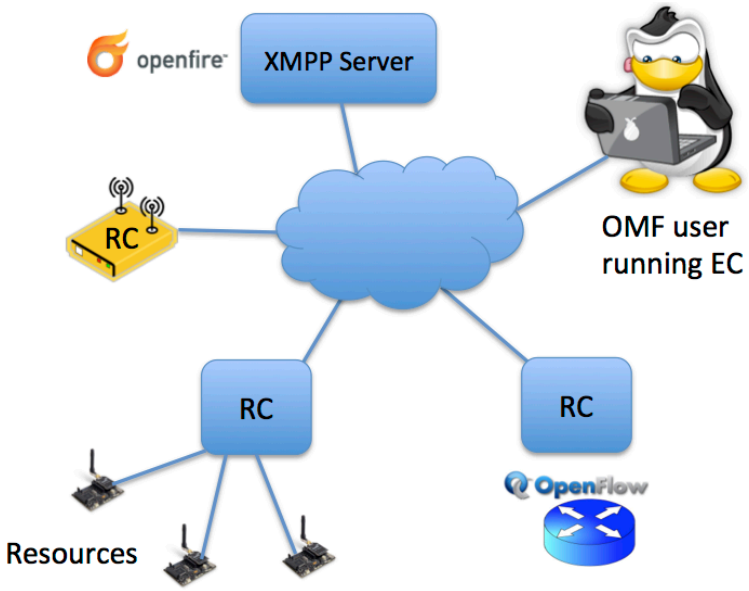


Figure 37 OMF 6.0 architecture [11]

It is possible to use different sensing devices at the TWIST testbed. They are ranging from low quality but simple TelosB nodes with sensor application, through WiSpy USB dongle, to the high performance Rohde & Schwarz (R&S) FSV7 Spectrum Analyzer. All can be used for different purposes with the great success but have different interfaces to gather the data. They also produce the results with different data formats, which also make it difficult to process the measurement data.

It would be impractical to try to uniform the basic interface to all of the devices. That is why we have developed wrappers around the standard interface of the devices. The main job of the wrapper is to correctly start measurements and store the data in the known location. The user does not have to worry about the knowledge of individual initiation and starting procedures, as they could be difficult to remember. This was especially true for TelosB devices, where it is necessary to program the node with the correct application, connect to the node over the serial interface and only then start the measurement gathering application.

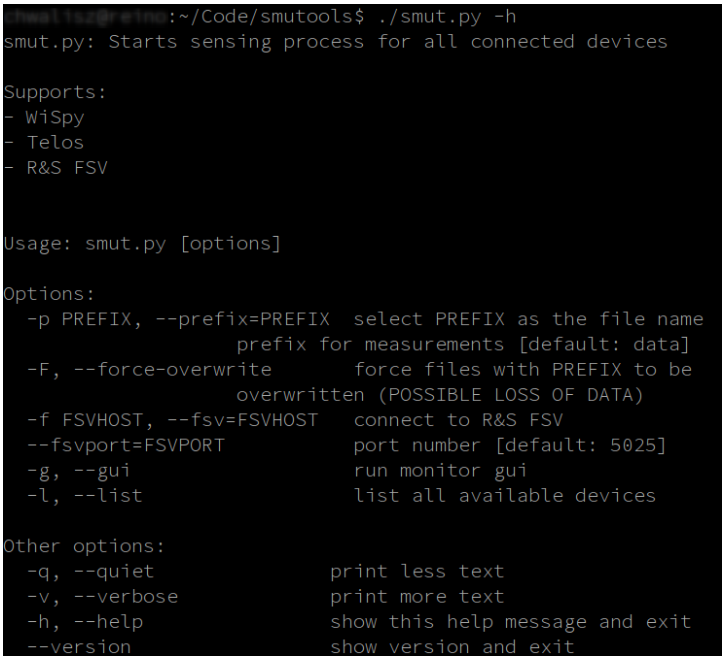


Figure 38 Spectrum sensing interface

The wrappers are implemented as the Python classes and can be started separately or with the common interface that looks for all devices connected to the PC and start measurement on all of them. The interface is shown in Figure 38. In the case of R&S FSV7 spectrum analyzer the user is responsible for the configuration of the device. It has too many options to be able to expose them in the data gathering tool that has the aim to simplify measurement procedure. The idea is to let the user set all the parameters separately and once they are correct easily and safely start the measurement.

The tool also provides graphical user interface for live preview of the measurement data. Example preview can be seen in Figure 39. It is independent to the data collection and can run without interfering with the measurements. It reads already stored data from the file and plots two graphs. The top one is the analyzer view with the current, average, minimal and maximal power per frequency. The bottom shows the spectrogram, where frequency is in X-axis, time on Y-axis and the power is represented by different colors.

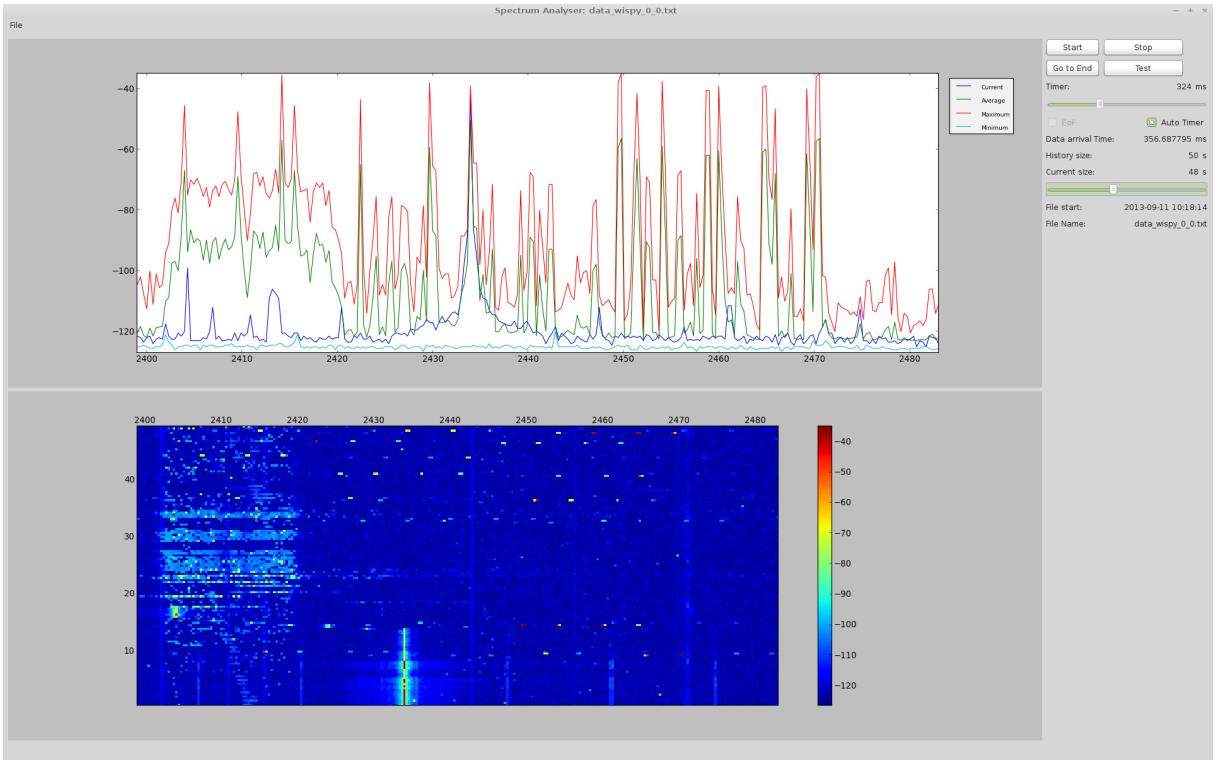


Figure 39 Measurement preview tool

2.5.1.2 CREW common data format processing scripts

All measurement tools tend to store data in its own format. It is done due to efficiency reasons. It also makes it harder for experimenter to analyse the data. In the previous years of the project we have developed set of Matlab processing scripts that aim to simplify the process of loading and working with the data. This year the scripts where migrated to the git repository for better update possibilities and are available under: http://www.crew-project.eu/repository/scripts or https://github.com/mchwalisz/crewcdf_toolbox

The Rohde & Schwarz Spectrum Analyser support has been added. It covers two types of file formats produced by custom developed measurement tools described previously and proprietary tool from Rohde & Schwarz called TraceRecorder. Both follow the same principles as for other devices where it is only necessary to provide path to the trace file and the function will load the data to the common structure as described in the following example:

```
% p = common data format structure
% p.Name       = Unique identifier of the sensing device
% p.Location   = Location of the sensing device (m) e.g [x,y,z]
% p.CenterFreq  = Array defining center frequencies
%                of the columns of power the matrix (Hz)
% p.BW         = Bandwidth around each center frequency (Hz)
% p.Tstart     = Start time of the measurement in datestr format
%                e.g. '24-Jan-2003 11:58:15'
% p.SampleTime  = Timestamp relative to Tstart (s)
% p.Power      = Matrix containing power measurements (dBm)
%                row contains all frequencies for one timestamp
```

The scripts were also extended with new plotting functions. Additionally, to the previously used spectrogram showing the measurement in the time vs. frequency plot where different power levels were represented by different colours, the so called persistence plot can be produced. The sample plot can be found in Figure 40. This type is new and very interesting way of showing spectrum measurements and is available in the newest high end spectrum analysers. It is standard power vs.

frequency plot but additionally the colour on the graph shows how many times given value was recorded during the measurement. It gives a great possibility to see and recognize rare signals in the measurement. It is not possible with any other type of plot. For example it would not be possible to see the narrow band spikes that are around -75 [dBm] strong with averaging, as the noise is received so much more often or maximum hold, as there are other signals that would cover the view. To use this function it is enough to run crewcdf_persistence(p) with the standard common data format structure as a parameter.

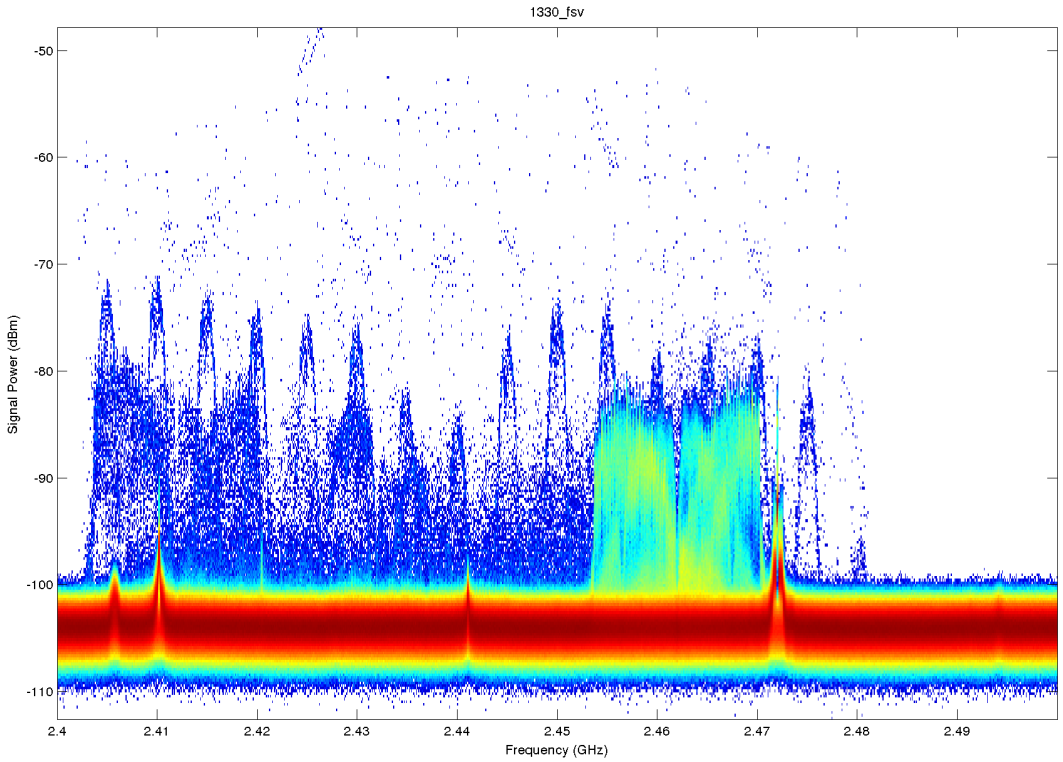


Figure 40 Sample persistence plot

2.5.2 IRIS-GUI

2.5.2.1 Graphical interface support

A key demand-driven extension for the Iris software radio architecture has been support for graphical interfaces for radio control and signal analysis. This extension has been built into the core Iris framework through support for Qt-based graphical widgets. The graphical support has been designed in such a way that graphical widgets may be easily added to any part of the Iris framework including the radio launcher, individual components and controllers. In extending Iris to support graphical widgets, it was vital that this extension should not compromise the use of Iris in non-graphical environments such as on embedded systems and headless servers. To achieve this, the CMake build environment was leveraged to ensure that graphical support is only built-in when the requisite graphical libraries are available on the system. In the absence of these libraries, Iris will successfully build and run but will not include graphics support. In addition to building graphics support into the Iris core framework, a suite of reusable graphical widgets and controllers were developed. These are described in the following sections.

2.5.2.2 Graphical interfaces for signal analysis

A suite of reusable graphical widgets have been developed to support signal display and analysis in the Iris framework. These include the following four widgets:

1. Scatter plot
2. Real plot
3. Complex plot
4. Waterfall plot

The Scatter plot widget supports the display of a vector of complex data points in two dimensions. This is useful, for example, to display the constellation of a received digital radio signal (see Figure 41).

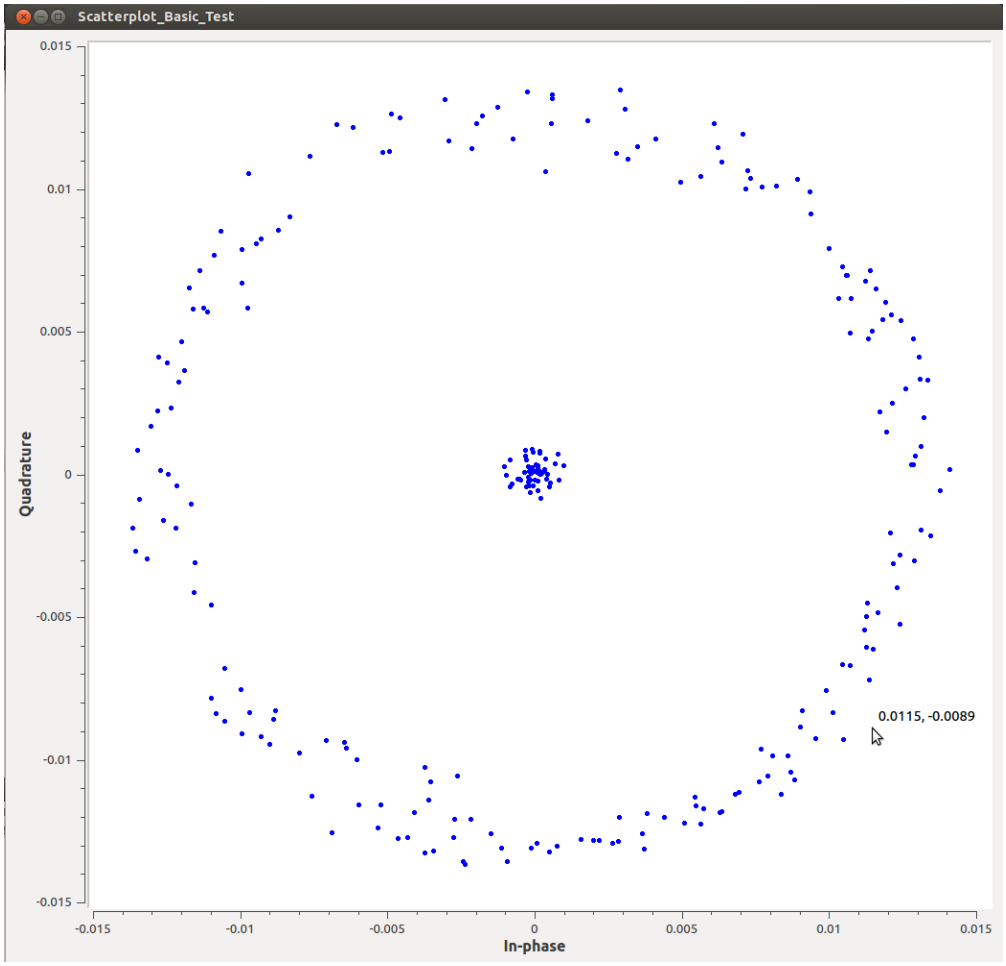


Figure 41 Scatter plot widget example

The Real plot widget can be used to display a vector of real-valued data points such as received signal magnitude. An example can be seen in Figure 42 below.

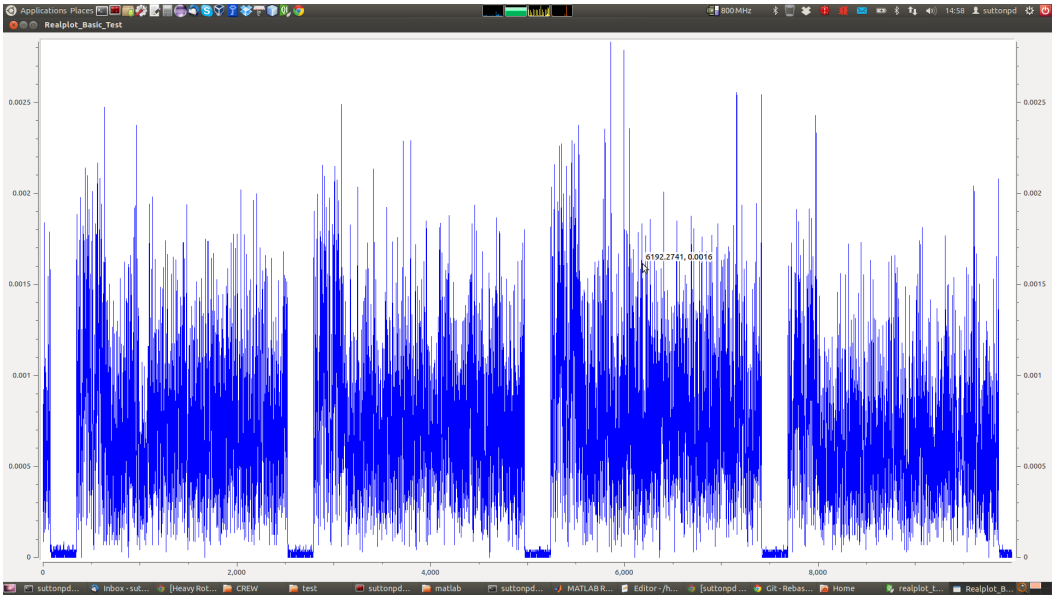


Figure 42 Real plot widget example

Similarly, the Complex plot widget can be used to display a vector of complex-valued data points. The Complex plot widget includes four subplots for in-phase, quadrature, magnitude and phase components of a complex valued vector. Again, an example can be seen in Figure 43 below.

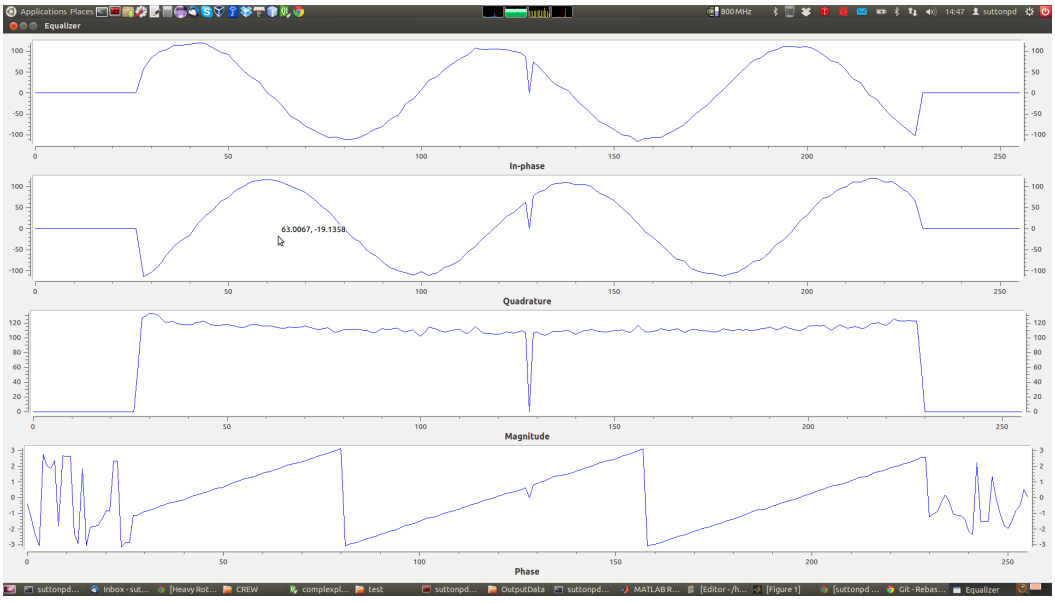


Figure 43 Complex plot widget example

Finally, the Waterfall plot widget can be used to display the historical values of a vector with colours used to indicate intensity. This widget is especially useful, for example, to monitor spectrum usage over a range of frequencies. An example can be found in Figure 44 below (here included as part of a larger graphical interface used for recording the throughput of two wireless links).

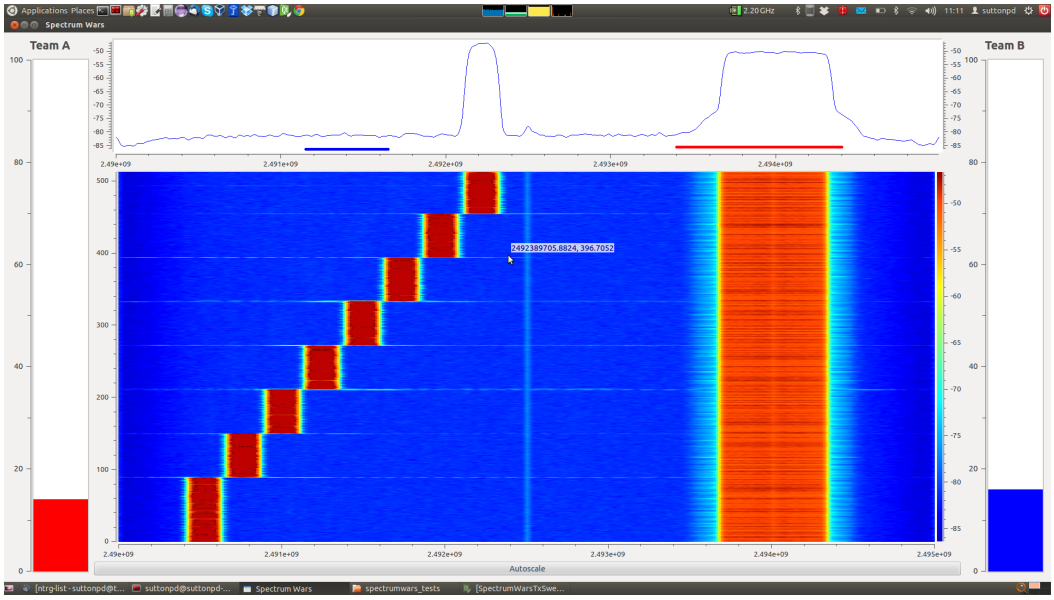


Figure 44 Waterfall plot widget example

Each of the signal display widgets is designed to support both static and time varying data. Every 10ms, each widget will check to see if new data has been provided and will replot if so. This approach ensures that the data refresh rate does not exceed 100 frames/sec (thus necessarily consuming processing resources).

2.5.2.3 Graphical interfaces for radio control

In addition to the signal analysis widgets described above, support has been built into the Iris framework for radio control widgets. These widgets can be used to manually control the performance of an Iris radio system and are often created within an Iris controller, with a global view of the running radio and the ability to reconfigure any parameter exposed within any component. Figure 45 below illustrates one such widget, provided to support reconfiguration and control of an RF front end.

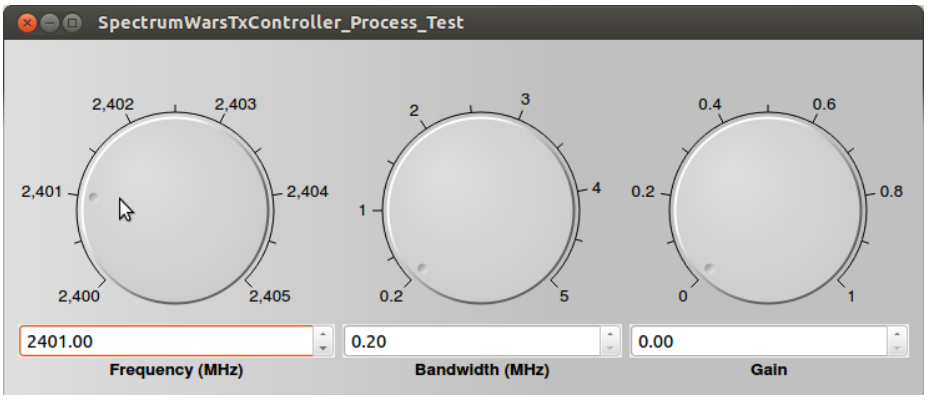


Figure 45 RF front end controller example

2.5.3 GRASS-RaPlaT

2.5.3.1 LOG-a-TEC web portal ↔ GRASS-RaPlaT – API and functions

The LOG-a-TEC web portal[12] communicates with GRASS-RaPlaT by issuing an OS-level command (implemented as a Python script) for the execution of a particular function, and providing the necessary input data with the command-line parameters and in one or more input files. GRASS-RaPlaT performs the required computation and returns results in one or more output files.

Input files are text files containing the required input data in a format specified for each function. Output files are text files containing the computed data in a format specified for each function, or a KMZ (compressed KML) format files containing a geo-referenced raster image (e.g. radio signal coverage).

The GRASS-RaPlaT command interface for the web portal includes an additional intermediate layer that temporarily creates GRASS runtime environment for each command issued by the web portal, allowing execution of GRASS commands directly from the OS-level command interface (instead of from within a running GRASS user session, as is normally the case).

The LOG-a-TEC web portal ↔ GRASS-RaPlaT communication is illustrated in Figure 46.

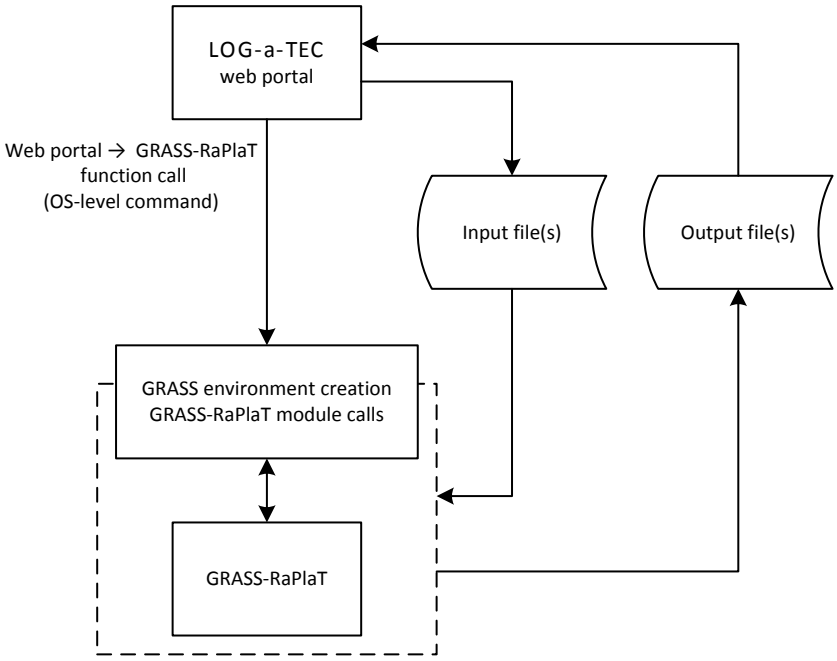


Figure 46 LOG-a-TEC web portal ↔ GRASS-RaPlaT API.

Currently, GRASS RaPlaT provides three basic functions for the CREW web portal:

- Computation of coverage or interference area. The results is a KMZ file containing a raster image of radio coverage (colour-coded received signal strength in each raster point), or interference area (single-coloured area).
- Computation of numeric received signal strengths (in dBm) for a given list of receive points (specified by their coordinates, i.e. longitude and latitude in degrees). This function is related to the coverage computation, but returns numeric values for a set of points instead of a colour-coded raster image of the entire area.
- Estimation of an unknown transmitter's location and transmission power (transmitter localisation), based on received power values from a set of receivers.

[12] http://log-a-tec.eu/

These functions can be combined by the web portal to achieve advanced functions. E.g., the transmitter localisation and interference area computations can be used in turn to compute the interference area of an unknown transmitter.

The first two of the above functions are rather straightforward and do not require further explanation (beyond the description of their use in the CREW web portal, given elsewhere). The newly implemented transmitter localisation function, however, is more specific and its internals are described in greater detail in the following section.

2.5.3.2 Unknown transmitter localisation

The transmitter localisation function returns the 2D location (longitude and latitude) and effective radiated power (ERP) of an unknown transmitter, based on the receive power measurements of at least three suitably placed receivers.

The function first computes path loss raster maps for all receivers (as if they were transmitters), which is a standard function in RaPlaT. By reinterpreting the loss values as gain values (by changing their sign), the receiver/transmitter roles are effectively interchanged. Using the standard GRASS-RaPlaT coverage computation procedures with these *gain* raster maps instead of the path-loss raster maps, and with the received power values as the transmit powers, estimated transmit power raster maps for the unknown transmitter are obtained as seen from each receiver. In this set of raster maps, the function searches for the point(s) where the estimated transmit powers agree, which would represent a possible location of the unknown transmitter. Actually, the function (more precisely the underlying RaPlaT module) first computes a *similarity* raster, where the value in each point represents the similarity between the expected transmitted powers for all receivers. The values are non-negative, with 0 (zero) meaning a complete agreement (ideal solution), and larger positive values larger disagreements between the estimated power values. The transmitter localisation function then searches this raster for a location with high similarity (low value). Since the location estimates are not exact but rounded to the raster resolution, the value of zero is generally not achieved. The current implementation calculates similarity values as the sum of absolute differences from the mean value for each raster point, and searches the similarity raster for the point with the lowest value.

The function returns two files. One is a KMZ file with a colour-coded similarity raster image. The other is a text file containing the estimated transmitter location (longitude and latitude) and its effective radiated power. Actually, there is not only one power value but a list of them, since the values estimated by each receiver are given in the file. Ideally, these values would be equal, but due to rasterisation there are generally small differences. When using more than three receivers, disagreements between them also enlarge the differences (up to the point where the transmitter position cannot be estimated any more).

The transmitter localisation function takes into account radiation patterns of the receive antennas. The transmitter antenna is unknown and an omnidirectional diagram is assumed (e.g. a vertical dipole/monopole surrounded by free space); the results can be completely wrong if this is not the case. Since the transmit antenna gain is unknown, only ERP (Effective Radiated Power) can be estimated, and not TPO (Transmitted Power Output).

At least three receivers are required for transmitter localisation, however with only three receivers two possible solutions exist. Both solutions generally differ in the estimated transmit power. The localisation function returns the position of one of them which is not necessarily the right one. By setting upper and/or lower bounds on the transmit power (optional command line parameters), the right position and power estimate can be obtained. This requires suitable positioning of the receivers relative to the transmitter, i.e. the transmitter may not be in close proximity of one of the receiver, because in such case both transmit power values would be very similar. For omnidirectional receiver antennas, the ideal positions of the receivers would be in the corners of an equilateral triangle with the transmitter in the centre. (In this idealised case, the second solution would be theoretically infinitely far away, with infinite transmit power. In practice, the second solution would be out of the computation area, or at least very far away from the receivers and with a very high transmit power.)

Figure 47 shows a simple example with the transmitter TX1 at equal distances from the receivers RX1 and RX2 and close to RX3. In this case, TX2 with about 4,6x larger transmit power then TX1 would produce the same received signal strengths at the receiver locations.

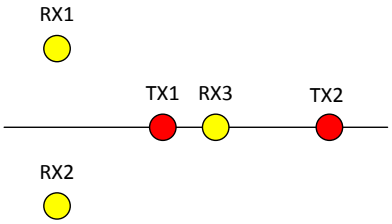
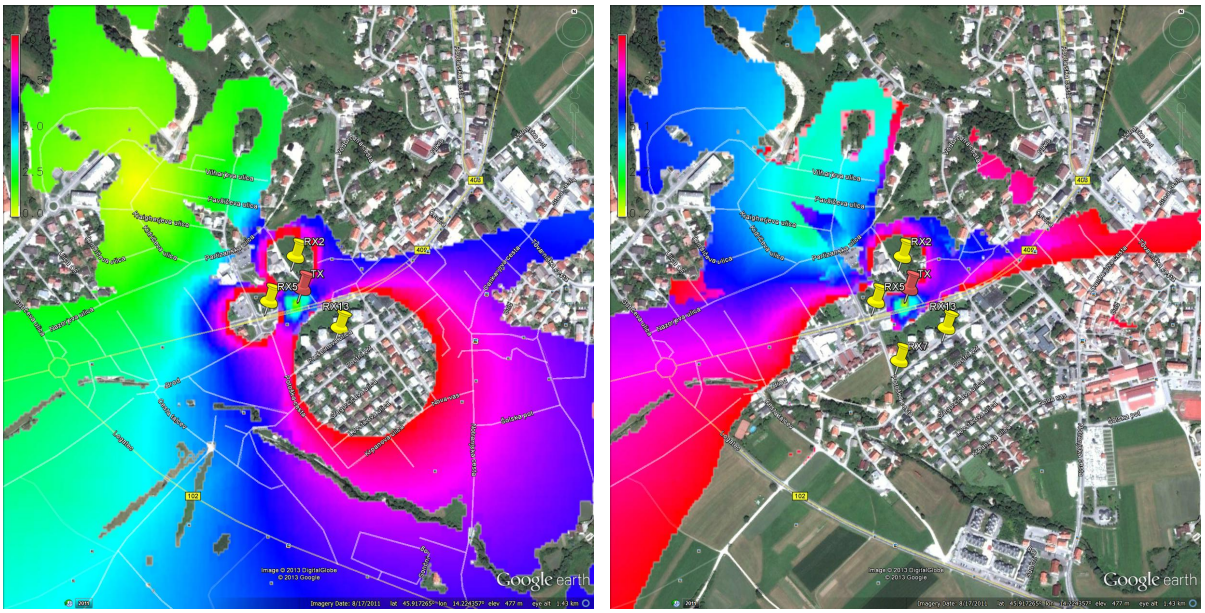


Figure 47 A simple illustration of two possible solutions with only three receivers.

A unique localisation solution can be found without providing transmit power bounds by using measurements from at least four suitably placed receivers. Figure 48 illustrates this by showing similarity raster for 3, 4, 5 and 6 receivers (yellow markers); only in the first case (3 receivers) the solution is not unique, with an alternative transmitter location in the upper left region (yellow colour) and its estimated power 13,7 dB higher than for the actual location (red marker).



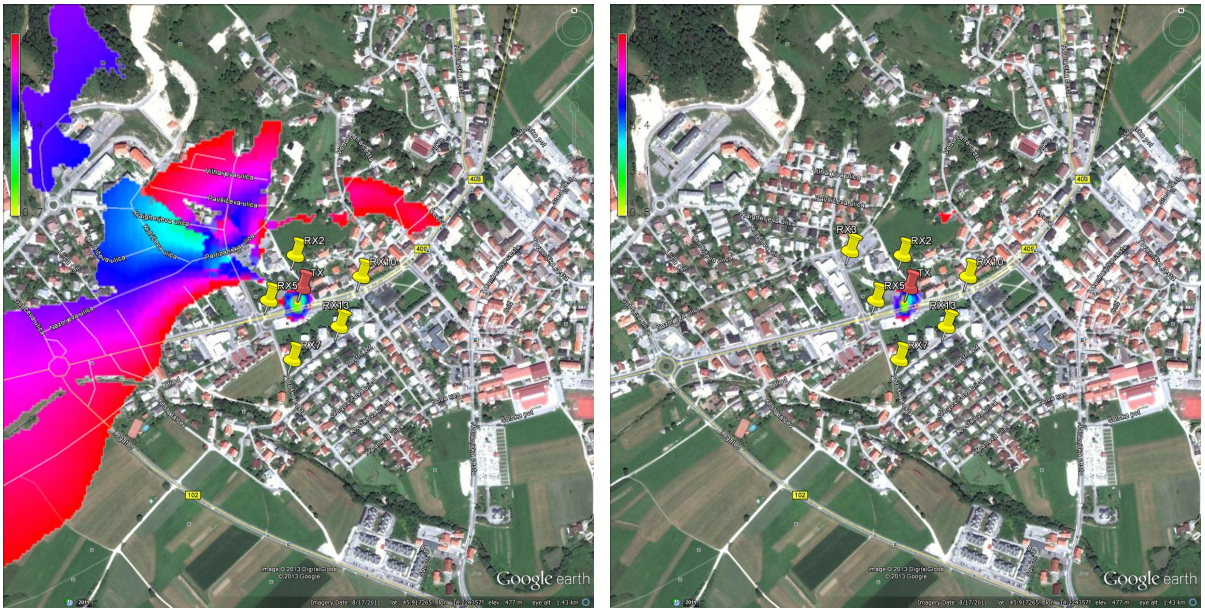


Figure 48 Probability raster images - 3, 4, 5 and 6 receivers.

The measurements from multiple receivers must be in agreement, i.e. without large individual errors, since these would prevent finding a valid solution (if necessary, an improved strategy may be implemented in the future to mitigate problems with erroneous measurements).

It should be kept in mind that the current computation raster is 5 m (limited by the available DEM raster resolution), and the distances between the receivers and the unknown transmitter should be well above this value to avoid numeric errors and problems due to rasterisation.

2.5.3.3 GRASS- RaPlaT portal integration

All features of GRASS- RaPlaT have been implemented in our web portal. The user is able to choose between 4 options. First is the calculation of coverage second is the calculation of interference region third is signal power at specified locations of the receivers and fourth is the unknown transmitter localization. The web portal supports placing transmitters anywhere on the map by double clicking on the map. The nodes that are integrated on the map are actually nodes mounted on the light pols in the city of Logatec. Each node that is marked on the map can be a receiver or a transmitter depends on the demands of the simulation.

The unknown transmitter localization is currently supported for a predefined and pre calculated example however it is going to be extended to support real measurements. Through the portal we can change the number of receivers and observe how it effects the localization result. The real measurements from the testbed where unknown node is transmitting and at least three nodes are receiving are stored in the database and used as an input for GRASS- RaPlaT transmitter localization module. When the unknown node is located it is placed on the appropriate place on the map. Figure 49 is showing the usage of LOG-a-TEC portal for coverage calculation.

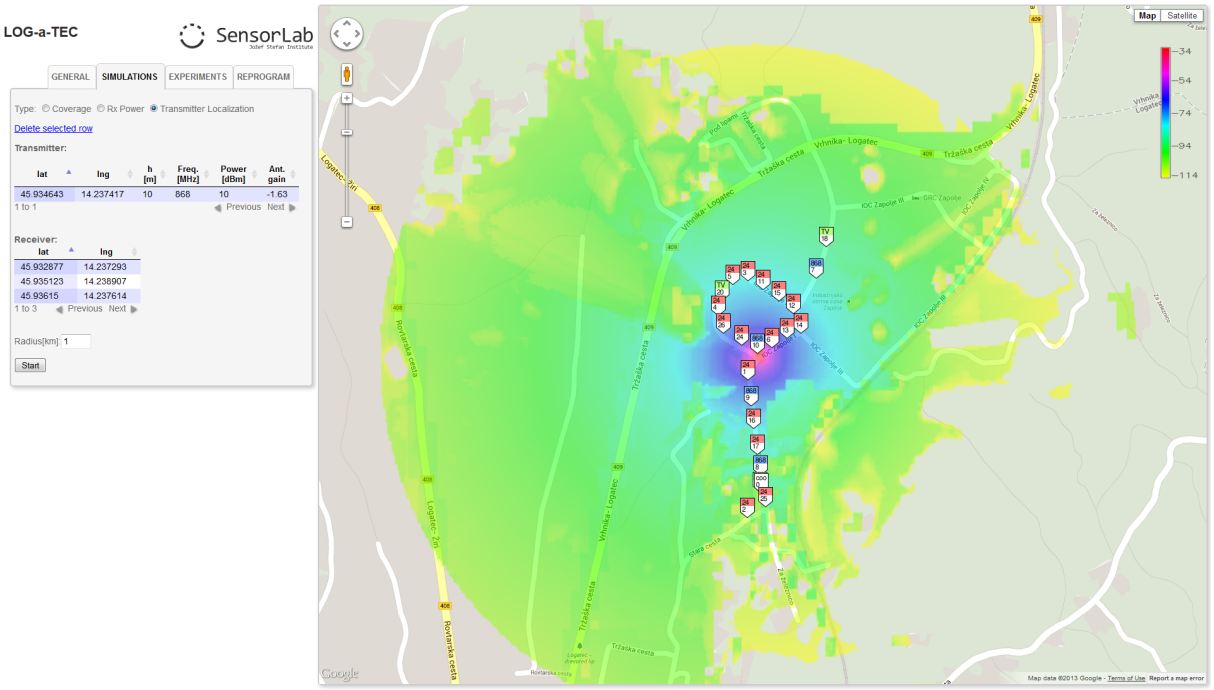


Figure 49 LOG-a-TEC Portal

2.5.4 Improvements on w-iLab.t

2.5.4.1 Improvement for USRP sensing engine

In this section, we focus on the improvement of the sensing efficiency of the USRP sensing engine. First, we introduce the background and motivation for such an improvement, then the details of the implementation are explained, finally we also demonstrate the flexibility and benefit via the list of configuration options.

2.5.4.1.1 What is sensing efficiency and why it is important

One of the most crucial aspects of sensing engine is its efficiency. Discontinuity in spectrum sensing often leads to inaccurate assessment and missed detection of interference. Spectrum sensing generally consists of two phases: the sampling phase, in which raw samples are collected from the air; the processing phase, in which buffered samples are processed for spectrum analysis.

Depending on the processing speed, the processing phase can partially or completely happen in parallel with the sampling phase. The time used for collecting samples from the air is referred to as the sampling time, while the time required by the processing phase in addition to the sampling time is referred to as the processing time. The sensing efficiency is then defined as the ratio of the sampling time and the summation of the sampling time and the processing time. During the processing time, the sampling of the wireless medium is put onto hold, which means the sensing engine is “blind”. It is possible that a number of transient signals are missed during this period, as illustrated in Figure 50. In this figure, one can see in the upper part, where FFT processing time is longer than sampling time, discontinuous sampling and missed transient signal. In the lower part, the analyzer is capable of detecting transient signal thanks to continuous spectrum sensing. The time interval when the sampling activity is put onto hold is referred to as the blind time.

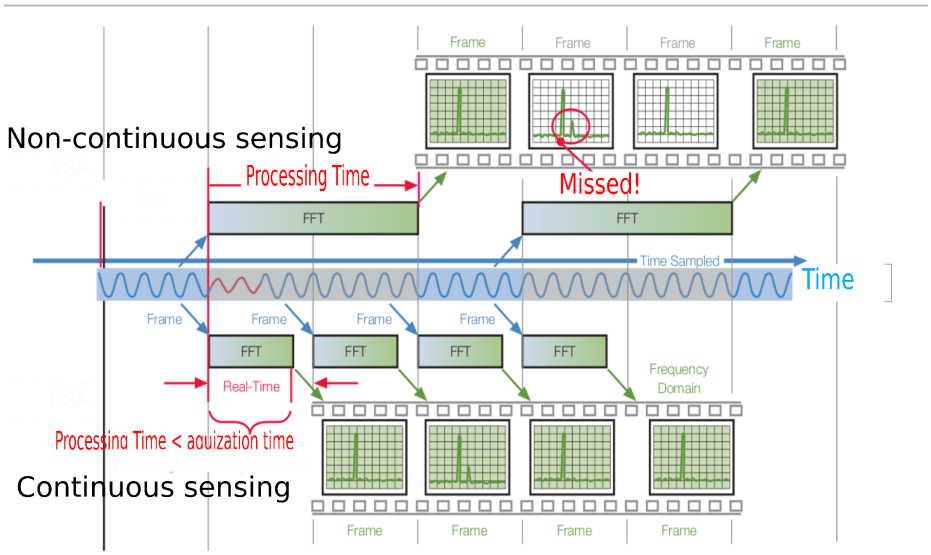


Figure 50 Continuous and non-continuous sensing. (This figure is adapted from [14].)

Ideally, the blind time should be reduced to zero, meaning 100% sensing efficiency to achieve seamless detection. There are various sensing devices on today's market. Solutions such as spectrum analyzers are capable of scanning a wide spectrum range, but are not dedicated for channel assessment and extremely costly. For instance, a spectrum analyzer usually cannot do continuous recording for a longer time period than a few seconds, and the recorded spectrum need to have high frequency resolution for visualization purpose. However, the raw spectrum information still requires further processing to obtain the energy for specified channels. On the other hand, low cost solutions are trimmed for simple and steady recording, but lack the flexibility and required performance. For instance, they are not able to achieve seamless spectrum sensing, and usually have non-configurable frequency span and resolution bandwidth.

After realizing the importance of sensing efficiency, and the lack of high efficiency in today's main stream spectrum analysis devices, we decide to rebuild the USRP sensing engine into ***an alternative*** sensing solution — a ***sensing engine*** with high efficiency and flexibility**.**

2.5.4.1.2 The software architecture

As stated previously, it is important that the processing time is shorter than the sample acquisition time in order to achieve continuous spectrum sensing. When no parallelism is present, the sample acquisition alternates with the processing phase, and as such the blind time of the sensing engine is equal to the processing time (as illustrated in plot (a) of Figure 51).

When pipelining between sample acquisition and processing is introduced, after the first batch of samples arrived, sampling the either and processing the samples obtained in the previous time frame are happening in parallel. This is the first level of parallelism. The blind time is equal to the original processing time minus the sampling time, as shown in plot (b) of Figure 51.

To further reduce the processing time, we seek to add parallel processing within the processing phase itself. The processing phase consists of splitting samples into small frames, applying FFT operation on all frames sequentially, and combining all the FFT result in one way or another. As FFT is a highly computational demanding operation, instead of having one single FFT core working sequentially, we utilize multiple FFT cores to work simultaneously: the incoming samples are divided among the multiple FFT cores for processing. Once the samples have been received, the FFT cores work independently from each other, hence ideal for parallelism. This is where the second level parallelism is introduced. We illustrate the case of two FFT cores working in parallel in the plot (c) of Figure 51. More FFT cores could be added if it is necessary to achieve continuous spectrum sensing.

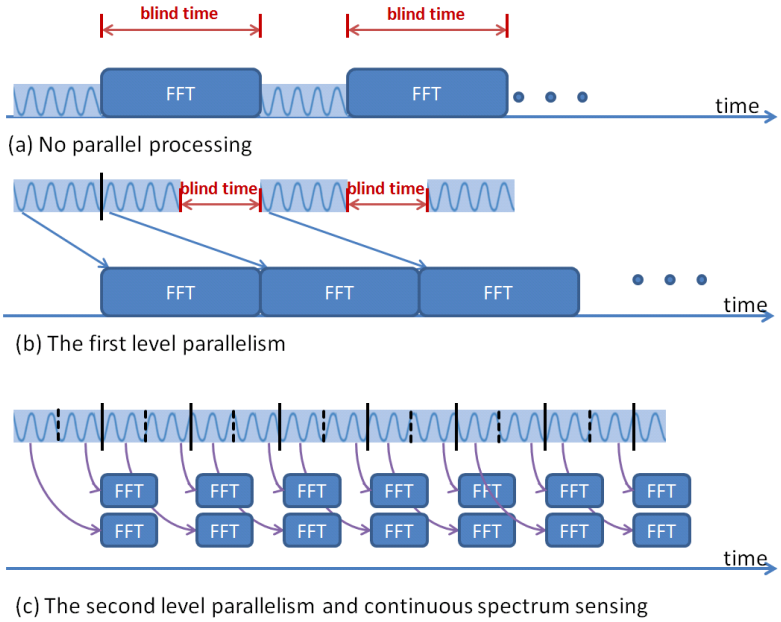


Figure 51 Parallel processing for seamless spectrum sensing

The sensing engine software relies on multi-threading to achieve parallel processing. There are two main threads running at any moment, one thread is responsible for collecting samples from the USRP (referred to as the sample-collecting thread), the other thread is responsible for processing the samples (referred to as the sample-processing thread). The sample-processing process again generates several sub threads to process the incoming samples in parallel. The sample processing in our solution calculates the FFT based power spectrum density (PSD) and the energy for specified channels. Once all sub threads finish processing, they terminate and the original sample-processing thread outputs the result to either a local file or the standard output. To simplify the collection of measurements in the wilab.t testbed, the measurements are first printed to the standard output and then piped to a predefined database using an OMF wrapper. The general structure is illustrated in Figure 52.

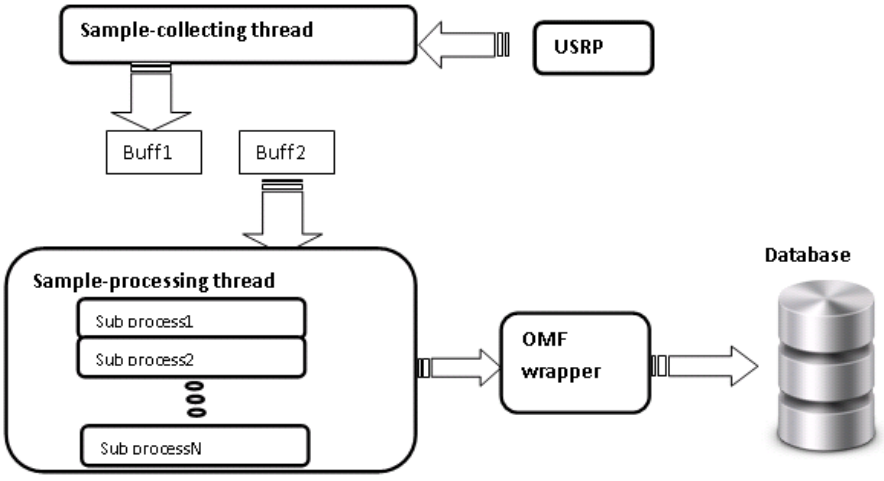


Figure 52 High level description of the software for seamless spectrum sensing

To achieve true parallel pipelining, two buffers are used to collect samples from the USRP. At any given moment, when one sample-collecting thread is writing to one buffer, the sample-processing thread will be reading from the other buffer. Therefore, once the first batch of samples have arrived, the two main threads work fully in parallel. To ensure that the two main threads do not read and write to the same buffer at the same time, the sample-processing thread needs to work faster than the sample-collecting thread. Hence within the sample-processing thread, several sub threads are created to accelerate the processing. The number of threads that should be used to achieve best efficiency depends on how many samples the buffer contains and the FFT size. During our experiments, 8 processing-threads are sufficient to support a sample-collecting thread at the highest sample rate of the USRP (25Msps). However, for configurations in which only a small amount of samples is collected, the overhead of creating multiple threads outweighs its processing benefit, hence the sample-processing thread can no longer follow the sample-collecting thread. When this happens, the software detects the overflow of samples and returns an error message.

2.5.4.1.3 Configurations and important features

The sensing engine software can be configured using various options, which are described in detail in this section.

Continuous FFT mode vs Swept FFT mode

First of all, the sensing engine can be used in two modes: the continuous FFT mode and the swept FFT mode. For the continuous FFT mode, the USRP front-end stays at the same frequency and continuously samples the wireless medium. Similar to spectrum analyzers, users can control both the center frequency and the sample rate in order to define the spectrum range. For the swept FFT mode, the USRP will always collect samples at its maximum sample rate of 25 Msps. The samples collected at a specific RF center frequency are called a block, while the complete measurement across several RF center frequencies is called one sweep. Between two adjacent blocks, the center frequency is incremented by a step of 20 MHz. Users can specify the center frequency of the beginning block, and how many blocks one sweep should contain. As such, by adding the swept FFT mode, the frequency span is no longer limited by the sample rate.

Measurement types

The sensing engine can be configured to perform different types of measurement.

(i) The sensing engine can measure the PSD in the required frequency range, and thus calculates the amount of energy detected in each specified channel. This is referred as the PSD measurement. The PSD measurement has three variants: averaging, maxhold and minhold. Typically, the number of samples per buffer is a lot larger than the FFT size, hence each buffer contains many FFT frames. For the PSD measurements, the software does either averaging, max hold or min hold across different FFT frames, and the final FFT result is used for the power integration for the requested channels. The maxhold mode is useful to detect the signal's presence, while minhold mode can be used for estimating the noise floor.

(ii) For the continuous FFT mode, the sensing engine can also measure how much portion of time the energy of the specified channels is above a certain threshold. This is referred as the duty cycle measurement. To realize this function, the software investigates a particular channel and counts how many times its energy is above a threshold, and then divide this number by the total number of FFT frames in the buffer. When the appropriate threshold is selected (a value that is slightly above the noise floor), the duty cycle mode can be a powerful tool to detect transient signals.

Sensing efficiency

Recall that the sensing efficiency is defined as the ratio of the sampling time and the summation of sampling time and the additional processing time. In our case, the processing phase happens entirely in parallel with the sampling phase. Hence no additional time is required by the processing phase. For the continuous FFT mode, the sensing efficiency is always 100%, since the USRP never stops sampling. For the swept FFT mode, the sampling phase must be interrupted for channel switching, which is the only cause for time loss. Hence the sensing efficiency for swept FFT mode is defined as:

$$\gamma = \frac{SamplingTime}{SamplingTime + ChannelSwitchingTime}$$

Unfortunately, channel switching of the USRP is more complicated than only tuning the radio front-end's center frequency. The host machine needs to communicate with the embedded processor on the USRP over the Ethernet interface. The exact handling of channel switching depends on the firmware on the embedded processor and the driver of the host machine.

To measure the channel switching time, Wireshark was used to record the packets between the USRP and the host machine. All configuration packets have a short packet length, while the packets containing IQ samples are typically 1514 bytes long. By using a packet length based filter, only the configuration packets used for channel switching and streaming commands can be displayed. Based on this output, Wireshark can generate an IO graph, plotting the packet/ms vs the time, as shown in Figure 53. Only packets with length smaller than 1514 are displayed. The y axis is the packet rate and the x axis is the time in ms accuracy.

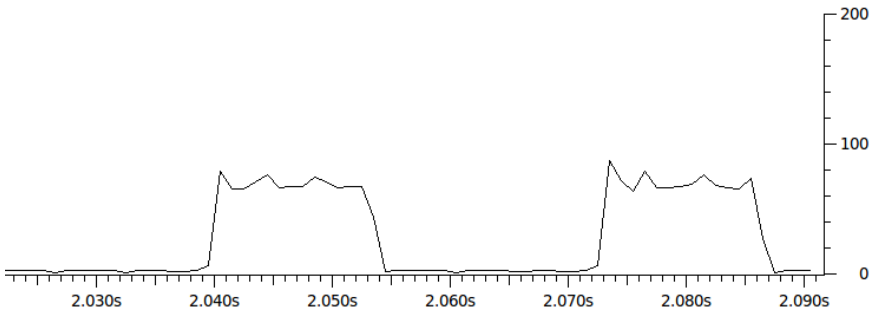


Figure 53 Wireshark IO graph derived from a packet trace between the USRP and the host machine.

This graph gives an indication on how much time is spent on sampling and how much time is spent on channel configuration. The sampling time is directly related to the requested number of samples by one stream command. This is defined by the option "—spb" in the sensing software, standing for sample per buffer. The packet trace shown in Figure 53 is generated with 524288 samples per buffer, the sampling time for each block is 524288/25Msps = 21 ms, this result corresponds with Figure 53. The configuration time for channel switching cannot be influenced by software options. It currently requires about 19 ms to switch a channel for the USRP. The channel switching time is a hardware and driver issue, it could be reduced by improving the driver and firmware.

Up till now, we have identified the channel switching time, the sensing efficiency in the swept FFT mode can be calculated by the above equation. Increasing the sampling time is an effective way to improve the sensing efficiency. When configuring the sensing engine with "spb" equal to 4194304, the sampling time is $\frac{4194304}{25000}\text{KHz} = 168\,\text{ms}$, the sensing efficiency is $\frac{168\text{ms}}{168+19}\text{ms} = 89.8\%$. With 5 blocks per sweep, the sensing engine can cover 100 MHz bandwidth and produce 1 sweep per second. Note that this configuration is very similar to the measurement capabilities of Airmagnet, however the sensing efficiency of Airmagnet is only 15% [15].

Channel Configuration

It is important to let the sensing engine know which channels to measure. Four configuration options are used to complete this target:

- numofchannel: specifies the number of channels to be measured
- firstchannel: the center frequency of the first channel
- channelwidth: the bandwidth of each channel
- channeloffset: the difference between adjacent channels center frequency

At this moment, software only allows to specify channels that are uniformly spaced and with identical bandwidth. This format is flexible enough to describe the channel specifications of the most popular wireless standard. As an example, to measure the 13 channels of WiFi in the 2.4GHz range, the following settings are used:

"--numofchannels 13 --firstchannel 2412000000 --channelwidth 22000000 --channeloffset 5000000"

The above options tell the sensing engine to measure 13 channels, with the first channel starting at 2412 MHz, each channel is 22 MHz wide, the center of all the channels are 5 MHz apart. This feature makes it easy to conduct measurements for different technologies.

Output format

The output of the sensing engine contains the following components:

- timestamp: a unix timestamp in microsecond precision
- usrpid: the id of the USRP used to collect samples
- energy or duty cycle array: an array that contains either the energy (in dBm) or duty cycle (in percentage) for all channels of interest

In contrast to the raw spectrum measurements from spectrum analyzers and some USB based devices, the output format can directly be used for channel assessment.

Resolution bandwidth and FFT size

Usually, the resolution bandwidth (RBW) of the FFT based spectrum analyzer is calculated as the ratio of sample rate over FFT size. Similar to the Tektronix RSA analyzer, users can also directly specify the FFT size and sample rate independently.

The final frequency resolution that is obtained is defined by the number of channels and the channel's bandwidth. So the RBW does not directly rely on the FFT size. However, it is still necessary for the underlying RBW of FFT to be sufficiently smaller than the interested channel bandwidth, otherwise the power integration for the specified channel will be less accurate.

2.5.4.2 Extension with new wireless technologies

iMinds' w-iLab.t has been extended with cellular technologie by the addition of 4 femtocells. Two of these are commercial 3G femtocells (UMTS, HSPA), while the other two are pre-commercial 4G femtocells (LTE). They have been purchased from ip.access, a well-know reseller of small cells. iMinds has purchased an academic license for a software tool to emulate the core network (Evolved Packet Core, EPC) which interconnects these femtocells Unfortunately, this license only allows national research and cannot be used within an international context. Therefore, the available EPC emulation cannot be used within the CREW context. To mitigate this problem, iMinds is further exploring other possibilities to have EPC emulation available for international research at its testbed.

2.5.5 Additional Iris usability improvements*

TCD has focused augmenting the usability of the Iris software defined radio package that underpins its testbed. This work has primary taken the form of the restricting and streamlining of testbed facility to support sustainable future use. That is, TCD has applied the knowledge gained from prior years of operation within the CREW project to refine their facility to better serve users. This refinement has taken the form of a repackaging of existing equipment and the deployment of an improved, cloud-based management system. Each of these actions has resulted in directly enhancing the experience of facility users.

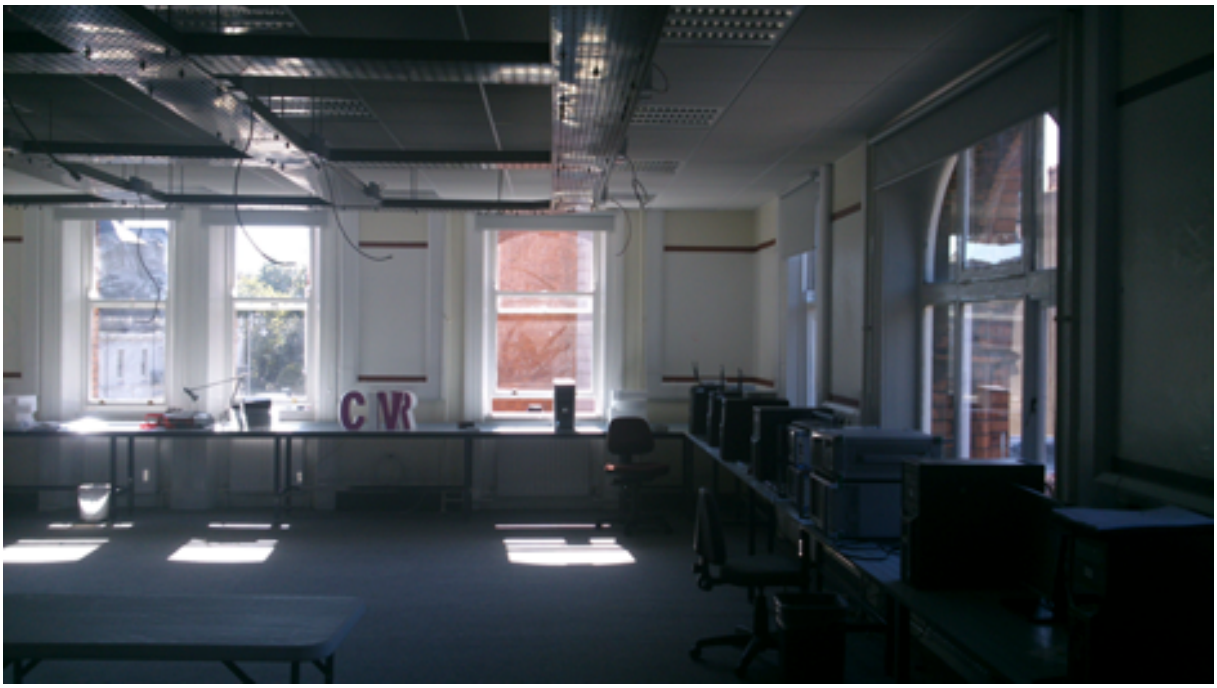


Figure 54 Ceiling mounting system in TCD testbed.

Through repackaging of existing equipment, TCD is able to tailor its facility to user requests. This repackaging concerns both the logical organization of testbed resources as well as the physical equipment. In terms of the logical organization of resources, TCD has conceptually packaged resources into experimentation units, consisting of a computational element, a hardware radio front-end component, and the Iris software define radio package. Each experimentation unit represents the minimum set of resources required by a user to construct a radio element in TCD's facility. Due to the capabilities of Iris, these experimental units effectively represent the potential to realize any arbitrary radio system that a user may require. Organizing resources in this manner provides users a clear picture of the capabilities offered to them by TCD's facility and is important for subsequent testbed improvements. In terms of the physical equipment, TCD has installed a ceiling mounting system for 16 radio front end elements (USRPs) as can be seen in Figure 54. These elements represent fixed resources, operating in a clear environment, dedicated to supporting remote experimentation. These ceiling mounted elements are isolated from tampering, underpinning stable experimentation units. Desktop mounted USRPs are still available for on-site configuration, as has previously been the case, but the new ceiling mounting system provides a more consistent experience for users.

TCD has deployed a cloud based management system on its testbed. Under this paradigm experimentation units consist of a virtual computational platform loaded with the Iris software defined radio package and connected to dedicated radio hardware. This approach of combining virtualized computational resources, a highly configurable (and reconfigurable) software-defined radio package, and widely flexible radio frontend hardware provides users an extremely customizable testbed, suitable for conducting advanced wireless research across a broad span of the spectrum. The management system employed automatically deploys individualized experimentation environments,

configured to connect to radio hardware, as needed by the users. Since these environments are virtualized, users are free to completely individualize their computational platform, including configuration of global libraries or loading of additional software, enhancing their research by avoiding the lack of control that plague even the most advanced share resource systems due to the shared software components in the underlying operating system. Furthermore, each virtual environment is isolated from all others, simply sharing a common starting point. This allows users to develop experiments in Iris without concern of interference with any other users. Users may save their configurations, including all details of the computational system, for later use, at any point. User environments are saved during the course of a project, even if a user is not active on the testbed. The result of cloud based management that each user gets the experience of a personal testbed, yet the hardware resources are shared among many separate users and projects.

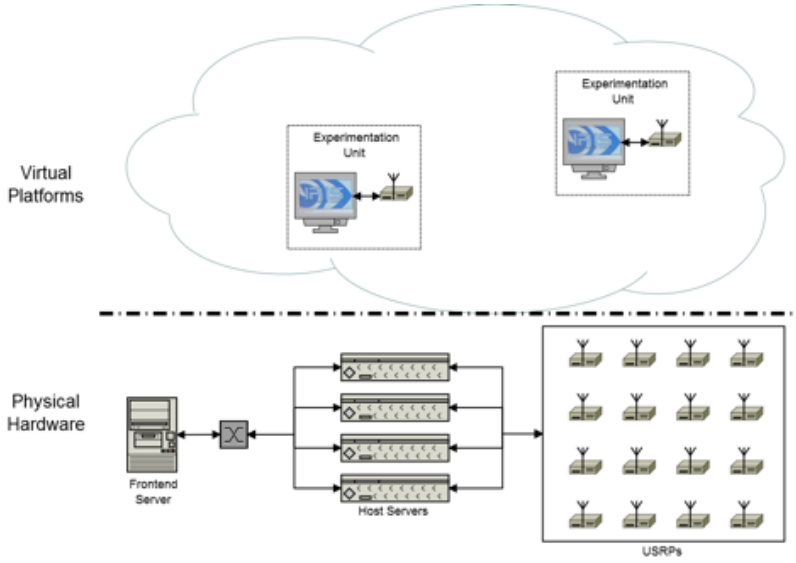


Figure 55 Virtualized testbed architecture.

Figure 55 displays the architecture of TCD's virtualized testbed. In this paradigm, underlying hardware is composed into experimentation units. Each experimentation unit initially provides users with an virtual computer running Ubuntu 14.04 and loaded with Iris. An array of servers, referred to as host servers, provide the computational power to run these experimentation units. Each virtual machine is connected to a USRP mounted on our ceiling grid within our dedicated testing space. These connections are made through dedicated network ports; i.e., each experimentation unit is provided with its own 10 Gbps connection to a USRP. The frontend server coordinates and controls virtual machines, handling the deployment of computational environments and their connection to USRPs. The frontend server also provides users access to experimentation units, currently through either SSH. Centralized control also allows the frontend to handle the scheduling of user access.

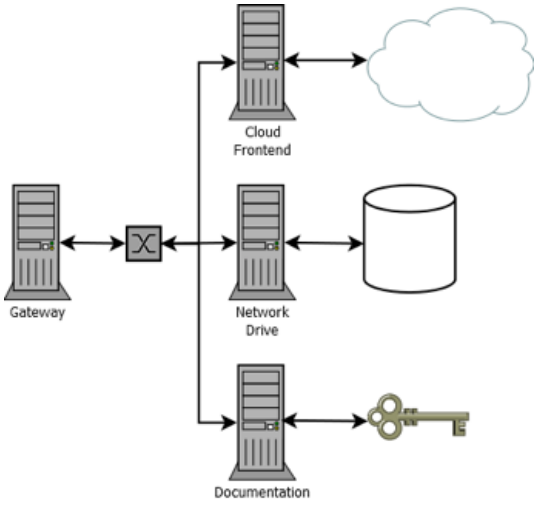


Figure 56 The virtualization management system.

Figure 56 displays the support structure for the virtualization management system. A gateway machine provides a central point for remote testbed access, data access, and documentation. The dedicated documentation server provides users with a wealth of information on how to use the testbed. The data server holds user data separately from virtual machines, providing a more permanent storage location and allowing the collection of information from several virtual machines. This support system allows users to take full advantage of the cloud base management structure.

2.5.6 Finalize and test the redesigned SNE-ISMTV-UHF receiver*

2.5.6.1 Introduction*

Sensor nodes deployed to LOG-a-TEC cognitive radio testbeds in Year 1 of the CREW project were equipped with the SNE-ISMTV expansion (see CREW deliverable D3.3). SNE-ISMTV was a multi-purpose circuit board that supported several combinations of versatile integrated radio front-ends. The radio hardware was dedicated to experimentation with spectrum sensing and advanced communication protocols for sensor networks. A separate radio interface board, SNR-MOD, was used for wireless management of sensor nodes.

Versions of SNE-ISMTV used in LOG-a-TEC included a 2.4 GHz CC2500 transceiver (SNE-ISMTV-2400), sub-1 GHz CC1101 transceiver (SNE-ISMTV-868) and a custom designed VHF/UHF receiver based on a TDA18219HN UHF silicon tuner (SNE-ISMTV-UHF)

SNE-ISMTV-UHF receiver was successfully used in a number of experiments involving detection of wireless microphones and DVB-T transmissions (see for instance “Avoiding interference through spectrum sensing and propagation modelling” internal JSI experiment described in CREW deliverable D6.3 and the CREW-TV OC2 experiment).

The original design proved to be reasonably robust. During the operation of the testbed none of the 8 SNE-ISMTV-UHF devices deployed in an out-door environment failed. Also, the small size of the receiver proved to be convenient when performing mobile measurements. The absolute power measurements it provided were reasonably accurate and the internal receiver noise floor proved to be excellent.

However the fact that its hardware design only allowed for energy detection meant that SNE-ISMTV-UHF was not capable of supporting state of the art experiments with advanced spectrum sensing methods. Even with energy detection the receiver had a disadvantage of a relatively high minimum channel bandwidth of 1.7 MHz compared to the typical 200 kHz bandwidth of PMSE transmissions in the TVWS.

Because of this we decided to design an updated version of the receiver in Year 4. With the update we indent to address some of the most common comments from internal and external experimenters using our testbeds in the TV whitespaces. We call the new design SNE-ESHTER, short for "embedded sensing hardware for TVWS experimental radio".

2.5.6.1 Discussion of changes*

This section discusses the most influential comments received from experimenters and specific changes to the receiver's design that were made to address them. Figure 57 and Figure 58 show block diagrams of the old and new designs respectively.

In addition to comments below, a large number of smaller changes have also been applied to the design to improve electro-magnetic interference, make the programming and debugging of sensor nodes using SNE-ESHTER easier and to help the experimenter understand how the receiver is functioning.

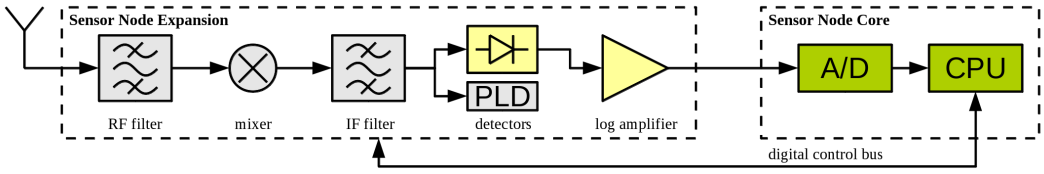


Figure 57: SNE-ISMTV-UHF block diagram.

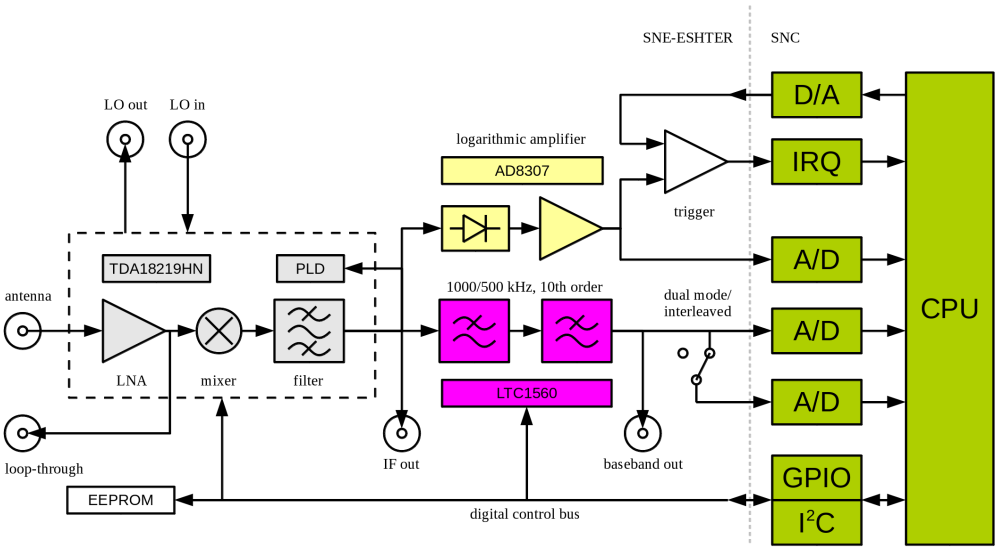


Figure 58: SNE-ESHTER block diagram.

*2.5.6.1.1 Possibility of capturing signal traces**

In SNE-ISMTV-UHF the baseband signal from the tuner was routed through an analog detector and logarithmic amplifier and then sampled at a low sampling rate (147 kHz) with an 12-bit analog-to-digital converter (ADC) integrated into the sensor node core's CPU.

This design made it possible to measure baseband signal amplitude with an excellent dynamic range and sensitivity beyond what would be possible with a 12-bit ADC. It also made it possible to measure the signal amplitude without excessively loading the relatively low-powered sensor node core CPU as only around a 100 ADC samples of the analog detector output were typically needed to calculate the

amplitude to the desired accuracy. On the other hand the use of the ADC already present on the sensor node core made it possible to keep a small physical size and low component count. It also removed the need for high-speed digital busses on the receiver board, which helped reduce the electro-magnetic interference (EMI) from the digital part of the circuit.

The inclusion of a non-linear component (the analog detector) in the signal path however meant that it was not possible to recover the original signal waveform on the CPU side. Even if the sampling rate of the ADC would be increased, only the square function of the signal ($x[n]^2$) could be approximated. Spectrum sensing methods other than energy detection however require access to the unmodified signal samples (x[n]) and were hence not possible to implement with this hardware.

To address this shortcoming, we added an additional signal path to the receiver that skips the detector and logarithmic amplifier and routes the baseband signal from the tuner to two ADC converters on the core board. The sampling rate on the core board has been increased to 1 MHz. This allows the capture of signal at 1 Msample/s when using a single ADC and 2 Msample/s when using two ADC in a fast dual-interleaved mode. It should be noted however that while the receiver hardware and the ADC are capable of sustaining a 2 Msample/s stream, the CPU is not capable of processing it in real-time. Therefore we predict that the typical use case for this mode of operation will be to a capture a buffer of signal samples, followed by offline signal processing. The amount of RAM on the CPU allows for a single of capture length of up to 25 ksamples (12.5 ms with 2Msample/s).

To make it possible to directly sample the baseband signal with ADC converters on the core board, additional analog signal conditioning components were required. Channel filters integrated into the TDA18219HN tuner can only limit the baseband bandwidth down to 1.7 MHz. To prevent aliasing with 1 and 2 Msample/s ADC, an additional analog, 10th order elliptic low-pass filter was inserted between the tuner and the ADC. The filter bandwidth can be set from software to 1 MHz or 500 kHz, allowing for optimal signal bandwidth in both 1 Msample/s and 2 Msample/s modes. Addition of the analog filter required a balanced-to-unbalanced signal conversion and an additional on-board low-noise 10V power supply. These additions were responsible for most of the increase in circuit component count compared to the old design.

In addition to enabling experimentation with advanced spectrum sensing methods, the new direct signal path in SNE-ESHTER can also be used for energy detection with a lower resolution bandwidth than what was possible with SNE-ISMTV-UHF. The additional filter lowers the channel bandwidth to 1 MHz or 500 kHz. By calculating the signal energy in software from an appropriate array of baseband samples it is possible to also perform energy detection at 1 MHz and 500 kHz bandwidths. This way SNE-ESHTER allows recording swept-tuned spectrograms with a higher frequency resolution than those possible with SNE-ISMTV-UHF. Hence this change addressed another common request from experimenters using SNE-ISMTV-UHF. It should be noted though that with the absence of a logarithmic amplifier, the dynamic range of this mode of energy detection is limited by the 12-bit ADC.

*2.5.6.1.2 Possibility of two coherent signal chains**

Some spectrum sensing methods require signal capture from two antennas, possibly with coherent sampling. To make this possible with SNE-ESHTER, two features were added to the new design: dual-receiver operation and external clock reference.

The assignment of pins on the standard stackable VESNA expansion connector was changed in a way that allows for two SNE-ESHTER boards to be stacked on top of each other, as shown on Figure 59. In such a configuration, we refer to the top board as “slave” and the bottom board as “master” board. Routing of pins between the top and bottom expansion connectors on SNE-ESHTER allows for a single circuit board design to act as both a master and a slave board, eliminating the need for two separate designs for the two roles.

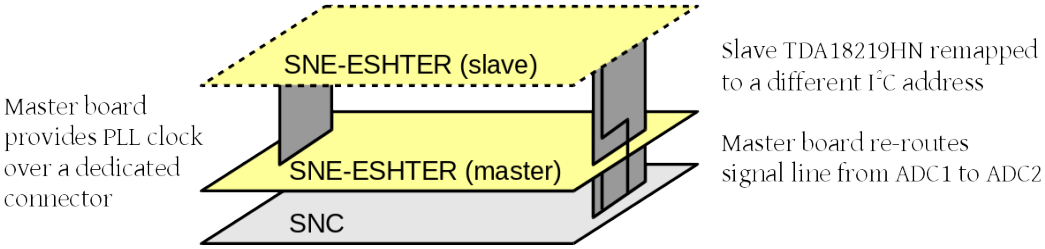


Figure 59: Stacking two SNE-ESHTER boards

In dual-receiver operation, both SNE-ESHTER boards share the same control signals and behave identically. The only difference is that addresses on the I^2C are remapped on the slave board. This allows software to control both tuners independently. For example, boards can be set to the same central frequency to sense the same channels using two antennas or, by using the loop-through functionality on the master board, two channels from the same antenna.

The baseband from the slave board is routed to the secondary ADC on the sensor node, while the primary ADC samples the baseband from the master board. In this case the two converters operate in the dual-simultaneous mode, coherently sampling both signals. Only 1 Msample/s sample rate is supported in the dual-receiver configuration.

The UHF tuner on SNE-ISMTV-UHF used an on-board 16 MHz crystal oscillator as a reference clock from which it derived the LO frequency using an integrated PLL. In SNE-ESHTER we retained this functionality and also added the possibility of running the tuner from an external reference clock instead of the on-board oscillator. To achieve this, the SNE-ESHTER board can be fitted with a clock input connector on the bottom of the board and appropriate clock buffers. In this case the crystal oscillator on the board is not used and the LO frequency is derived from the external clock signal.

In dual-receiver operation, a slave board with the clock input connector can be used. In that case the clock output connector on the top of the master board and clock input connector on the slave board connect. Tuners on both boards hence run synchronously from the same oscillator. This eliminates unpredictable phase shifts between the two baseband signals that would be caused by two free-running local oscillators.

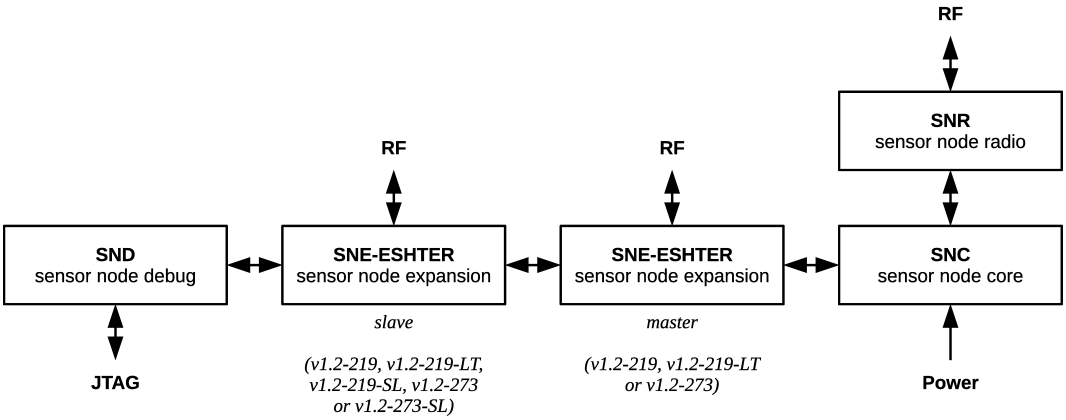


Figure 60: Block diagram of a VESNA sensor node with all possible expansion boards.

*2.5.6.1.3 Enable faster responses to signal power changes**

One of the comments regarding SNE-ISMTV-UHF was that in order for the CPU to respond to a change in the signal level, it must continuously perform analog-to-digital conversions. ADC sampling is relatively slow compared to the core CPU clock. In some use cases timing is critical, for example in carrier-sense media access protocols (CS-MAC). For those cases it would be preferable if the radio

frontend could independently raise an interrupt request on the sensor node code CPU when a channel becomes vacant.

To address these comments a trigger subsystem was added to SNE-ESHTER. SNE-ESHTER retains the signal path through the analog detector and logarithmic amplifier that was present in SNE-ISMTV-UHF. A comparator circuit was added to that signal path that compares the signal level to a threshold value set by the CPU. When the detected signal level in the tuned channel passes the threshold, an interrupt request is sent to the CPU. The interrupt can be triggered on both the falling edge (i.e. a channel has become vacant) and rising edge (a channel has become occupied). In the latter case, SNE-ESHTER can be switched to signal sampling mode with minimal latency, which allows for accurate triggering of signal sample collection when a sufficient signal level is detected. This capability can be useful for example to trigger advanced signal analysis only when energy detector detected significant power present in the channel.

*2.5.6.1.4 On-board storage of serial number and calibration tables**

SNE-ISMTV contained no on-board non-volatile memory. It required that the firmware running on the sensor node core was manually programmed with correct calibration tables for the physical receiver that was installed on it. Because of this, only one average calibration table was used typically used in practice for all deployed nodes to remove the need for customized firmware images for each individual sensor node. This reduced the accuracy of measurements due to inaccurate calibration tables.

There was also no programmatic way for the software to read the serial number of the hardware. This proved problematic when tracking deployment of a large number of sensor nodes in LOG-a-TEC testbeds. Reproducibility of experiments also depended on the experimenter manually noting the serial numbers of hardware involved in the experiment.

To address these comments, we added a small amount of non-volatile EEPROM memory to SNE-ESHTER. Typically we predict that a calibration table will be written to the EEPROM once after the board has been tested. Firmware on sensor node core could then download the table from the board as needed. The specific EEPROM integrated circuit we used also provides a factory-programmed, read-only 128-bit identifier that can be used by the sensor node firmware to uniquely identify the radio that is installed.

2.5.6.2 Testing the prototypes*

15 prototypes of the initial SNE-ESHTER design have been manufactured. Due to the limited supply of the TDA18219HN tuner we opted to manufacture the first batch of SNE-ESHTER circuit boards in three variants: 5 with the TDA18219HN tuner, 5 with a similar, pin-compatible TDA18273HN tuner and 5 with only baseband components. The last variant allows mounting of other tuners. This would be necessary in case supply problems will force us to switch to yet another tuner IC in the future. So far, only the version using the TDA18219HN has been tested.

The top of the printed circuit board can be seen in Figure 61.

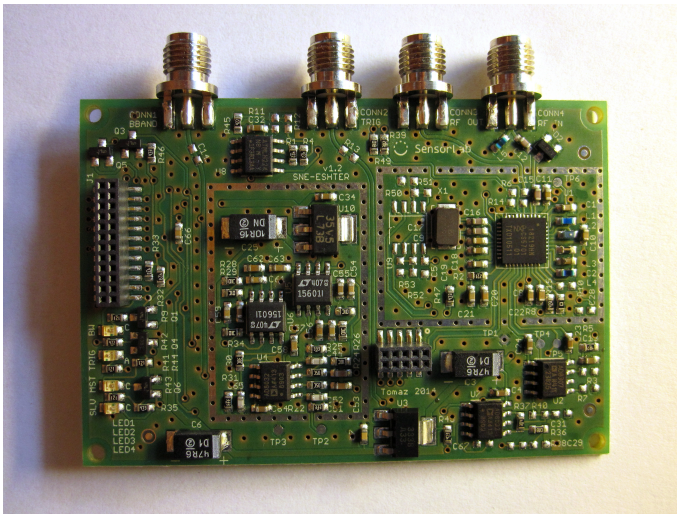


Figure 61: SNE-ESHTER printed circuit board

The prototype SNE-ESHTER boards have so far passed automated tests that were developed to test SNE-ISMTV-UHF. These tests confirmed that when using the analog detector for energy detection the capabilities of SNE-ESHTER are comparable to SNE-ISMTV-UHF. However not all tests for capabilities unique to the new design have been developed yet.

Shown in Figure 62 are measured channel filter characteristics using the 8 MHz, 1.7 MHz, 1 MHz and 0.5 MHz settings. 8 and 1.7 MHz settings are also present on SNE-ISMTV-UHF. The measured characteristics in these two cases were identical to the old design, since these characteristics are defined by the integrated channel filter in the tuner IC, which is identical in both the new and old designs. 1 and 0.5 MHz settings are due to the new elliptic baseband filter and only available on SNE-ESHTER.

Shown in Figure 63 are energy detector responses. 8 and 1.7 MHz settings use the old analog detector and logarithmic amplifier signal path with a high dynamic range. With the new 1 and 0.5 MHz settings, the energy detector is emulated in hardware. As expected, the dynamic range with these settings is considerably worse.

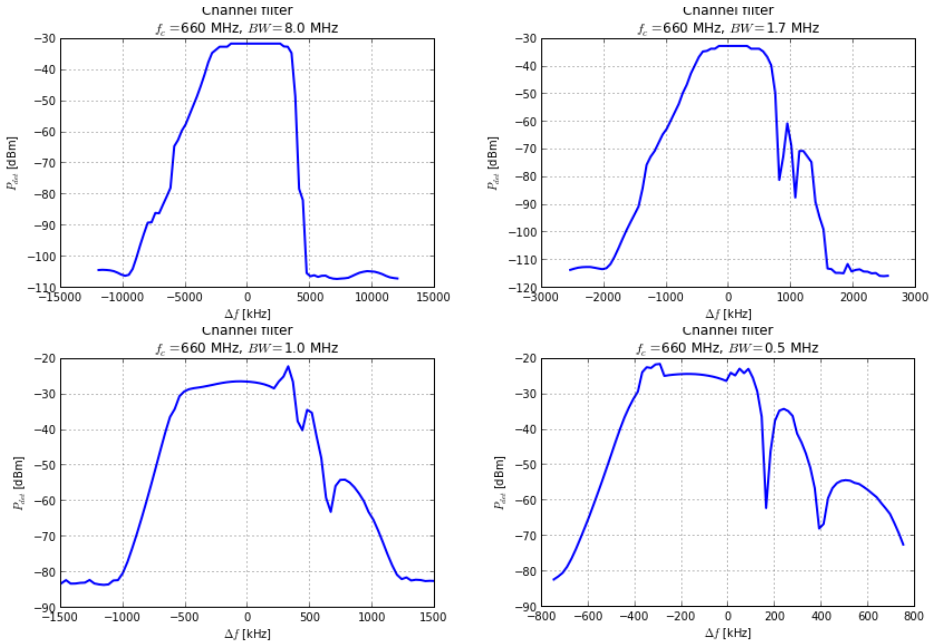


Figure 62: Measured channel filter characteristics.

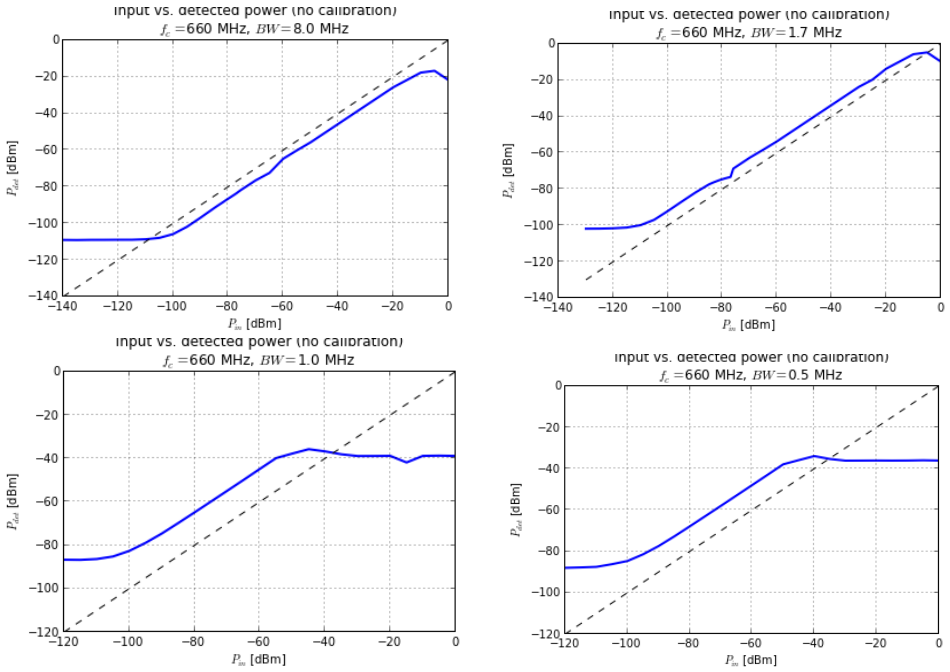


Figure 63: Measured energy detector responses with different channel filters. Graphs compare calibrated input power with detector response without calibration. An ideal response is shown with a dotted line.

Thorough testing of the new signal capture capabilities is yet to be performed. However for demonstration purposes we have created an example application running on the sensor node that compares the performance of a number of spectrum sensing methods based on signal sample covariances.

In the demonstration shown in Figure 64, SNE-ESHTER is used to capture 25 ksamples of baseband signal at a time. The signal samples are sent to the CPU for off-line processing. The CPU computes test statistics for 7 covariance-based methods as well as energy detection. Signal samples, power density spectrum of the baseband signal and all computed test statistics are displayed in real-time on a screen of laptop computer.

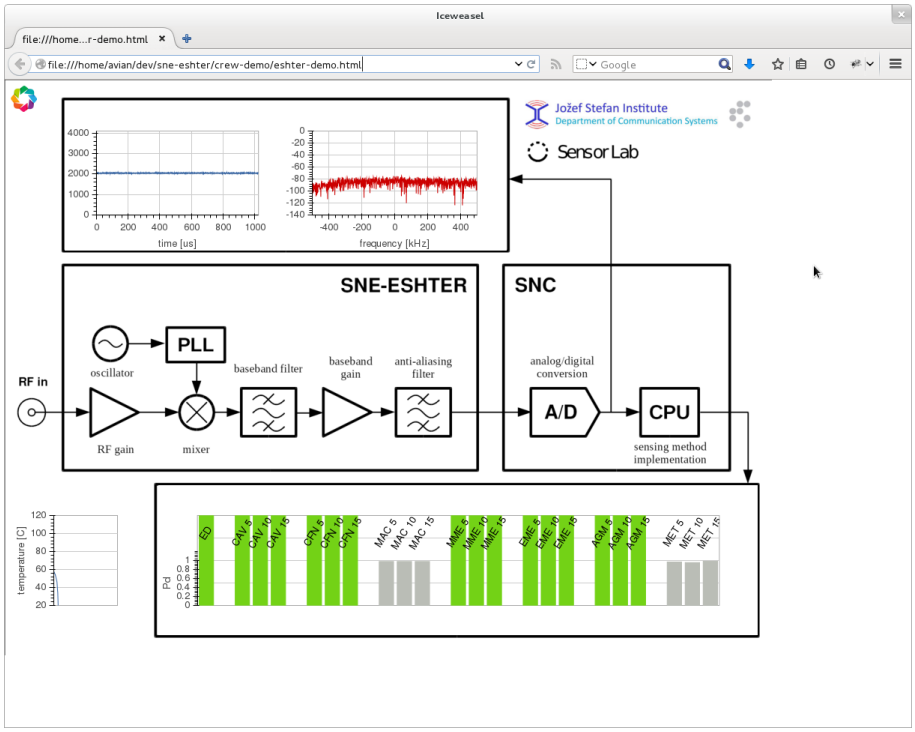


Figure 64: Demonstration of covariance-based spectrum sensing using SNE-ESHTER

2.5.6.3 Expanding the Ethernet connectivity options for VESNA*

VESNA sensor nodes in LOG-a-TEC testbeds typically participate in a management wireless mesh network. Only one node in the cluster, the coordinator, has an Ethernet interface. In such a setup the coordinator is responsible for forwarding requests for other nodes in the cluster from the Ethernet interface to the wireless mesh network and vice-versa. In fact, the coordinator is dedicated to the packet forwarding duty and does not have any experimental hardware.

Recently, a number of use cases appeared where it would be beneficial to deploy only one spectrum sensing node at a remote location. One such example was the proposed involvement of JSI in the Ofcom trials. With the current setup, a single sensor node cannot have both the spectrum sensing hardware (SNE-ISMTV or SNE-ESHTER boards) and the Ethernet interface board (SNE-WG) installed since both of these boards occupy the top expansion connector.

Figure 65: VESNA nodes with SNE-WG (top) and SNR-ETH boards (bottom)

To enable such a combination, a new circuit board with the Ethernet interface was developed. The new board, called SNR-ETH, connects to the radio connector instead of the expansion connector. In current deployments, the radio connector is occupied by a radio module that is dedicated to communication with the wireless management network. Since a single node deployed remotely does not need the mesh network connection, it was free to be used by the new Ethernet interface.

SNR-ETH board differs from the SNE-WG only in the physical shape. Both use the same Ethernet interface module. Because of this VESNA firmware needed only minimal changes to add support for the new board.

2.5.7 Improving the LOG-a-TEC wireless management network*

One of the main demands for LOG-aTEC from external experimenterd was related to the improvement of the low speed of the wireless management network through which the testbed is accessed and controlled. In order to take this into account for the new generation of the testbed that uses the same hardware for completely different embedded software, we first identified the use cases that need to be accommodated by testbeds in general and designed, implemented and performed an initial evaluation of the testbed. The full work is reported in a joint conference paper [51] with open access external experimenters from a Slovenian SME Xlab, who contributed part of the reported solution and was particularly interested in the evaluation of different (re)programming and (re)configuration approaches.

2.5.7.1 Use cases and required functionality*

We identify two generic use cases for testbeds consisting of constrained devices such as sensor nodes: the monitoring use case and the experimentation use case. the monitoring use case refers to sensor based testbeds that enable monitoring of some phenomena such as energy consumption, humidity, temperature, motion, sound, gases, etc. These kinds of testbeds are typically used by researchers to automatically acquire some data about the phenomena under study. Examples of such testbeds are SmartSantander and CitySense. The experimentation use case refers to sensor based testbeds that support the development of new communication and networking technology by enabling experimentation with new algorithms and protocols. Motelab, TWIST and LOG-a-TEC are examples of such testbeds.

In spite of this division in two major groups, from the point of view of the wireless management network, these testbeds have a set of common functionalities.

*2.5.7.1.1 Need for software upgrades**

From the perspective of software upgrades, we identify three types of required updates: OS/firmware upgrades, driver updates and application updates. OS/firmware upgrades are expected to occur when new versions of these software are released or when a major flaw is discovered and needs immediate fixing. The frequency of these upgrades is expected to be of 3-4 per year at most for both use cases. From the perspective of the size of the code to be transfered over the air to the nodes of the testbed, these upgrades tend to be large.

The driver updates are also expected to be required at most few times per year for the monitoring setupsand for most instances of the experimentation setups. However, for experimental setups that involve MAC layer experiments, the need for upgrades might be more frequent. In many cases, these upgrades can be achieved using dynamic linking, thus avoiding the need for realizing an OS/firmware upgrade. In such cases, the file sent to the nodes of the testbed is relatively small compared to the full OS/firmware image.

The expected application updates vary a lot across testbeds. The more flexible and generic the testbed is, the higher the number of expected application updates. These updates are best performed using

dynamic linking and in most cases their expected size is relatively small. Performing a full OS/firmware upload for each application tends to be uneconomical.

*2.5.7.1.2 Need for data collection**

Testbeds used as the monitoring setups require sensor measurement data collection while the ones used as the experimentation setups require the collection of the experimental results. Here we distinguish non-time critical data collection and time-critical data collection. When the data collection is time critical, the measured phenomenon or the experimental results have to be sent to the consumer within a small predefined time period from when they were produced and can also be referred to as (near-)real time data collection. This kind of data collection is encountered in sense-act type of systems that base their actuating decision on the sensed value. When the data collection is not time critical, it is typically saved locally on the node and transmitted all at once in batch mode. This collection mechanism is found in scenarios where the data is being post processed.

We also distinguish reliable and unreliable data collection. In the first case, the loss of the measurement data is undesired while is the second case loss of data is not considered a major issue.

*2.5.7.1.3 Need for remote reconfiguration and control**

Remote reconfiguration is a desired feature for both setups. For the monitoring setup, the user of the testbed might want to change the sampling rate of the sensor or might want to change the size of the buffer that stores the measurements. For the experimentation setup, the user might vary several parameters such as the frequency at which packets are sent, the number of retransmissions, transmitting power, receive channel filter bandwidth, carrier sense indicator etc. In some cases, using remote reconfiguration, the entire experiment can be remotely reconfigured. For instance, using run-time reconfiguration, the entire protocol stack can be reconfigured without flashing the node. Using a fully modular implementation such as CRime, an experiment that uses a gossip based algorithm for sending data can be easily reconfigured to use another type of algorithm.

Remote control is a desired feature for sense-act scenarios that can also appear in both setups. For instance, in a monitoring setup, a light can be dimmed in response to the sensed values of luminance and presence. In an experimentation setup, a node can be controlled to start a transmission after another node sensed a free channel.

2.5.7.2 Design, implementation and initial evaluation of the LOG-a-TEC cognitive networking testbed*

*2.5.7.2.1 Design choices **

When building a new testbed from scratch, one is faced with selecting a desired hardware platform and a supported operating system. Heterogeneous testbeds may chose several different such platforms. The use cases and functionality, as well as the considerations with respect to reprogramming, reconfiguration, speed and reliability should be taken into account when selecting the hardware and OS. For instance, with some OSes it will be impossible to support dynamic linking and/or run time reconfiguration. The resulting combination of hardware/OS selected needs to be evaluated and optimized similar to the example provided in this section.

When upgrading an already existing testbed, there may be already existing constraints on the choice of hardware and software. For instance, the starting points for the extension of the already existing LOG-a-TEC testbed was the VESNA sensor node and the ProtoStack tool. The VESNA platform is already being used in the existing testbed for spectrum sensing and cognitive radio experimentation and is the supporting block of most of our research activities. The ProtoStack tool has been developed to support modular protocol development that would enable easy experimentation with multi-hop routing algorithms using also learned link characteristic for the routing decision - thus enabling cognitive

networking experimentation. As ProtoStack relies on the Contiki OS, the extension has to use this operating system.

Starting with these constraints, answers for the following three main issues have to be found:

- How to enable two wireless - experimental and management - networks running in parallel on VESNA with Contiki (subsection 2.5.7.2.2)?
- How should experiment reconfiguration, control and software upgrade be performed (subsection 2.5.7.2.3)?
- Which transceiver should be used for the management network in order to achieve the best possible throughput (subsection 2.5.7.2.4)?

*2.5.7.2.2 Dual-stack Contiki on VESNA**

The VESNA platform had a pre-existing dual radio extension board with the following options: TI CC1101 (868 MHz), TI CC2500 (2.4 GHz), AT86RF230 (2.4 GHz) and AT86RF212 (868 MHz) transceivers. Note that the extension board can support an Atmel and a TI transceiver at the same time, however it cannot support two Atmels or two TIs at the same time. So, while the hardware set-up already supported dual stack (one for management and one for experimentation), a solution for a dual-stack OS had to be developed.

Contiki OS includes two protocol stacks, one based on uIPv6 that can be configured as 6LowPAN/uIPv6/UDP/CoAP and the second protocol stack is custom and is referred to as Rime. 6LowPAN assumes a IEEE802.15.4 compatible transceiver and since only the Atmel transceivers comply to this, the most natural decision was to consider the Atmel transceivers for the management network and the TI transceivers for the experimental network. However, in the normal release of Contiki OS the two stacks cannot run in parallel but only one at a time. This required extension/adaptation of the Contiki OS to support dual stack operation. The original Contiki OS code uses compile-time defined network layers. Some layers are used by both Rime and uIP at the same time (see framer_nullmac in Figure 66), so with the support from open access experimenters from Xlab we modified the networking code to explicitly pass information about which network stack the current packet belongs to. It should be noted that in a single stack Contiki, Rime uses 2 bytes for node network address, while uIP requires 8 bytes. To keep Rime packet small, thus maintaining the low power consumption of the Rime stack, we modified Contiki to permit different network address size for Rime and uIPv6 packets respectively.

Finally, we integrated the new, Composable Rime network stack that enables reconfigurable protocol stacks in the Contiki OS and configured the operating system to support the 6LowPAN based management network and the CRime based experimental network in parallel as depicted in Figure 66.

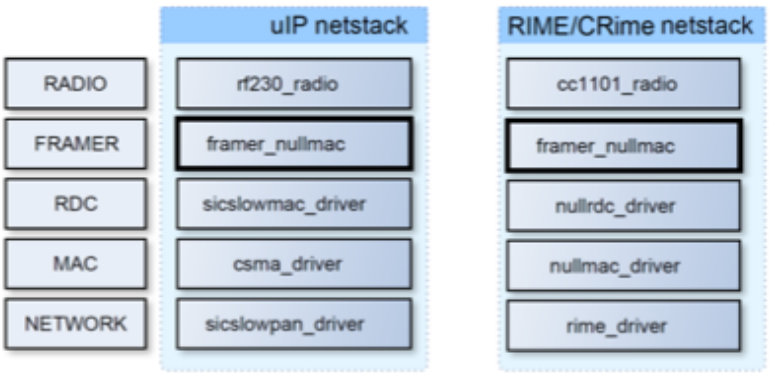


Figure 66 Dual-stack Contiki on VESNA.

*2.5.7.2.3 Software upgrades, reconfiguration and control**

Software upgrades require a bootloader running on the VESNA platform and a large image to be sent to the node over the management network. In the case under investigation, the full image of the monolithic dual stack is in the range of 150 kB as shown in Table 8. This image corresponds to a particular application (i.e. single experiment) and needs to be changed should another application be needed (i.e. flash the node).

Table 8 Transfer size.

Approach	transfer size [B]	useful size [B]	overhead [%]
monolitic dual stack image	151540	151540	0
hello-world ELF app	1752	399	88
trickle RIME ELF app	11316	3930	75
monolitic dual stack config packet	1635-7937	1635-7937	0

In order to support minor updates of drivers and applications, we used dynamic loading through the Contiki ELF (Executable and Linkable Format) loader module. Our main interest was to enable dynamically reprogrammable network stacks. In other words, we investigated the possibility of transferring new stack compositions, each representing a new experiment. This requires splitting the application into two parts.

The first part, called core, is responsible for loading the minimal Contiki OS with added ELF loader functionality. This part of the node firmware is not changed during reprogramming. It is responsible for downloading the ELF application through the management network and to dynamically link it with the core OS.

To implement it, we had to include the base Contiki image with uIPv6, TCP/UDP and CoAP, also support for:

- SD card driver and Contiki Coffee FS.
- Utility application to receive ELF file from network, and write it to a file.
- ELF loader (generic and CPU architecture specific part) to do actual ELF file relocation.
- Symbol table stores addresses and names of all core OS functions, which might be called by the ELF application.

The second part is the ELF application. The application calls functions exported by the core OS, and is compiled as a standard ELF file. When splitting the previous monolithic dual stack image into core OS (with uIP management network) and ELF application (with CRime experimental network) we have the option to leave some code parts, used only by CRime, in the core OS. In particular, we decided to leave the TI CC radio driver in the core OS. As that particular piece of code is already stable, we expect it will not require frequent updates. This resulted in about 50% smaller ELF application file.

The automatically generated symbol table contains the address and the name of each function in the core. Many of them are not even supposed to be used by the application (low level hardware initialization, static functions, ARM CMSIS library functions). Thus we minimized the core OS image size by excluding unneeded function entries from the symbol table.

We looked at the size of the file to be transferred to the nodes over the air for the very simple hello-world application and for a more complex trickle stack. The size of the hello-world ELF file is 1.7 kB while the size for the trickle ELF file is 11 kB as listed in Table 8. The trickle ELF application is small compared to the full OS image, but it still does have a significant overhead due to the ELF file metadata.

This observation led us to look at run-time reconfiguration options, where all the CRime modules are loaded on the node using a monolithic system image and then a stack composition message, which describes and configures the experiment, is sent. We used an un-optimized JSON format for the configuration message whose size can vary between 1.6 kB for a simple experiment to 8 kB for a more complex experiment as shown in Table 8. The code required for parsing the JSON and generating the experiment added additional 7.5 kB to the size of the system image.

*2.5.7.2.4 Transceiver selection**

As explained in Section Design choices *2.5.7.2.1 the selected hardware platform supports operation of the management and experimentation networks in two ISM bands, depending on the choice of transceivers to be soldered on the SNE-ISMTV boards used on VESNA, and this subsection deals with the selection of the most suitable transceiver for the management network. The candidate transceivers supporting the management network are two Atmel transceivers: AT86RF230 (2.4 GHz) and AT86RF212 (868 MHz). The first step in evaluating these transceivers was to determine the three operating regions, *effective*, *clear* and *transitional*, referring to regions with the packet success rate above 90%, below 10% and with highly varying value between these boundaries, respectively [41].

The experiment was carried out on a 55 meter corridor of a long building where several WiFi access points are also active. Figure 67 plots the three regions empirically determined in our experiments for the AT86RF230 (2.4 GHz) transceiver. The tests show that the effective region goes up to 22 m in the line of sight conditions (no obstacles on the corridor). Additionally, we performed experiments to understand how the throughput is affected by the packet rate as shown in Figure 68. The results show that rates exceeding 70 packets per second lead to packet losses.

The plots for the AT86RF212 (868 MHz) transceiver are similar, with the effective region ending at 28 m and the optimal application rate being also at 70 packets per second.

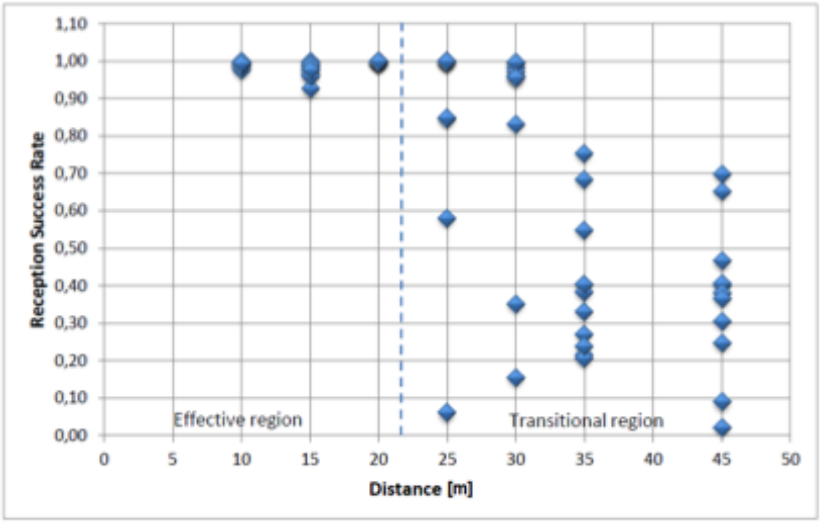


Figure 67 Reception success rate as a function of distance for the AR86RF230 (2.4 GHz) transceiver.

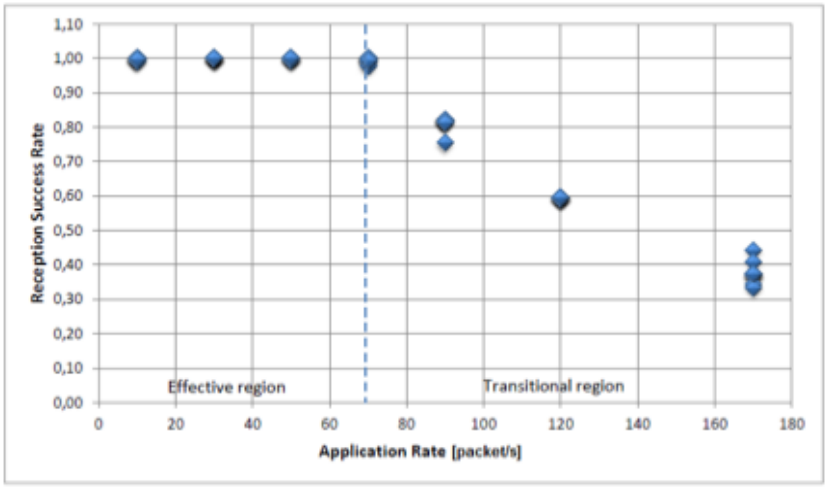


Figure 68 Reception success rate as a function of application packet rate for the AT86RF230 (2.4 GHz) transceiver.

After determining the three regions for the two transceivers under consideration, we looked at the application transfer rates and various settings that influence these. For instance, by using the header compression enabled by 6LoWPAN, the application payload can be increased thus maximizing the data rate. The CoAP client was mimicking reprogramming functionality by sending large files for reprogramming the nodes running CoAP server and data collection functionality by requesting (randomly generated) data from the nodes (see Figure 69). The CoAP clients are located on a wired IPv4/IPv6 network, then use a gateway towards the border router which has a wireless management interface for the nodes. The links between the nodes and the border router were within the effective regions, hence reliable.

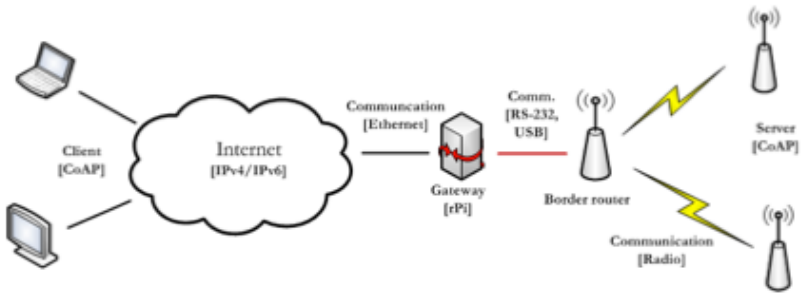


Figure 69 Experimental set-up for the initial evaluation.

We performed two different types of experiments, upload and download for two different sets of radio transceivers Atmel AT86RF212 (Table 9) and AT86RF230 (

Table 10). Each type of the experiment had five different steps and with each step we were increasing the size of the data (i.e. file) to be transmitted between 128 and 256000 bytes. To get more reliable results we repeated each step 10 times and then we calculated the average throughput value. With respect to the upload and download we performed 100 measurements per radio transceiver. From the results in Table 9 and

Table 10, it can be seen that AT86RF230 is achieving higher data throughput and the links are more stable.

In our evaluation we only considered packets that contain payload data, avoiding acknowledgments messages that are sent for each packet and that are not relevant for the application data rate. We decided to use HC01 and HC02 compression for 6LoWPAN because if the CoAP client is accessing the testbed from a different network subnet, the IPv6 address will not be fully compressed in any case.

Packet fragmentation was disabled. MAC header compression was not used, because it is supported only by the AT86RF212 MAC. As a note, there are several configurations and tunings that can be performed with such an evaluation. Configurations in the Contiki OS, the used drivers and the point from which the client is accessing the node influence the final performance of the wireless management network.

Table 9 Evaluation of AT86RF212.

Size [bytes]	Download time [s]	Upload time [s]	Download throughput [kbs]	Upload thoughput [kbs]
128	0.185	0.167	5.405	5.988
1280	1.705	1.655	5.865	6.042
12800	16.614	16.666	6.019	6.000
128000	176.324	168.800	5.671	5.924
256000	346.699	339.327	5.768	5.894

Table 10 Evaluation of AT86RF230.

Size [bytes]	Download time [s]	Upload time [s]	Download throughput [kbs]	Upload thoughput [kbs]
128	0.086	0.069	11.627	14.492
1280	0.686	0.694	14.577	14.409
12800	6.673	6.965	14.985	14.357
128000	68.191	68.899	14.664	14.513
256000	139.696	139.241	14.316	14.363

Tables Table 9 and

Table 10 are summarizing results where 64 bytes of application payload has been used. It can be seen that transferring 128000 bytes (a bit less than a full system image from Table 8), 68 seconds are needed when using AT86RF230 and 177 seconds when using AT86RF212. For the dynamic loading of a simple application or its run-time reconfiguration using a non-optimal JSON format, less than a second is needed with AT86RF230 and under 2 seconds with AT86RF212. Our initial experiments showed that the AT86RF230 transceiver is more suitable if speed is the only selection criterion. Further tests and evaluations with respect to other criteria are underway.

2.6 Collaborations with Other Projects

2.6.1 CREW-OpenLab

As described in D4.2, the CREW-Openlab collaboration results in the web tool CONCRETE. The main functionality of this tool is that it is able to calculate cross correlation of traces obtained in different experiment rounds. The correlation value is a fairly good indication for the stability of the experiment, high correlation value indicates that certain experiment is repeatable, low correlation value indicates there might be external interference or a particular trace is an outlier.

In the year, we aim to explore this tool further by a set of experiments. There are two different scenarios, performed on the w-iLab.t and NITOS testbed. For the experiment scenario on w-iLab.t, it involves a moving robot, connected to a fixed WiFi access point. The moving robot receives packet from its access point while it is moving on a predefined route. At the same time, it tracks the RSSI and the throughput. The route is carefully chosen to include an obstacle between the robot and the access point, as indicated in Figure 70. This route design gives sufficient variation in the RSSI trace.

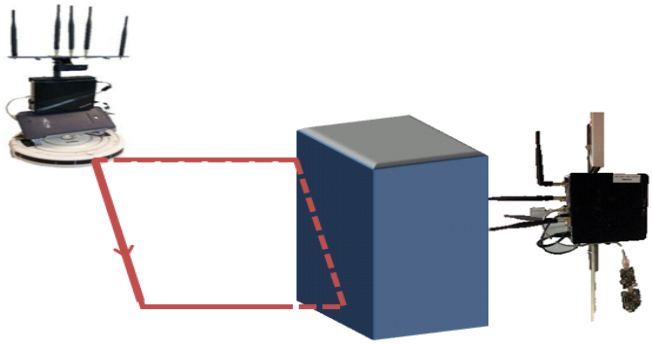


Figure 70 the route of robot in w-iLab.t experiment

The complete set of experiments is still undergoing. Therefore at this moment no comprehensive analysis is presented. But we show some initial result to illustrate how the RSSI and throughput is correlated. Figure 71 contains two typical graphs of the RSSI and throughput traces obtained during one round of experiment. The left side of Figure 71 shows the RSSI and the right side shows the throughput performance. We can see that as the robot driving close by the access point, the RSSI increases but with more fluctuation, the same trend is observed from the bandwidth trace. The fluctuation is caused by the shadowing and fading of the obstacle.

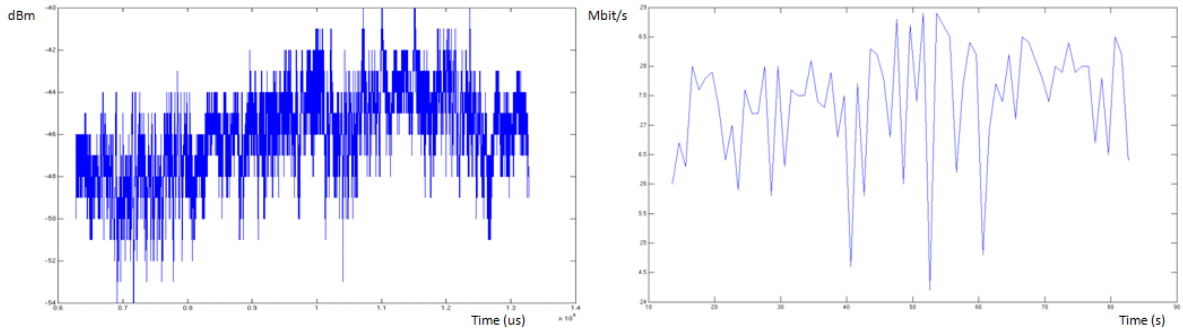


Figure 71 The traces of RSSI and throughput performance

2.6.2 CREW-Geni*

2.6.2.1 Year 3 collaboration

The CREW-GENI collaboration consists in creating a common ontology for describing spectrum sensing and cognitive radio experiments that is sufficiently generic and expressive to be used by a large number of experimentation facilities. The work on the ontology started from the CREW common data format (CDF) that is used by the CREW federation for experiment specification as reported in D5.1.

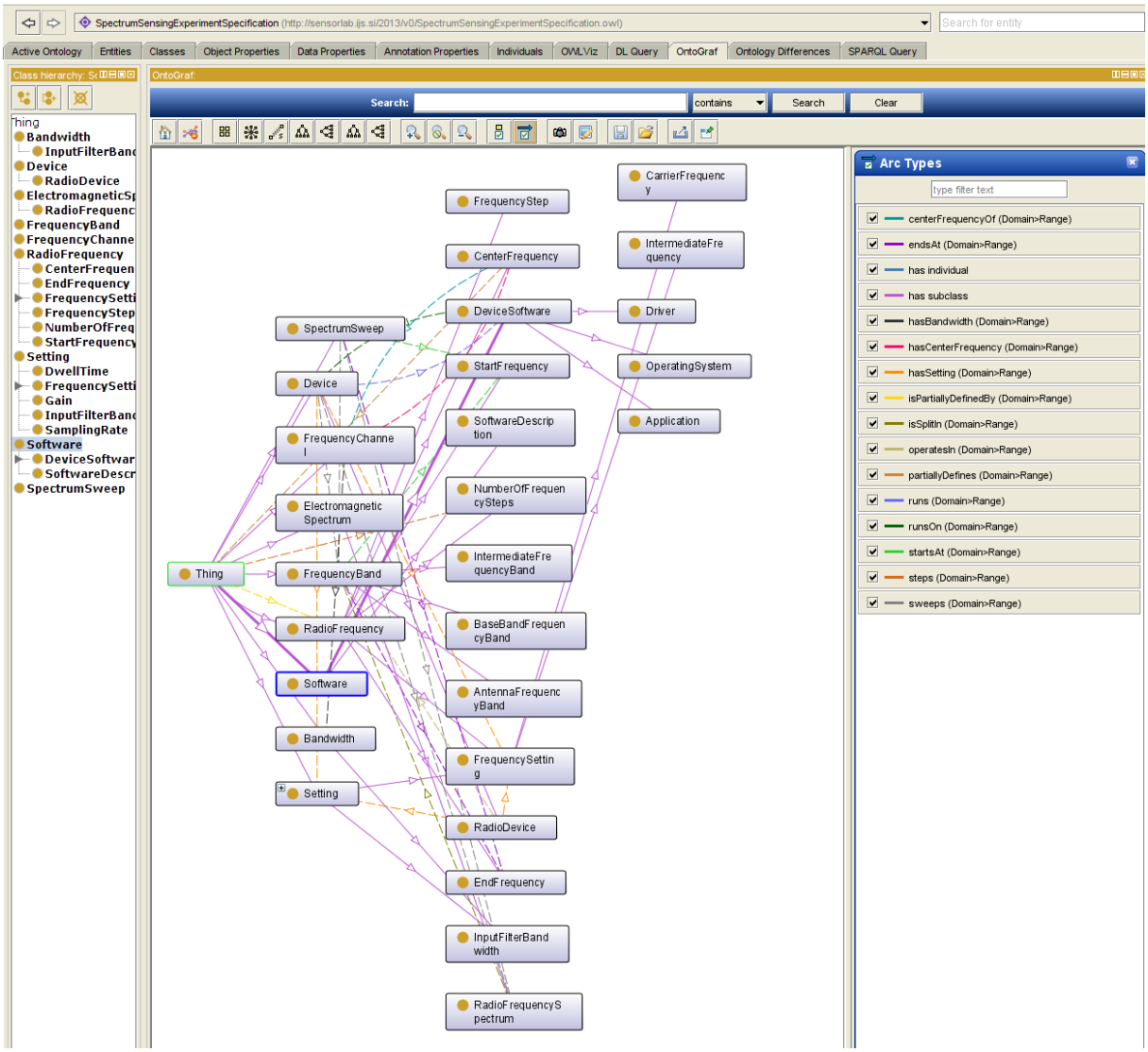


Figure 72 The Spectrum sensing experiment description ontology.

The current version of the ontology focuses on specifying device capabilities in the scope of configuring spectrum sensing experiments and is depicted in Figure 72. The latest version of the ontology with a wiki section to which information will be added is available at https://github.com/cfortuna/CROntology

The Spectrum sensing experiment description ontology is being used by the Orbit testbed in the Orbit device inventory:

- Human readable version http://www.orbit-lab.org:8080/tasor/#http://sensorlab.ijs.si/2013/v0/SpectrumSensingExperimentSpecification.owl%23RadioDevice
- Machine readable version http://www.orbit-lab.org:8080/tsc/resources/OrbitInventory

In the Spectrum sensing experiment description ontology we define three orthogonal concepts that allow the description of:

- spectrum related theoretical aspects,
- device spectrum sensing capabilities and
- ranges of values for each

This is depicted in Figure 73 where the light blue concepts represent the theoretical layer, the light green ones represent the device spectrum sensing capability layer and the dark blue individuals represent ranges that can be sets or intervals.

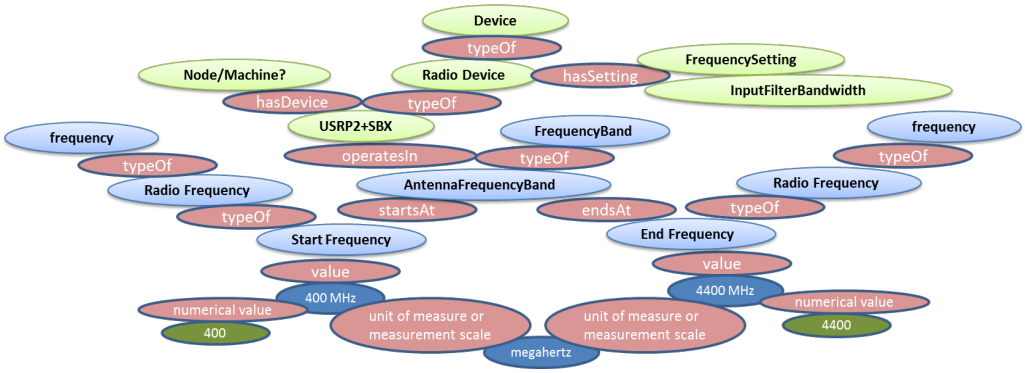


Figure 73 Conceptualization of the 3-layered approach.

2.6.2.2 Year 4 collaboration*

The CREW-GENI collaboration continues in the fourth year of the project with the finalization of the ontology and its usage in the TaSoR repository at Rutgers University (part of GENI). The results have been presented at the FIA FIRE workshop and will be published in a conference or workshop paper that is currently being finalized.

3 Demand-driven Extensions Derived from Open Call 2 Experiments*

In this section we describe the extensions and actions that were necessary to support the experiments carried out by the new project partners who participated in CREW via the second open call (WP7). The section is updated with the support provided also during Year 4 of CREW. All the OC2 experimenters also provided at the end of the experiment leaflets describing their experiments, feedback and recommendations for further improving the facilities.

3.1 Support for CREW-TV

3.1.1 JavaScript library for communication with LOG-a-TEC testbed

CREW-TV involves integrating the LOG-a-TEC testbed as a spectrum sensing data provider with the geolocation TVWS database developed by Instituto de Telecomunicações. The TVWS database must be able to instruct sensor nodes in the testbed to periodically perform spectrum sensing sweeps of the UHF frequency band and retrieve channel occupancy data for integration with existing database contents.

For the high-level testbed control the preferred means of programmatic access to the LOG-a-TEC testbed is the Python module library *vesna-alh-tools*. However, because the existing TWVS database implementation is incompatible with Python, it was decided that using *vesna-alh-tools* was impractical in this case.

Because of the familiarity of Instituto de Telecomunicações with JavaScript, an open source JavaScript library analogue to *vesna-alh-tools* has been developed at JSI instead to serve as a middle layer between the TVWS database and the LOG-a-TEC testbed. This library is called *vesna-alh-js*[13]

[13] https://github.com/sensorlab/vesna-alh-js

vesna-alh-js provides JavaScript equivalents of objects from the Python module library. It can be used in a web browser or a headless server environment like node.js. Like the Python library, it provides convenient methods of accessing the HTTP-like resources exposed by the sensor nodes over the Internet gateway and the LOG-a-TEC management network. These are contained in the *ALHWeb* and *ALHProxy* objects. On top of these abstractions, a higher layer of objects is built that abstracts the spectrum sensing abilities of sensor nodes. Namely the *SpectrumSensor*, *SpectrumSensorProgram* and *SpectrumSensorResult* classes allow for programming individual sensor nodes equipped with the SNE-ISMTV-UHF receiver for energy detection spectrum sensing tasks and retrieving the results over the Internet. Currently the JavaScript implementation lacks the other abstractions present in the Python module, like the support for signal generation and firmware reprogramming, since these features have not been required for integration with CREW-TV database.

Following is a brief description of the object classes exposed by *vesna-alh-js*:

ALHWeb(base_url, cluster_id)

ALH protocol implementation through the HTTP infrastructure gateway server.

Constructor parameters:

- **base_url** Base URL of the HTTP API (e.g. https://crn.log-a-tec.eu/communicator)
- **cluster_id** Numerical cluster ID

ALHProxy(alhproxy, addr)

ALH protocol implementation through an ALH proxy.

This implementation forwards arbitrary ALH requests through the *nodes* resource on an ALH service used as a proxy.

Constructor parameters:

- **alhproxy** ALH implementation used as a proxy
- **addr** Zigbee address of the node to forward requests to

SpectrumSensor(alh)

ALH node acting as a spectrum sensor.

Constructor parameters:

- **alh** ALH implementation used to communicate with the node.

SpectrumSensorProgram(sweep_config, time_start, time_duration, slot_id)

Describes a single spectrum sensing task.

- **sweep_config** Frequency sweep configuration to use.
- **time_start** Time to start the task (UNIX timestamp)
- **time_duration** Duration of the task in seconds
- **slot_id** Numerical slot ID used for storing measurements

SpectrumSensorResult

Result of a spectrum sensing task.

Public properties:

- **program** *SpectrumSensorProgram* object that was used to obtain this result.

- **sweeps** Array of *Sweep* objects containing results of frequency sweeps.

The fact that *vesna-alh-js* can be used from a web browser greatly simplifies experimentation with the LOG-a-TEC interface since no additional software needs to be installed on the experimenter's computer. However because of the same-origin security policies of most common web browsers, the JavaScript code running in the browser cannot usually access the LOG-a-TEC gateway directly. This can be worked around by disabling the same-origin policy in the browser, or by loading the *vesna-alh-js* and accessing the LOG-a-TEC through a specially created, standalone HTTP proxy server *devserver.py* that comes with *vesna-alh-js* distribution. This proxy, written in Python, makes the *vesna-alh-js* JavaScript module and the LOG-a-TEC Internet gateway appear to come from the same originating domain from the standpoint of the browser's JavaScript interpreter.

3.1.2 Geolocation data for sensor nodes

To integrate spectrum sensing data obtained from sensor nodes in the LOG-a-TEC testbed into the CREW-TV TVWS database the geolocation (i.e. geographic coordinates, latitude and longitude) of receiver antennas need to be known.

Individual sensor nodes in the LOG-a-TEC testbed are not equipped with a GPS receiver. However at deployment of each sensor node into the testbed the mounting location has been manually noted using a hand-held GPS receiver. Since the nodes themselves are not mobile the location of any individual node can be determined from its network address and the list of coordinates compiled at testbed deployment.

To make the LOG-a-TEC integration with the TVWS database more flexible and reduce the need for hard-coded knowledge of the testbed on the Instituto de Telecomunicações side of the integration, geolocation information for individual sensor nodes has been made available through the HTTP-like protocol that is also used to measure and access the spectrum data. This involved making sensor nodes in the testbed aware of their location and exposing this knowledge through a resource that can be queried over the LOG-a-TEC management network.

Specifically, this involved modifying the *description* resource on spectrum sensing nodes so that a GET query also reports the node's geographical coordinates. For example, the *description* resource on node 19 in the industrial zone environment of the LOG-a-TEC testbed now returns the following:

GET nodes?19/description

id:19

mac:20

firmware:2.35

location:14.2375512,45.931374

description:node 19

The *location* field contains geographical longitude and latitude, separated with a comma. This information can then be directly used by the TVWS database together with spectrum data, without relying on any external database tables mapping network addresses to coordinates.

Since sensor nodes themselves cannot obtain the location data, the coordinates have to be programmed after each sensor node deployment. The *description* resource handler on the sensor node stores the location data in a section of the non-volatile MRAM present on the sensor node code (SNC) board. The contents of the MRAM can be reprogrammed by issuing a POST request for the *description* resource.

3.1.3 Direct digital synthesis support for VESNA with SNE-ISMTV-868

TVWS database in CREW-TV will use spectrum sensor receivers in the LOG-a-TEC testbed as a distributed spectrum sensor. To test this dynamic component of the database and demonstrate its abilities primary users have to be simulated. To remove the need to manually introduce wireless microphones into the testbed it has been decided to simulate wireless microphone transmissions using the remotely accessible sensor nodes in the LOG-a-TEC testbed as well.

10 nodes in the LOG-a-TEC testbed are equipped with the SNE-ISMTV-868 transceiver that is capable of transmitting a narrow-band signal at frequencies between 779 and 928 MHz. This frequency span includes the upper UHF channels that are used by licensed wireless microphone users. One of more of these nodes will be remotely triggered to act as wireless microphones during CREW-TV experiments and demonstrations.

To create as realistic environment as possible in the testbed it was decided to use the IEEE wireless microphone simulation profiles when testing the CREW-TV deployment (*Chris Clanton, Mark Kankel and Y. Tang, "Wireless Microphone Signal Simulation Method", IEEE 802.22-07/0124r0, March 2007*). This method approximates a radio signal transmitted by a legacy analogue studio wireless microphone (PMSE) using a frequency modulated continuous sine wave. Three operating conditions for the microphone are defined in literature: silent, soft speaker and loud speaker, each with its own FM parameters:

Operating mode	**F_m [kHz]**	**F_{dev} [kHz]**
Silent	32.0	5.0
Soft	3.9	15.0
Loud	13.4	32.6

Table 11: FM parameters for wireless microphone simulation profiles.

Because the SNE-ISMTV-868 transceiver is not capable of transmitting a modulated analogue signal as required by this method, an approximation has been implemented. The transceiver has been configured for a continuous digital frequency-shift keying transmission with 4 symbols (4FSK) and 200 ksymbols/s. This is equivalent to a transmission of a sampled analogue signal with 2-bit precision and 200 kHz sampling rate.

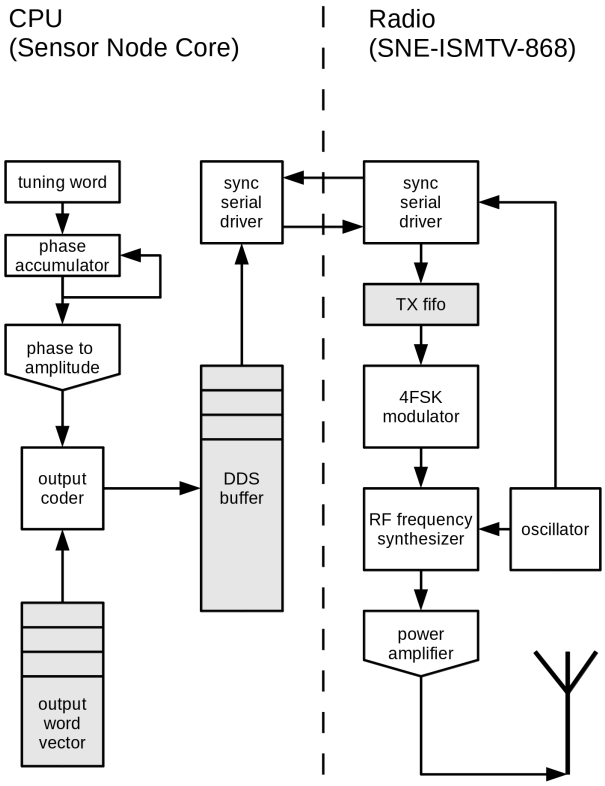


Figure 74: Block diagram of Direct Signal Synthesis on VESNA with SNE-ISMTV expansion

A direct digital synthesis (DDS) algorithm has then been implemented in software on the sensor node's CPU to continuously feed a sine waveform to the transmitter hardware through a synchronous serial bus. Modulation frequency of the modulation signal can be set by adjusting the DDS tuning word in software while the FM deviation can be set by changing the hardware FSK modulator deviation setting. These two settings allowed us to closely match the FM parameters required for all three wireless microphone simulation profiles.

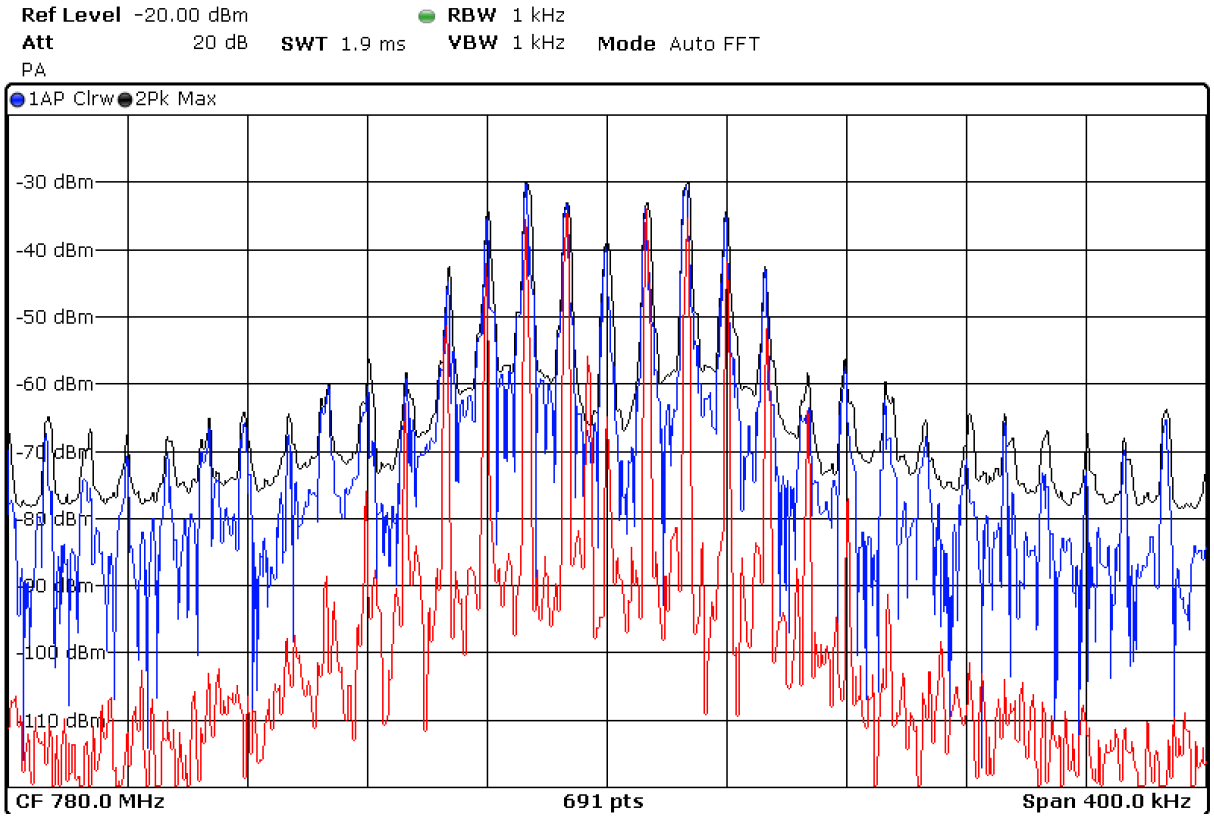


Figure 75: Spectrum of a "loud speaker" wireless microphone simulation signal produced by SNE-ISMTV (blue trace) compared to the same signal produced by a USRP device (red trace)

Wireless microphone simulation profiles have been seamlessly integrated with the existing signal generation API exposed by sensor nodes in the LOG-a-TEC testbed. All 10 nodes with the SNE-ISMTV-868 transceiver have been updated with new firmware supporting wireless microphone simulation. When queried for signal generation profiles, these nodes now report 6 additional signal generation configurations, two for each wireless microphone simulation profile (due to the limitation on the number of channels per configuration, the UHF frequency band had to be split between two different profiles)

GET nodes?8/generator/deviceConfigList

dev #0, CC1100, 9 configs:

cfg #0: CC1100, FM noise, 200 kHz deviation:
base: 779999908 Hz, spacing: 199814 Hz, bw: 197754 Hz, channels: 256, min power: -30 dBm, max power: 12 dBm, time: 5 ms

cfg #1: CC1100, FM noise, 200 kHz deviation:
base: 829999924 Hz, spacing: 199814 Hz, bw: 197754 Hz, channels: 160, min power: -30 dBm, max power: 12 dBm, time: 5 ms

cfg #2: CC1100, wireless mic, silent:
base: 779999908 Hz, spacing: 199814 Hz, bw: 197754 Hz, channels: 256, min power: -30 dBm, max power: 12 dBm, time: 5 ms

cfg #3: CC1100, wireless mic, silent:
base: 829999924 Hz, spacing: 199814 Hz, bw: 197754 Hz, channels: 160, min power: -30 dBm, max power: 12 dBm, time: 5 ms

cfg #4: CC1100, wireless mic, soft speaker:
base: 779999908 Hz, spacing: 199814 Hz, bw: 197754 Hz, channels: 256, min power: -30 dBm, max power: 12 dBm, time: 5 ms

cfg #5: CC1100, wireless mic, soft speaker:
base: 829999924 Hz, spacing: 199814 Hz, bw: 197754 Hz, channels: 160, min power: -30 dBm, max power: 12 dBm, time: 5 ms

cfg #6: CC1100, wireless mic, loud speaker:
base: 779999908 Hz, spacing: 199814 Hz, bw: 197754 Hz, channels: 256, min power: -30 dBm, max power: 12 dBm, time: 5 ms

cfg #7: CC1100, wireless mic, loud speaker:

base: 829999924 Hz, spacing: 199814 Hz, bw: 197754 Hz, channels: 160, min power: -30 dBm, max power: 12 dBm, time: 5 ms
cfg #8: CC1100, 868 MHz SRD band, FM noise, 50 kHz deviation:
base: 863999695 Hz, spacing: 49953 Hz, bw: 49438 Hz, channels: 140, min power: -30 dBm, max power: 12 dBm, time: 5 ms

3.1.4 Estimating DVB-T signal quality using a low-cost receiver*

3.1.4.1 Introduction*

CREW-TV experiment used a dynamically updated geolocation database to assign vacant frequency channels in the TV whitespaces to an experimental video streaming system. The video streaming demonstrator acted as a secondary user that was required to avoid interference with primary users. The final trials of the system were performed in Logatec municipality. Two known primary users of the UHF spectrum where the demonstrator operated were two DVB-T multiplexes. These signals were transmitted from a local transmitter tower located on a low hill in the center of the municipality. One of the objectives during the trial was to estimate the impact, if any, of the experimental system on the quality of TV reception in the area.

The objective was to estimate the quality of digital TV reception for an average user. Therefore it was preferred that the measurement was performed using a device that would be matched in specifications as closely as possible to ordinary, low-cost consumer devices. Because of that consideration we preferred not to use laboratory equipment in this evaluation.

Initial requirements called for a visual inspection of the video quality using a digital TV receiver: an experimenter would watch the TV broadcast while the experimental system was in different modes of operation and would note any visible artifacts in the picture or sound. If possible, an objective measure of the signal quality, for example signal-to-noise ratio (SNR) or bit error rate (BER) would also be recorded at the same time for later analysis.

Several spectrum sensing nodes for the UHF band are permanently installed in the LOG-a-TEC outdoor sensor network testbed. These nodes were used as parts of the distributed sensing network for providing up-to-date channel occupancy data to the geolocation database. The sensor nodes were used to detect wireless microphone transmissions as well as DVB-T broadcasts. However signal detection on the nodes was limited to energy detection. Current hardware deployed in LOG-a-TEC is not capable of decoding a DVB-T signal or measuring any signal quality parameters except for received signal strength. Therefore, a custom DVB-T monitoring solution needed to be developed for the purpose of the CREW-TV OC2 experiment.

3.1.4.2 Experimental DVB-T receiver setup*

For the trials we planned to perform measurements from a mobile measurement station consisting of a van equipped with antennas and signal monitoring equipment. Because of the limited space available and other practical considerations we opted against a full-sized commercial TV receiver. Instead we used a compact, low-cost USB-connected DVB-T receiver in combination with a laptop computer.

Our particular receiver reported the capability of measuring four signal parameters:

- Received signal strength indicator (RSSI),
- signal-to-noise ratio (SNR),
- bit error rate (BER) and
- uncorrected blocks count

During the preparations for the experiment however we discovered the Linux kernel driver for our DVB-T receiver would consistently return an error when reading RSSI and uncorrected blocks count statistics. This causes the “dvb-snoop” program to terminate with an error. To work around this problem we created a modified version of the program that ignored the kernel error and kept recording the statistical data. In the later analysis we discarded the RSSI and uncorrected blocks count data.

3.1.4.3 Evaluating the measurements*

The mobile measurement setup that was used in the CREWTV trials can be seen in Figure 76.

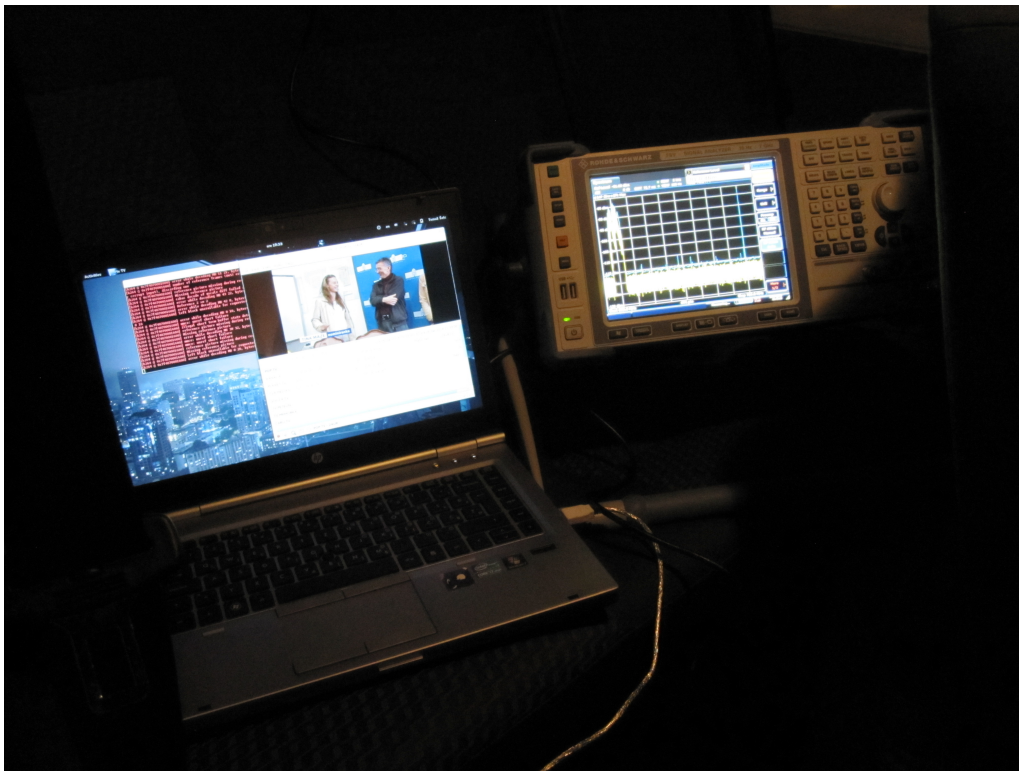


Figure 76: Recording DVB-T signal quality statistics on a laptop with the DVB-T receiver.

Before the trials we have performed several data collection runs using this method to evaluate its reliability. Shown in Figure 78 and Figure 79 are results of two example measurement runs, each consisting of approximately 10000 samples of the signal quality parameters.

Figure 77: Ezcap DVB-T receiver used during the trials

The Ezcap DVB-T receiver was chosen because it was readily available and because of relatively stable operation with the Linux operating system running on the laptop used in the measurement station. During the experiments, the original small dipole antenna for the Ezcap receiver was used, magnetically attached to the roof of the van.

"Me TV"[14] software was used on the laptop to set the receiver parameters for the reception of the DVB-T signal transmitted from the local transmitter (channel frequency and bandwidth, MPEG stream, forward error correction rate, modulation scheme, number of carriers and guard interval width). This same software was also used to display the stream on laptop's monitor for visual inspection.

Linux DVB API allows for digital video receivers to also report signal quality parameters in addition to providing the video stream. To record the values of these parameters during the experiment, "dvb-snoop"[15] software was used. We used the signal statistics mode of the program ("dvb-snoop –s signal")

Because RSSI and uncorrected block counts were unreliable, we only included SNR and BER in the analysis. We have not tested this method in a controlled environment, therefore we cannot be certain that even these two parameters are calculated correctly by the device. However from the figures below we can see that the statistical distribution of returned values changed from one measurement run to the other, which made us confident that the values have some relation with reality.

Documentation for the driver does not mention the units in which these two measures are recorded. Numerical values returned are not consistent with any well-known unit. Therefore this method is only usable for monitoring relative changes in the signal quality.

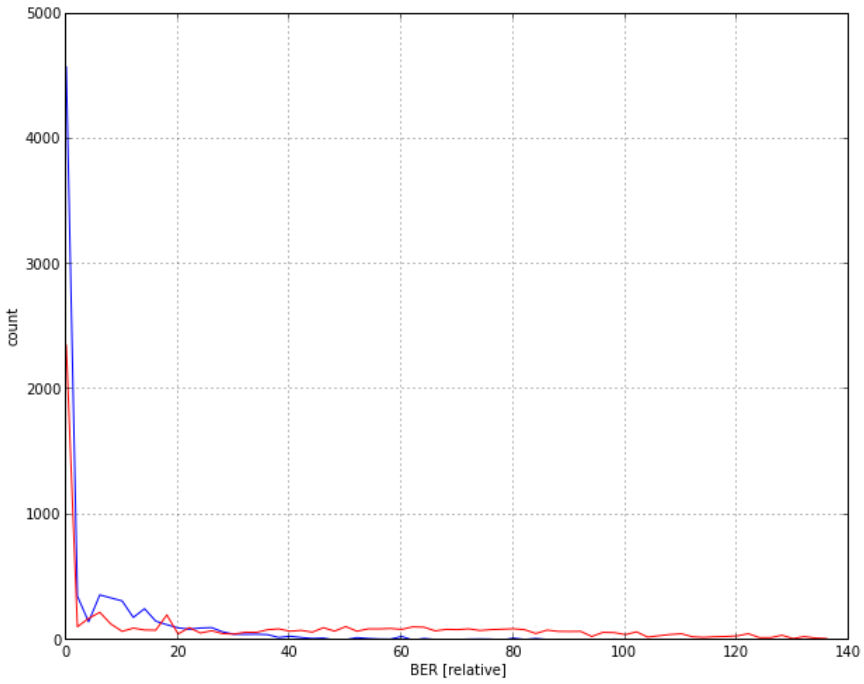


Figure 78: BER histogram for two different test measurements.

[14] Available from https://launchpad.net/me-tv

[15] Available from http://dvbsnoop.sourceforge.net

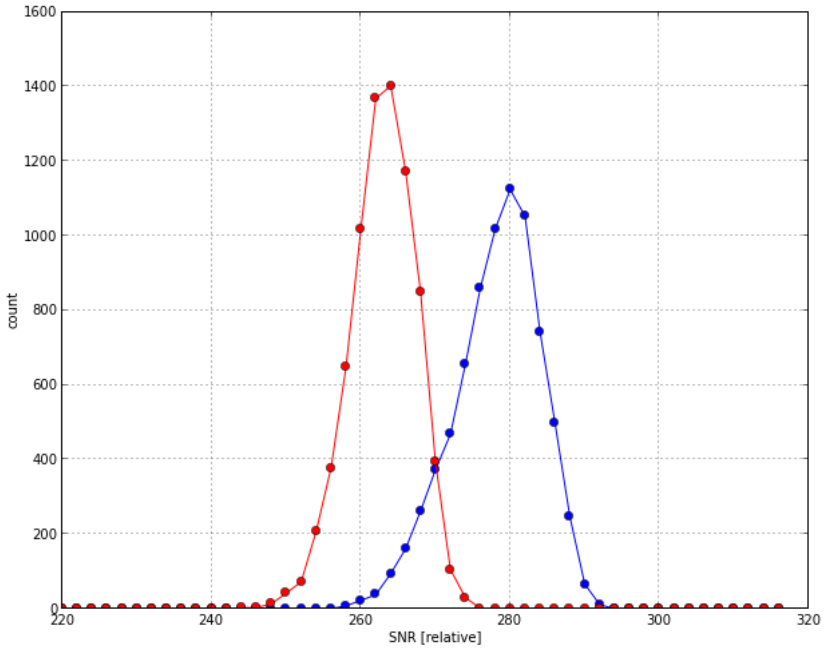


Figure 79: SNR histogram for two different test measurements.

3.2 Support for EVOLVE*

3.2.1 Support in year 3

The EVOLVE experiment made use of the wireless nodes in the iMinds w-iLab.t. The testbed framework was used for provisioning the nodes and executing repeatable experiments (OMF). Experiments were executed using the 802.11a/b/g/n wireless cards. Several wireless multi-hop scenarios were set up to test the signalling load of the proposed algorithm.

No additional hardware or software was installed in the w-iLab.t testbed to support the EVOLVE experiment. Technical support was provided during the experiment setup phase regarding the use of the testbed and the hardware of the wireless nodes.

In addition, the EVOLVE project made use of the Iris testbed at TCD to develop an OMF control interface for the Iris software radio framework. Two researchers from WINGS ICT Solutions visited Dublin and used the testbed onsite with technical and design support from TCD researchers.

3.2.2 Support in year 4*

In addition to testing only on fixed wireless nodes, iMinds supported the EVOLVE experiment to run on mobile nodes. A close collaboration was set up between the w-iLab.t administrators and the EVOLVE experimenters to further develop and improve the mobility framework in the w-iLab.t testbed.

Experiment statistics:

- Number of reservation slots: 44
- Average duration of a reservation entry: 9 hours
- Average number of fixed wireless nodes: 6
- Average number of mobile nodes: 1

- Total duration of reservation entries: 17 days

3.3 Support for CABIN-CREW*

3.3.1 Support in year 3

Support to CABIN-CREW was provided for the two proposed experiments:

To support experiments with the Wireless MAC Processor (WMP) in the iMinds testbed, four Alix devices equipped with Broadcom WiFi cards were installed in the iMinds testbed. These nodes were integrated in the Emulab environment and several experiments were created to use the nodes as well as one of the USRPs as a measurement device. OMF support was installed on the alix nodes and the network configuration was modified to support OMF experiments. Additional OMF support was provided in person and by e-mail.

For the planned experiments at TUB, three Alix devices equipped with Broadcom WiFi cards where installed in the TUB testbed. Those nodes where integrated in the new OMF environment. It is possible to use them in the experiments.

For the WARP based experiments a temporary setup was installed at iMinds itself to evaluate remote operation capabilities. Documentation was provided to use this setup and two more WARPs were installed in the iMinds testbed and integrated in Emulab. As was the case for the alix nodes, the required experiments were created to allow remote experimentation. The WARP boards can be programmed from the connected Zotac nodes using installed Xilinx LabTools, while the ethernet connection is connected to a gigabit switch and can be connected to any of the 8 available servers where Matlab is available for processing purposes. OMF is not yet supported for the WARP.

3.3.1 Support in year 4*

In addition to the original OC2 experiments, the collaboration between CNIT and iMinds led to an additional experiment showcasing the capabilities of the WMP (developed by CNIT) and SnapMAC (developed by iMinds). A short description of this experiment is included in this deliverable, a more detailed description of this experiment can be found in D7.6.4.

The goal of this experiment was to demonstrate the feasibility of rapidly developing new MAC solutions using the WMP and SnapMAC. To this end a cross-technology TDMA scheme as illustraded in Figure 80 was implemented on both WMP-enabled WiFi nodes and SnapMAC enabled sensor nodes.

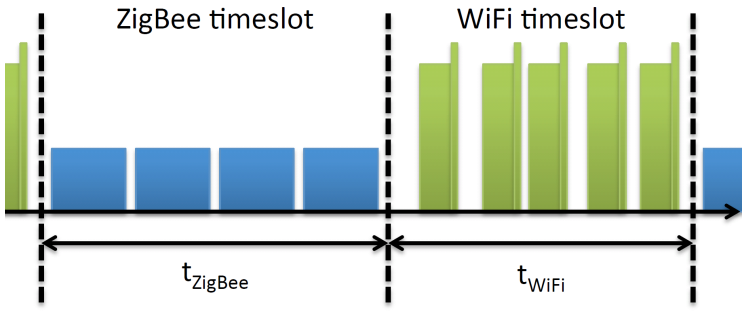


Figure 80: Implemented cross-technology TDMA scheme [49]

By controlling this TDMA scheme with a cognitive loop (Figure 81) that is capable to detect cross-technology interference, it became possible to significantly reduce wasted channel time when WiFi and ZigBee nodes are using the same channel at the same time. This is demonstrated in Figure 82:

when a legacy WiFi interferer is active, the PER of the ZigBee link is about 25-30%. By using the TDMA scheme, PER returns to interference free levels without any throughput drawbacks.

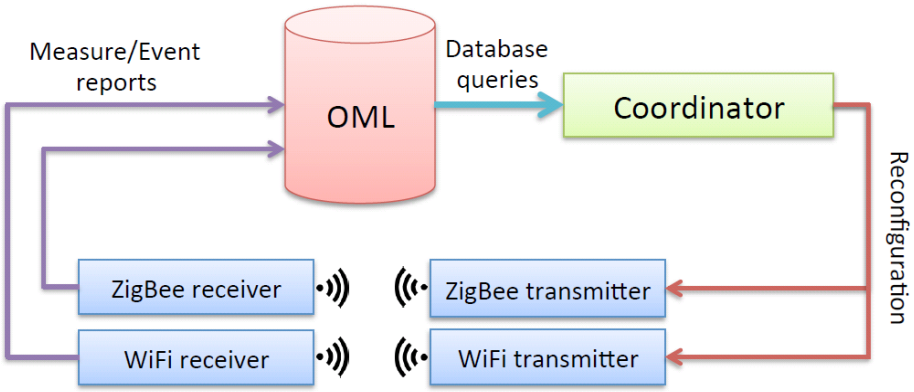


Figure 81: Cognitive cycle of the experiment [49]

During this experiment iMinds was responsible for the implementation of the proposed communication scheme on the RM090 sensor nodes. The back-end scripts for running the experiments and the cognitive cycle were jointly developed by CNIT and iMinds as well as the visualization web-interface. This experiment was successfully demonstrated at the OC2 review meeting in Bologna and the results have been reported in a journal paper that is under review.

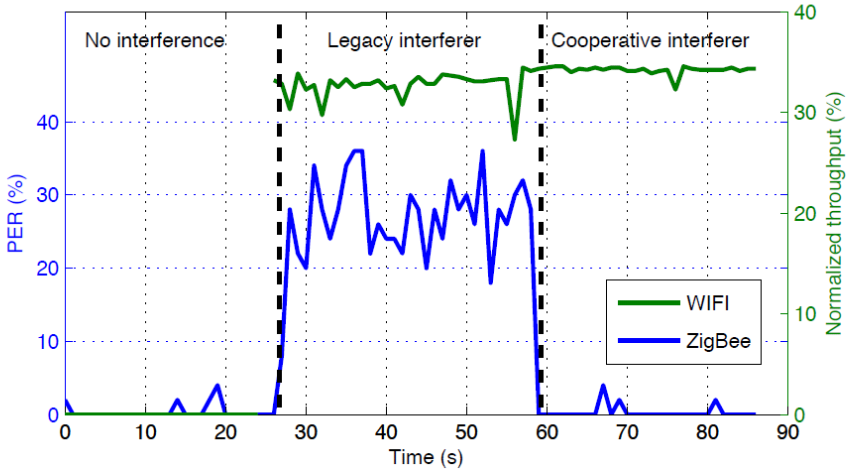


Figure 82: Using the TDMA scheme results in a reduced PER [49]

3.4 Support for UTH + NICTA*

3.4.1 Measurement of sensing delay in USRP sensing engine

This section focus on the support we provided for measuring the sensing delay of USRP based sensing engine. The experiment involves two types of USRP's, the USRP N210 and USRP E110.

The USRP N210 belongs to the network series in the USRP product family. It connects to a host computer via fast Ethernet connection, and relies on the host for signal processing. The USRP E110 belongs to the embedded series, it is meant to be used standalone. There is an ARM processor on board of the E110, and hence signal processing can be conducted on the device itself. In case of spectrum sensing, USRP N210 only streams raw samples to the host computer while E110 provides the final result to the host computer. It is interesting to comparing the delay of the two distinctive

software radio approach, which is also the main reason that we consider to use the two distinctive types of USRP's.

For the sensing with USRP N210, we make use of the Iris platform. Several Iris components are developed for spectrum sensing. We provide those sensing components and guide lines on how to use them. In the beginning of February, one UTH researcher visited our testbed for a hands-on experience of using USRP on w-iLab.t. Iris platform provides built-in time delay measurement inside its core. For each component there is an average timing report about how much time it consumes at the end of the execution. Therefore we do not need to modify the software especially for the delay measurement.

For the experiment with USRP E110, we had an initial test of the hardware. The E110 boots from a micro SD card, which contains a disk image of embedded Linux with the UHD driver and GNU Radio installed. The first effort we made is to update the disk image, because the factory version of the disk image is out of date and somehow unstable. After that we also explored to use the default UHD driver on the E110 for simple FFT measurement. We passed the knowledge of using E110 together with the hardware to UTH for further measurement.

3.4.2 Measurement of the energy consumption and sensing delay of the IMEC sensing engine

The IMEC sensing engine is a compact stand-alone sensing device that combines a low-power sensing ASIC with a wideband reconfigurable analog frontend. The design goal of the IMEC sensing engine was to design a monolithic and rugged device and as a consequence the options for monitoring internal signals are limited. To cater for the requirements of this experiment the various power nets of the sensing engine (digital part and the two nets of the analog frontend) were made available. Some iterations back and forth were required to get the IMEC sensing engine up and running in the UTH lab. Next to the physical transfer of the hardware a software mapping task was required: in order to enable a comparison between the various sensing engines a new firmware mode was mapped on the IMEC sensing engine. The new firmware enables identical functionality over the various sensing engines (max-hold of band scanning sensing through FFT).

3.4.3 Support in year 4*

To facilitate the experiment, 6 ACM (Advanced Chassis Manager) cards developed by UTH-NICTA are deployed in the w-iLab.t testbed. Additionally, iMinds also assisted with the process to collect measurement data from the ACM card and to integrate this setup into an OMF / OML experiment. The capability of this platform was also presented by iMinds as part of the CREW facility on the CREW training days 2014. More details of the support for this experiment are described in section 3.1 of D7.7.3.

Some experiment statistics:

- Number of reservation slots: 26
- Average duration of a reservation entry: 30 hours
- Average number of fixed wireless nodes: 4 (+ 2 servers, + 6 NITOS CM cards + 2 USRP)
- Average number of mobile nodes: 0
- Total duration of reservation entries: 32 days

4 Open call 3 Experiments*

This section summarizes the OC3 experiments with emphasis on the final demos and final results. The detailed report on the actual support and OC3 demand driven extensions are described in the next section.

4.1 CARE*

CARE is an OC 3 experiment proposal from the Paris Descartes University, France. The experiment has been performed on the w.i-Lab.t testbed with the support of iMinds.

The CARE experiments investigated distributed algorithms for solving the blind rendezvous problem, namely the problem of establishing a control link on the same channel among secondary users of a cognitive radio system without any central coordination. Specifically, we evaluate a rendezvous protocol that guarantees to randomly select a common channel in fully decentralized environments even with asymmetrical system properties (e.g., asymmetrical channel perceptions for the different CR nodes and system clock drift).

The rendezvous protocols have been implemented for evaluation on the testbed, and preliminary experiments were performed in typical application scenarios. Specifically, the impact of coordination among cognitive nodes on the network performance when using some of the protocols designed to establish a common control channel without any central coordination have been investigated.

The baseline scenario where two rendezvous nodes have symmetrical channel perceptions and their slots are aligned is depicted in Figure 83. Specifically, the focus is on the following two performance metrics:

- The worst-case and average rendezvous delay,
- The rendezvous diversity, i.e., the ability to rendezvous on every common channel.

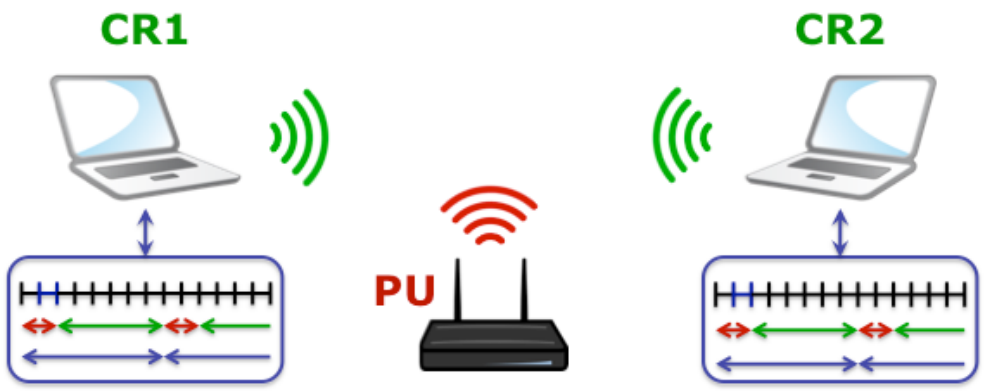


Figure 83 Bling rendezvous experiment scenario.

In the experiment, the primary User (PU) floods its channel(s) to simulate the transmission activity on the licensed spectrum. The Cognitive Radio (CR aka Secondary Users) nodes periodically switch channel to establish a common control channel.

We then evaluate the proposed solution with asymmetrical channel perceptions and unaligned slots, i.e., a challenging scenario in which the rendezvous nodes do not have the same knowledge on the number of channels, and their clocks are drifted away from each other for an arbitrary amount of time. The same performance metrics will be traced and analyzed to derive more insight on the proposed solutions.

As for the blind rendezvous experiment (second scenario), we evaluate the performance metrics discussed in the previous section, increasing the available number of channels (from 6 to 11). Therefore, we evaluate approximately 60 different configurations (30 with synchronized nodes and 30 with unaligned time slots). More experiments will be ran in the next two months to thoroughly evaluate the two scenarios.

4.2 GAME-COG-NET*

GAME-COG-NET is an OC 3 experiment proposal from the Technical University of Cluj-Napoca, Romania. The experiment has been performed on the LOG-a-TEC testbed with the support of JSI.

The GAME-COG-NET experiment applied a game theoretic approach to the problem of interference between spectrum users. A demo will be presented that features a cost-adaptive, discrete-power interference mitigation game for constrained devices. The convergence to stable states of a dynamic distributed power allocation algorithm will be shown on a 2.4 GHz reconfigurable testbed containing four active Tx-Rx pairs. The experiment has implications for the Internet of Things applications such as home and industrial automation.

In wireless networks transmit power control is important. By having less interference, more efficient use of spectrum can be made, battery life can be prolonged and higher energy efficiency can be achieved. A large number of game theoretic models for cognitive radio have been proposed thus far, however only few have been experimentally validated to take into account the practical limitations of the devices and the wireless environment (e.g. only discrete TX power levels supported by hardware). GAME-COG-NET validated a distributed, interference-aware power control algorithm that is suitable for M2M and wireless sensor network user scenarios. The work is based on previous work on the PAPU algorithm and studies the interactions between users in a real-world testbed, testing theoretical results on the JSI campus testbed indoor as depicted in Figure 84 and Figure 85 and outdoor depicted in Figure 86.

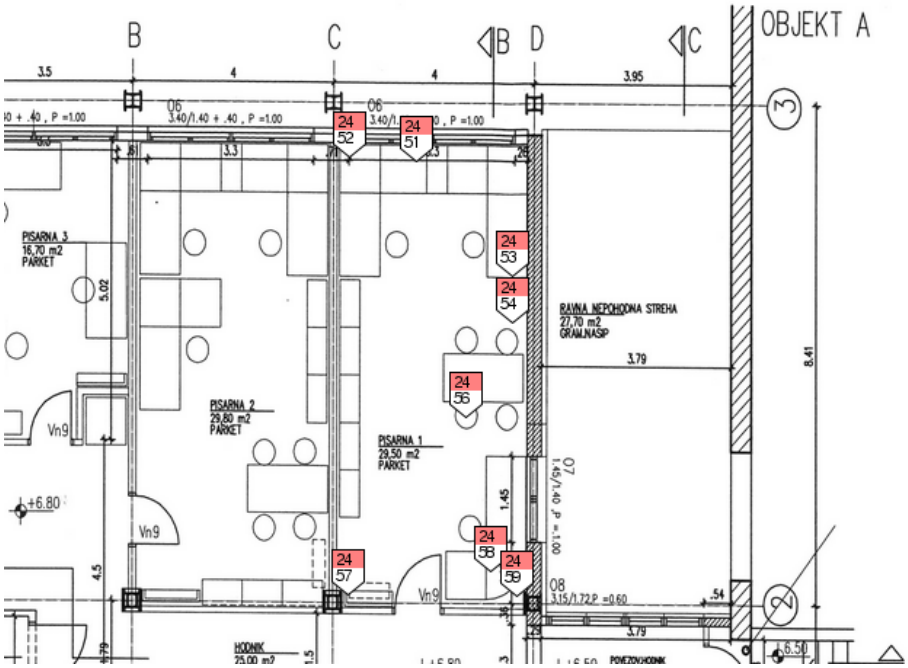


Figure 84: Location of nodes in the JSI in-door cluster for the GAME-COG-NET experiment

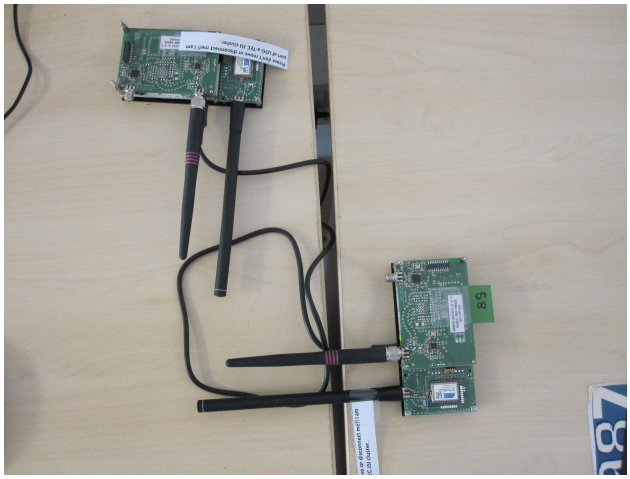


Figure 85: Photograph of nodes 58 and 59 in the JSI in-door cluster for GAME-COG-NET experiment.

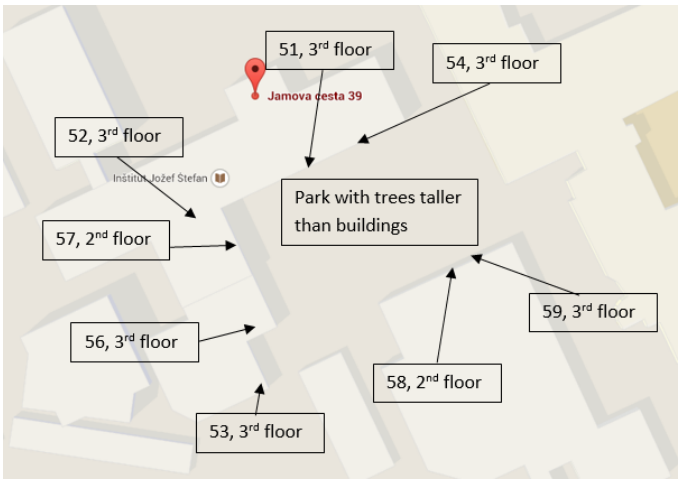


Figure 86: Location of nodes in the JSI out-door cluster for the GAME-COG-NET experiment.

The results of the experiment were reported in a journal paper that has been submitted in September 2014. Some results refer to estimating channel gains where two different approaches (i.e. sequential and parallel) were consider and the best selected for using during the game. For instance, in the case of sequential gain measurement, it takes 5-7 seconds for one direct or cross gain measurement. Even in an office environment with minimal movements, gains can vary considerably as can be seen in Figure 87, however, variations in the gain did not translate into a variation of the transmit power higher than the quantization step.

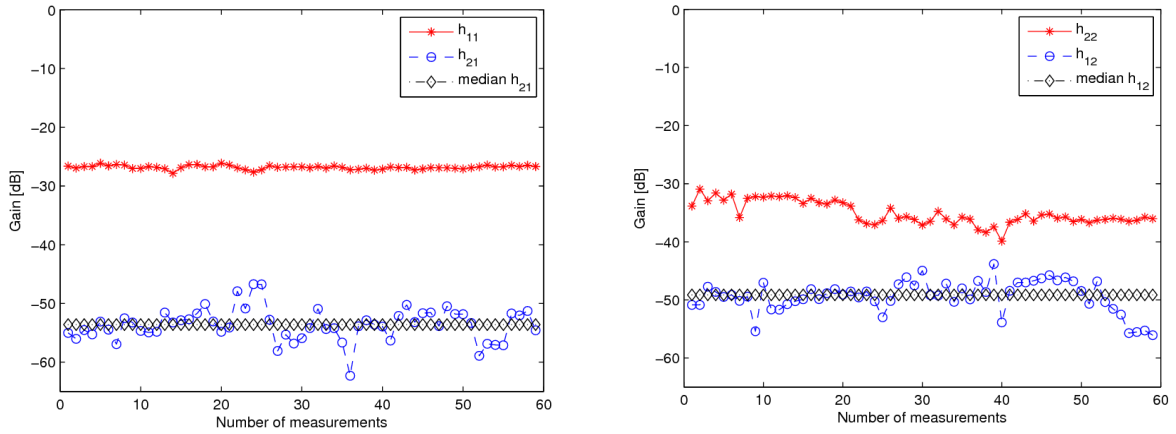


Figure 87 Sequential channel gain measurement for a 2 player game.

Other results of this experiment include the evaluation of the effect of the dynamic cost adaptation as depicted in Figure 88. The experiments confirm the theory that if costs are not adapted, a game designed with a continuous transmit power assumption will not converge and will oscillate continuously while with cost adaptation is will stabilize. Other results refer to a dynamic scenario in which players enter and leave the game. All this is detailed in a submission to IEEE Transactions on Wireless Communications.

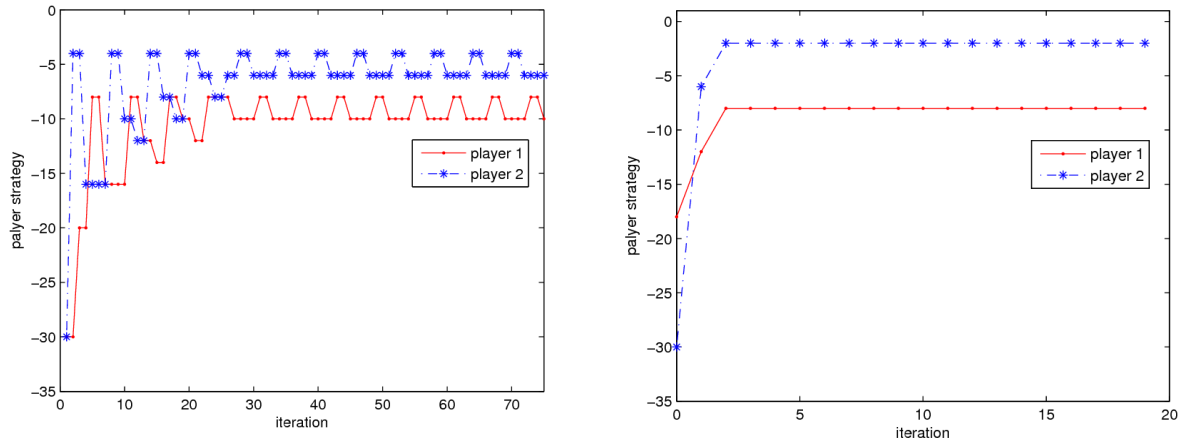


Figure 88 Sequential channel gain measurement for a 2 player game.

4.3 MUCO*

MUCO is an OC 3 experiment proposal from AED engineering Gmbh, Germany. The experiment has been performed on the TWIST testbed with the support of TUB.

We designed and implemented a hardware platform composed from a FPGA and a PCB for IEEE 802.15.4 wireless sensor networks. The platform is able to handle several channels within the 2.4 GHz band simultaneously. Using the TWIST testbed benchmark tests are conducted to evaluate the performance of this hardware compared to state-of-the -art network coordinators. The focus of these experiments are parameters like packet loss and error rate. The TWIST testbed is ideal to measure these parameters within a realistic environment using a large amount and different types of nodes.

4.4 picoMESH*

picoMESH is an OC 3 experiment proposal from TASS Belgium. The experiment has been performed on the w.i-Lab.t testbed with the support of iMinds.

The goal of the experiment is to prove that the Optimized Link State Routing Protocol (OLSR) (IETF RFC3626) implementation provided by PicoTCP fits a scenario where a mesh topology based on 802.15.4 is challenged by multiple issues (i.e. nodes with a limited visibility or partially hidden, nodes joining and leaving the topology with no notice).

When all the nodes are running the same firmware, addressing and routing is automatically configured so that all the nodes that are not in direct visibility can still be addressed, and the communication is guaranteed across the whole testbed with no intervention needed from outside. Moreover, the network is capable to react reasonably quickly to topology changes, such as nodes joining and leaving the network, by automatically adjusting the routing tables accordingly.

The consistency of the network is verified by exchanging data among nodes while forcing sudden topology changes, and evaluating the reaction time required by the nodes to cope with the introduced modifications at run time.

Furthermore, the routes can be affected by RSSI with a simple cross-layer modification to the OLSR module, in order to take into account the signal quality while calculating the best routes among each pair of nodes in the network.

OLSR is a standardized protocol which makes it possible to expand the topology across heterogeneous devices and physical layers, for example to joint multiple sections of the mesh network together even if they are physically placed in remote locations.

4.5 SIRI*

SIRI is an OC 3 experiment proposal from Ss. Cyril and Methodius University, Skopje, Macedonia. The experiment has been performed on the TCD, LOG-a-TEC and w.i-Lab.t facilities with the support of TCD, JSI and iMinds.

The SIRI demonstration intends to show the benefits of using the Radio environmental Maps (REMs) in small-cell optimization scenarios as depicted in the scenario of Figure 89. The demo foundation is a generic, modular, and flexible REM prototype implementation with on-field spectrum measurement devices to provide real-time environment data, and a rich set of statistical inference algorithms for, propagation modeling, single/multi source localization, Radio Interference Fields (RIFs) estimation, temporal and spatial channel occupancy estimation, etc (see Figure 90). In particular, the REM data serves as a feedback to the small-cells RRM, which calculates the optimal frequency, bandwidth and transmit power values for the small-cells. The synergy between REM and RRM can be especially beneficial in femto-cell scenarios due to the unavailability of clearly defined interfaces, techniques and solutions to manage the installation and the optimization of femto-cell operation.

The results of the SIRI experiment has been presented at two demos, one at IEEE Infocom 2014 [50] and the second at at the IEEE BlackSeaComm 2014.

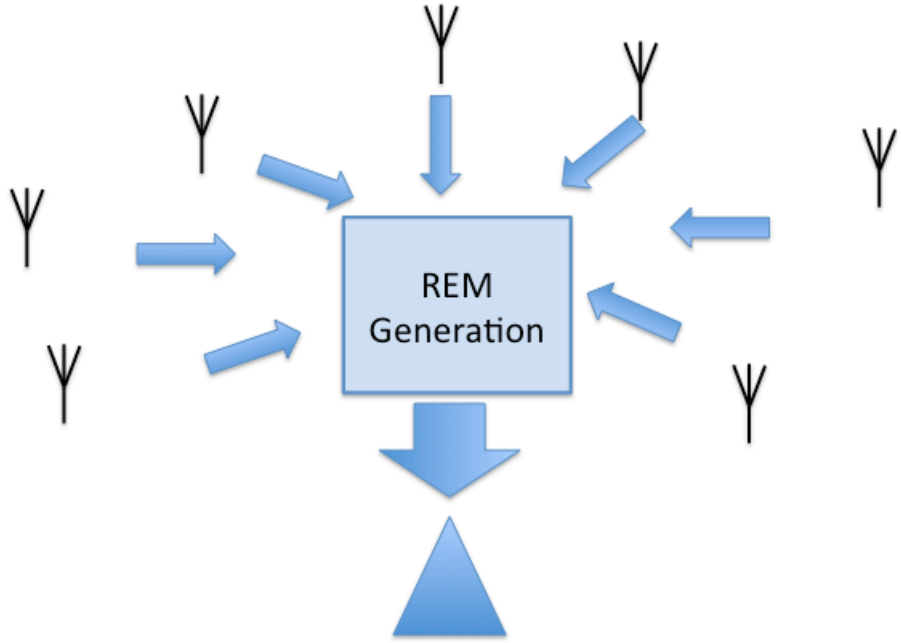


Figure 89 Radio environment maps for small cell optimizaton scenario.

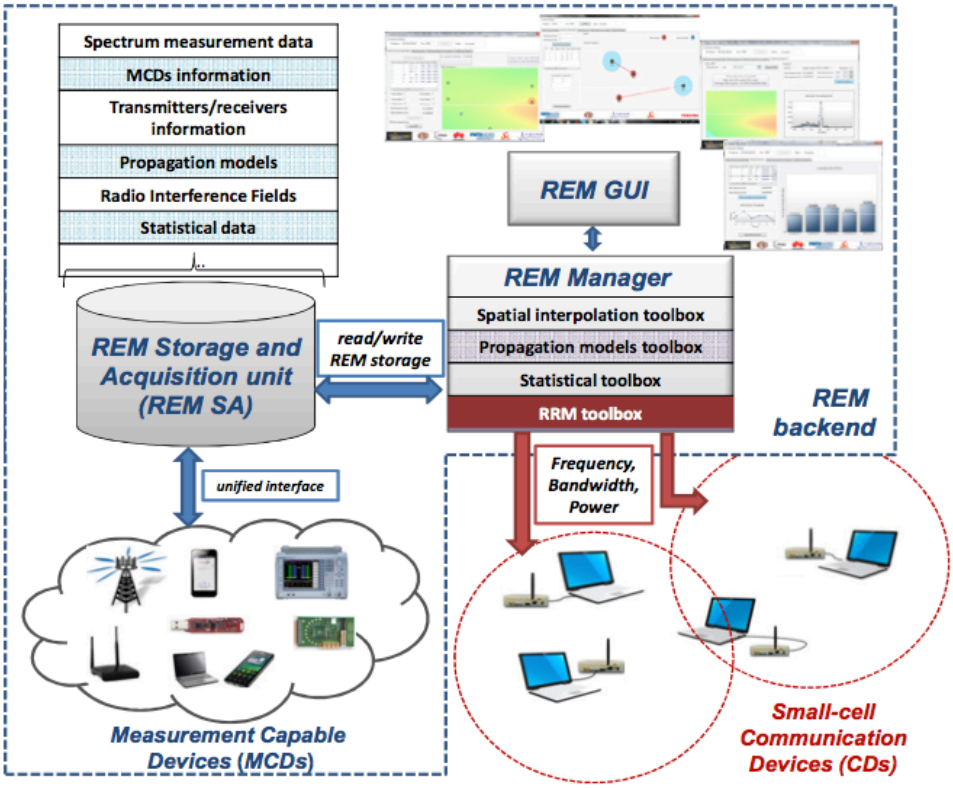


Figure 90 Integrated REM based small-cell optimization prototype (from [50]).

4.6 FACT*

FACT is an OC 3 experiment proposal from Katholieke Universiteit Leuven, Belgium. The experiment has been performed on the w.i-Lab.t testbed with the support of iMinds and IMEC.

The FACT experiment aims to further context-aware multi-hop communication networks. Key in those networks is the ability to adapt the communication across multiple layers to the dynamic context. One example is an aerial network, where airborne nodes are mobile and hence the network has to adapt its communication to the varying topology. The experimental network consists of a large number of off-the-shelf nodes at the CREW testbed (regular IEEE 802.15.4 nodes). In addition, the network has a central sink (steering the context-awareness) and two enhanced FACT data collection hubs. Those hubs are nodes that have more cross-layer configuration capabilities compared to the off-the-shelve nodes (based on off-the-shelf IEEE 802.15.4 chipsets). The nodes are Cross-layer Adaptable Wireless System (CLAWS) nodes. They consist essentially of an SDR (software defined radio) implementation of IEEE 802.15.4, running ContikiOS and 6LoWPAN. All functionality can be modified, and the only constraints are determined by the PHY to ensure communication with the off-the-shelf nodes.

For the experiments, we have deployed the cross-layer adaptable wireless system build upon IEEE 802.15.4 SDR in w-iLab.t Zwijnaarde which has many IEEE 802.15.4 and 802.11 nodes. The scenario is depicted in Figure 91 and shows two data collection hubs through which the data from the sensor network is collected and a central intelligence module that controls the network.

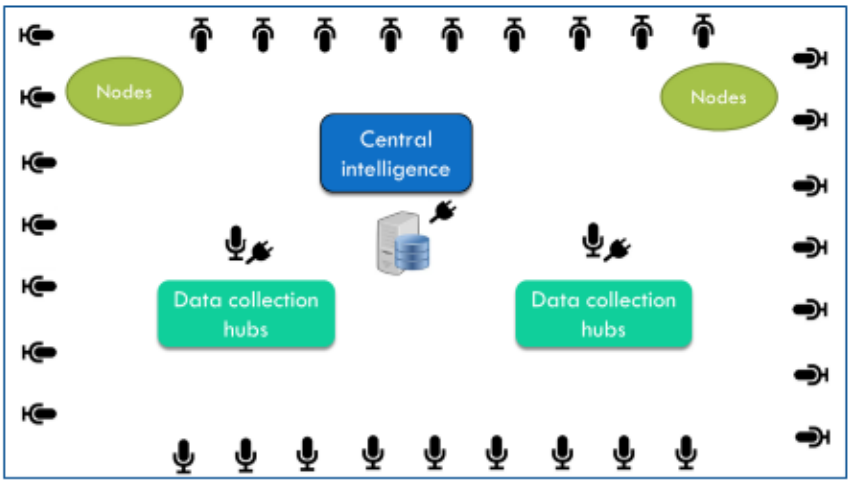


Figure 91 FACT test scenario.

In the first step of the experiment, a baseline characterization of the environment was performed where the performance of the IEEE 802.15.4 single link where 18 nodes were transmitting with a peak throughput of 250 kbps has been evaluated. As depicted by Figure 92, it can be seen that the measured performance is well below the ideal one. The drastic reduction in throughput it due to retransmissions, thus reducing these will lead to performance increase. Further experiments, to show the advantage of CLAWS for both single and multi-hop, are being carried out.

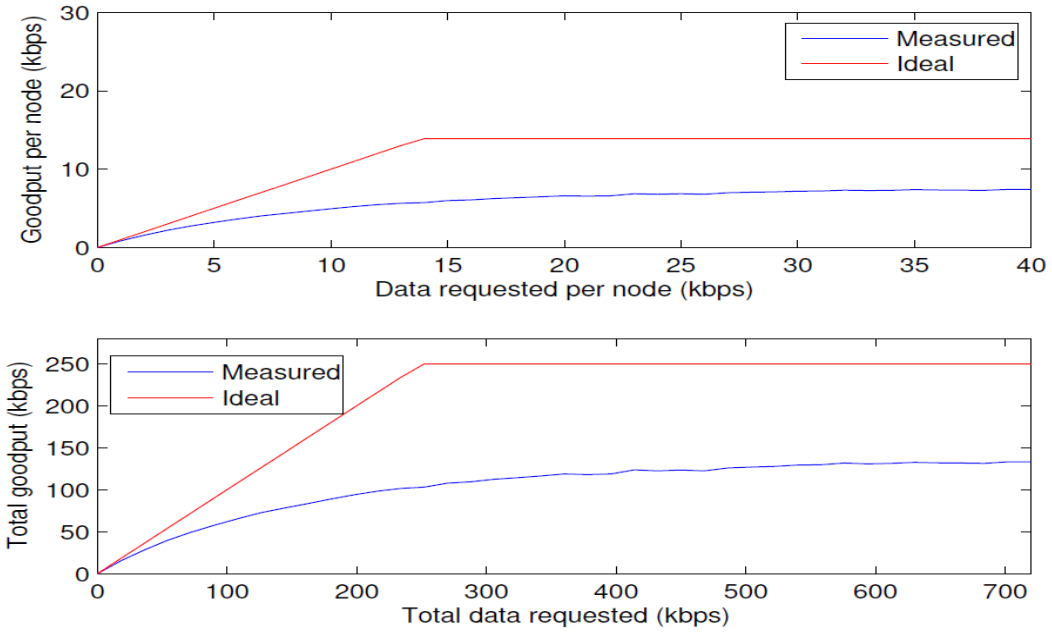


Figure 92 Single link characterization of IEEE 802.15.4.

4.7 wiHeT*

wiHeT is an OC 3 experiment proposal from Televic, Belgium. The experiment has been performed on the w.i-Lab.t testbed with the support of iMinds.

The goal of this experiment was to assess the types of WiFi roaming available on smartphones, particularly the test will verify if and how existing smartphones roam across WiFi networks with multiple access points (with the same SSID). We performed experiments to see which of the three hypotheses is true 1) the device doesn't roam, 2) the devices roams at L2 and 3) the devices roams at layers above L2.

The experimental scenario we used is presented in Figure 93. In our case, a Samsung Galaxy S2 was used. The phone is connected to a WiFi access point and slowly moves out of its coverage area into the area of another access point that has the same network SSID. The actual w-iLab.t deployment is depicted in Figure 94. The mobile phone was placed on a mobile robot having a pre-defined path. The smartphone was behaving as an iPerf client and the AP as an iPerf server and the OMF experiment controller was used. For this particular phone, our experiment shows that it does not roam (see Figure 95). Further experimentation using other phones as well as efforts to understand the current results are underway.

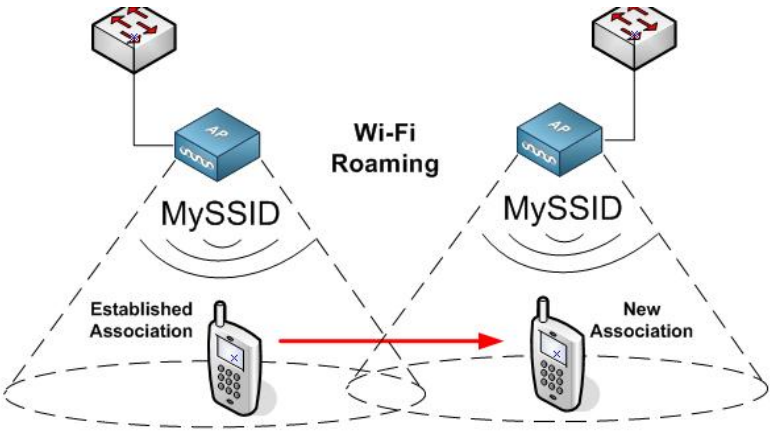


Figure 93 Smartphone roaming test scenario.

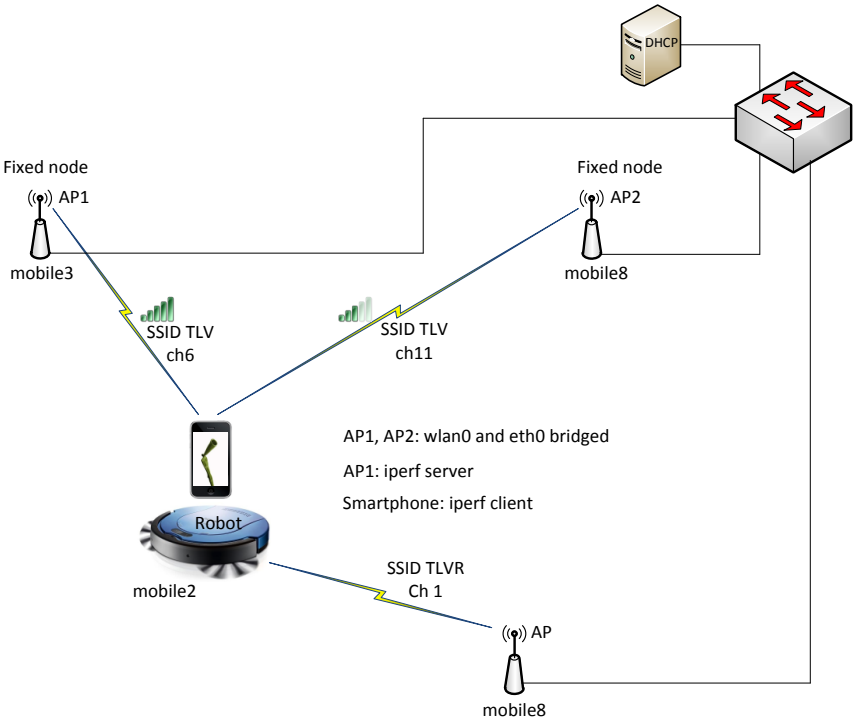


Figure 94 w-iLab.t deployment of the experiment.

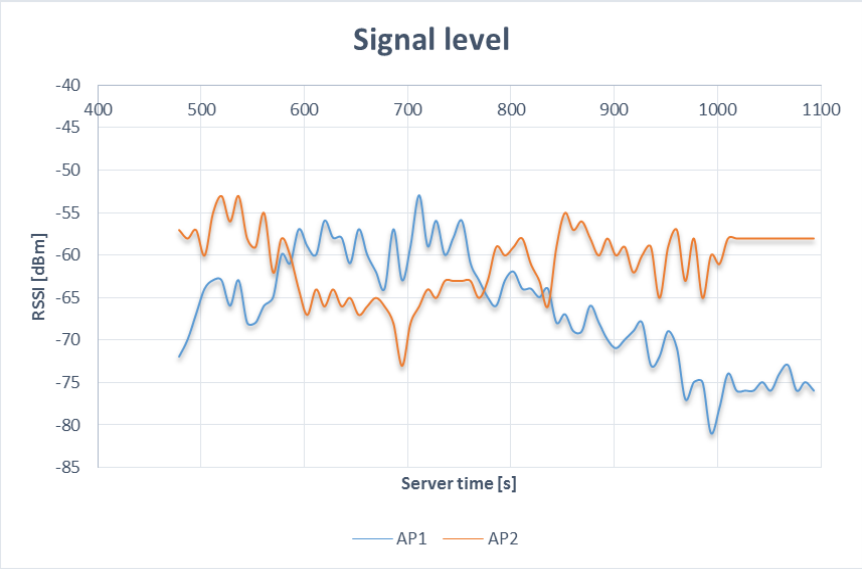


Figure 95 Samsung Galaxy S2 roaming experiment result.

5 Demand driven extensions and support derived from external experiments – Open call 3 Experiments*

This section summarizes the support and testbed extensions performed by the partners supporting OC3 experimenters. It also provides a discussion of the estimated and actual support of the core partners for these experimenters.

5.1 Support for CARE*

The CARE, Cognitive Access and Rendezvous Experiments investigate distributed algorithms for solving the blind rendezvous problem, namely the problem of establishing a control link on the same channel among secondary users of a cognitive radio system without any central coordination. Specifically, it evaluates a rendezvous protocol that guarantees to randomly select a common channel in fully decentralized environments even with asymmetrical system properties (e.g., asymmetrical channel perceptions for the different CR nodes and system clock drift).

In order to analyze the developed protocol in a practical environment, a thorough experimental evaluation was performed using a real-life testbed within the CREW framework. More specifically, the facilities of the w-iLab.t testbed are exploited.

Support was provided by iMinds to do the initial setup of the experiments in the w-iLab.t testbed, using cognitive radio equipment such as USRP devices. No hardware modifications were needed to conduct the experiments. To ensure repeatability and comparability, the experimenter can make use of the OMF experiment control framework, so no software extensions were needed to support this experiment.

5.2 Support for GAME-COG-NET*

5.2.1 Extending the JSI campus testbed*

Power update algorithm used in the GAME-COG-NET experiment had certain requirements regarding the channel gains between transmitter and receiver pairs. After initial survey of the LOG-a-TEC testbed it was noted that existing clusters do not offer enough node combinations that would fit these requirements. Because of that, two new, temporary clusters were set up at the JSI campus site for the GAME-COG-NET experiment.

5.2.1 Correcting issues with sensor node firmware*

During the course of the GAME-COG-NET experiment, several issues were discovered with the software used in LOG-a-TEC testbeds by TU Cluj-Napoca experimenters and reported to the JSI. This software included the sensor node firmware based on the "vesna-drivers" library that was running on the spectrum sensing nodes and the "vesna-alh-tools" Python library that the experimenters used to access the testbed from their computers. Reported issues were corrected where possible, or workarounds suggested where solving them during in the short term was not possible.

In total, approximately 10h of software development time were spent investigating and correcting these issues.

Software fixes developed during the GAME-COG-NET experiment were incorporated into the main software repositories used in other experiments. This ensured that future work with the LOG-a-TEC testbed will benefit from experience gained during GAME-COG-NET.

Following is a list of reported issues and a short description of their solution.

5.2.1.1 "sensing/quickSweepBin" reports a CRC error*

Problem: It was reported that a warning about a CRC error was present in the communication log when accessing the "sensing/quickSweepBin" resource on sensor nodes.

Analysis: The "sensing/quickSweepBin" resource is used to instruct a spectrum sensing node to perform a single spectrum sweep and record the RSSI for each frequency channel. The result of the spectrum sweep is returned immediately in a binary format that contains a CRC checksum to ensure data integrity. Initially, an old version of firmware was deployed to spectrum sensing nodes in the clusters used by GAME-COG-NET that had a bug in the CRC calculation routine. In spectrum sensing nodes using that firmware version, only the first half of the binary data was included in the checksum. Version of the "vesna-alh-tools" library used by GAME-COG-NET was more recent than the firmware and contained a workaround for this problem. When it detected a bad CRC calculation, it accounted for the bad CRC and printed a warning in the log to warn its user that the firmware on the sensor node needs upgrading.

Solution: Sensor nodes used in the GAME-COG-NET were upgraded to the latest firmware version.

5.2.1.2 Error retrieving device configuration*

Problem: It was reported that when querying the sensor node for the list of supported receiver configurations using the "get_config_list()" Python method, the library occasionally raises an unexpected exception.

Analysis: The "get_config_list()" Python method in the "vesna-alh-tools" library queries a sensor node resource "sensing/getConfigList". This resource returns a list of hardware configurations supported by the radio on the sensor node. The list is returned in a verbose, human readable ASCII format. The Python library parses the ASCII response and stores the list in the form of Python objects. The ASCII response from the resource does not contain a checksum for ensuring that the response was not corrupted while passing through the cluster's management network. This means that occasionally, the response received by the Python code would be corrupted. In some cases of corruption the Python code did not detect it but rather followed an unexpected execution path that finally led to an exception in an unrelated part of the code.

Solution: Python code for parsing the ASCII list of configurations was amended to be more robust in case of data corruption and always raise a CRCError exception when the response could not be parsed. During the resolution of this problem several other methods with a similar issue were amended as well.

5.2.1.3 A sensor node stops responding after several power cycles*

Problem: It was reported that one of the sensor nodes stopped responding to network requests after it several power cycles.

Analysis: VESNA sensor nodes used in the LOG-a-TEC testbeds use a proprietary low-powered mesh network solution to participate in the management network. The network stack is implemented in a separate, sealed microcontroller module on the sensor node running a proprietary firmware. This wireless network is used to setup the nodes in a cluster for an experiment. A problem we have often observed with the network module is that for an unknown reason its non-volatile flash ROM memory occasionally suffers from data corruption. This corruption typically manifests itself during a power cycle and prevents the sensor node from joining the mesh network.

Solution: On the affected sensor node the network module firmware image in flash ROM was refreshed via manual reprogramming. Given the proprietary nature of the network module and lack of support from the vendor a permanent solution to this problem is unlikely.

5.2.1.4 "slot is not empty" error reported when programming a sensor node*

Problem: When attempting to upload a spectrum sensing program to a sensor node, the node returns a "slot is not empty" error message.

Analysis: Sensor nodes in LOG-a-TEC testbeds can be programmed to perform a series of spectrum sensing frequency sweeps in different regimes. The sensing data thus produced are saved onto the SD card for later retrieval. SD card is divided into slots. When uploading a program, the user must assign a slot that will be used by the uploaded program. An error is returned when uploading if the selected slot has been previously used and has not been freed using the "sensing/freeUpDataSlot" resource. Due to a bug in the VESNA firmware library, a SD card slot could be freed while a spectrum sensing program was still writing to it. In that case a "sensing/freeUpDataSlot" request had no effect, but did not return an error either. This led to seemingly erroneous "slot is not empty" messages.

Solution: VESNA firmware library was corrected so that a "sensing/freeUpDataSlot" request for a slot that was still in use raised an error.

5.2.1.5 "failed to read SD slot header, code -1" error reported when retrieving data*

Problem: When attempting to download spectrum sensing data from a sensor node, the node returns an error message.

Analysis: The "failed to read SD slot header, code -1" error message is reported by the node when the SD card driver on the sensor node reports a hardware failure. This error has been sporadically observed in sensor nodes deployed in out-door environments. It is likely caused by unreliable connection to the SD card, possibly due to moisture or degradation of the connector on either the sensor node's or the SD card's side.

Solution: None during the course of the experiment. Next version of sensor node core will have a more robust SD card connector.

5.2.1.6 Some links between nodes seem to have unexpectedly high attenuation*

Problem: Experimenters noted that some pairs of nodes that were physically close had unusually high attenuation when using the 2.4 GHz radio.

Analysis: Sensor nodes in the LOG-a-TEC testbeds are equipped with SNE-ISMTV expansion that provides versatile radio frontends for experimentation. The 2.4 GHz version of the expansion uses an integrated transceiver CC2500 from Texas Instruments. It has been observed in several experiments that the performance of this transceiver sometimes drops significantly after being deployed in an out-door environment. Both receive sensitivity and transmit power have been observed to drop on the range of 10 to 30 dB. The exact cause of this degradation has not yet been determined, however

factors other than the transceiver integrated circuit (e.g. degradation of other components on the SNE-ISMTV, antenna, connectors, …) have been mostly ruled out.

Solution: SNE-ISMTV expansion has been replaced on affected sensor nodes.

- **Total time on testbed**

65 hours total reserved time on testbed

45 days between first and last reservation

- **Number of experiments**

approx. 100

- **Average duration of experiment**

10 minutes

- **Number of nodes used**

8 nodes

5.3 Support for MUCO*

The main goal of the CREW-MUCO project was to evaluate hardware components developed by AED Engineering in the real deployment. The hardware platform is composed from a FPGA and a PCB for IEEE 802.14.4 Wireless Sensor Networks and is designed to act as the sensor network coordinator node. It is able to handle several channels within the 2.4GHz band simultaneously.

To be able to conduct the experiments the hardware needed to be deployed at the TUB testbed. During that process the face to face meeting has been conducted to make sure that the device is correctly placed and is available for remote experimentation. During that meeting the TWIST testbed was again introduced and the usage tutorials where given. Also all required access accounts where setup. This means access to the TWIST testbed as well as VPN access to the additional subnet where the coordinator device was located.

The device was placed centrally in our testbed ensure connectivity range with as many nodes as possible. It is connected to our backbone network over Ethernet. Additionally the Power Distribution Unit (PDU) and a laptop where deployed. A PDU allows for remote power control of the devices, which allows the user to power the coordinator node and the laptop on/off on demand. The laptop will wake up when there is a power supply attached and allows for windows remote connection to be setup. The additional laptop is necessary for debugging purposes. It was supplied by AED and has the whole FPGA development tool-chain installed. This allows for remote reprogramming of the coordinator device as well as accessing the debug information via the Chipscope interface of the FPGA.

Additional support was provided during the initial stage of usage of the TWIST testbed. Main problems where connected to finding a free reservation slot to use the testbed, as it was heavily used by other experimenters, the biggest one was connected with EVARILOS project and the organized Open Challenge that required manual scheduling of the experiments. Some support was also provided with regard to the developing application for the sensor network testbed. It required hints and pointing to the TinyOS programming documentation.

In summary, the biggest effort was required for the deployment of the hardware in the testbed. Although, the infrastructure was good prepared, it still requires the user to gain full knowledge of the internal communications. It is not enough to connect all the devices together, which is easy, it is also necessary that the user understands how he or she can use it in the practical scenarios. It all can be solved with one day efficient face to face meeting in the testbed. Such a meeting can be conducted with travel of the experimenter joint with other meetings to reduce the cost and effort, but often it is most efficient way of solving problems.

5.4 Support for picoMESH*

The goal of the experiment is to prove that the OLSR (IETF RFC3626) implementation provided by PicoTCP fits a scenario where a mesh topology based on 802.15.4 is challenged by multiple issues (i.e. nodes with a limited visibility or partially hidden, nodes joining and leaving the topology with no notice). When all the nodes are running the same firmware, addressing and routing is automatically configured so that all the nodes that are not in direct visibility can still be addressed, and the communication is guaranteed across the whole testbed with no intervention needed from outside. Moreover, the network is capable to react reasonably quick to topology changes such as nodes joining and leaving the network, by automatically adjusting the routing tables accordingly.

The consistency of the network is verified by exchanging data among nodes while forcing sudden topology changes, and evaluating the reaction time required by the nodes to cope with the introduced modifications at run time. Furthermore, the routes can be affected by RSSI with a simple cross-layer modification to the OLSR module, in order to take into account the signal quality while calculating the best routes among each pair of nodes in the network. OLSR is a standardized protocol which makes it possible to expand the topology across heterogeneous devices and physical layers, for example to joint multiple sections of the mesh network together even if they are physically placed in remote locations.

The number of nodes has been scaled up from 2 to 30 nodes within the w-iLab.t testbed while testing the different aspects of the implementation:

- verify if stack is operational on all nodes
- monitor the cognitive adaption of the network.
- test the multihop routing capabilities of the nodes , send ping from first to last node.

Identifying every node in a unique way and fine-tuning the delay between olsr HELLO, TC and MID per experiment can be quite time-consuming if it needs to be done manually (=building a unique firmware per node and per experiment). The w-iLab.t testbed already has support to parameterize the experiment as it can derive from one firmware template a unique firmware for every node based on the parameters assigned to the experiment (=1 build).
The support was mainly related to the use of this parameterization tool on w-iLab.t.

Some experiment statistics:

- Number of reservation slots: 223
- Average duration of a reservation entry: 15 minutes
- Average number of fixed wireless nodes: 29
- Average number of mobile nodes: 0
- Total duration of reservation entries: 18 days

5.5 Support for SIRI*

The SIRI experiment examines the benefits of using the Radio Environmental Maps (REMs) in small-cell optimization scenarios. The core of this examination is a generic, modular, and flexible REM prototype implementation with on-field spectrum measurement devices to provide real-time environment data, and a rich set of statistical inference algorithms for, propagation modeling, single/multi source localization, Radio Interference Fields (RIFs) estimation, temporal and spatial channel occupancy estimation, etc. In particular, the REM data serves as a feedback to the small-cells radio resource management (RRM), which calculates the optimal frequency, bandwidth and transmit power values for the small-cells. The synergy between REM and RRM can be especially beneficial in femto-cell scenarios due to the unavailability of clearly defined interfaces, techniques and solutions to

manage the installation and the optimization of femto-cell operation. This experiment includes work conducted on three CREW facilities, that of TCD, JSI, and iMinds.

The examination of the REM-assisted RRM optimization of small-cell devices on TCD's testbed is comprised of the following hardware/software components:

- Laptops to run the REM backend with the toolboxes for data processing/analysis, along with the REM GUI.
- In total 10+ spectrum sensors to perform the real-time spectrum measurements as an input to the REM processing/inference algorithms.
- Two pairs of small cell reconfigurable devices (i.e. USRP2 devices run under the IRIS software platform) which utilize the REM based RRM to perform real time communication
- The work exhibits several REM and RRM benefits, including:
- Real-time performance assessment of the REM backend, in terms of single-/multi- source localization, estimation of the propagation model parameters, estimation of the interference levels in the environment, estimations on statistical/spatial spectrum usage.
- The ability to calculate the most optimal frequency/bandwidth/power parameters of the small-cell devices, based on the estimated REM data: available channels, propagation model estimates, locations of estimated transmitters, interference levels estimations.
- Dynamic configuration of communication parameters to respond to changes in the environment including the activation of a co-band interferer. The RRM toolbox of the REM Manager will dynamically perform a new frequency/ bandwidth/ power allocation, so the small-cell devices can adapt to new environment conditions and continue the communication in the most optimal manner.
- Dynamic monitoring of all of the environment changes via the REM GUI, including appearance of new emissions, changes in propagation characteristics etc.
- Continuation of this work on JSI's testbed facilities tested performance of wireless transmitter localization based on Received Signal Strength (RSS) in practice. Two distinctive experiments were defined based on the capabilities of the platform and the targeted problem:
- Experiment 1, focusing on localization in the TV band and
- Experiment 2, performing localization in the ISM band.

Both experiments assumed simplified propagation model under log-normal uncorrelated shadowing and implement Maximum Likelihood (ML) localization algorithm. The propagation model parameters were estimated using Least Squares (LS) parameter fitting. Both experiments tried to use all available sensors in the testbed for experimentation. However, at the moment of experimentation, only few sensors were available for our purposes thus limiting the localization performance as well as the scope of the experiments. The obtained results show the performance of the localization algorithm under realistic settings with limited number of measuring sensors. It is evident that the performance of the transmitter localization is significantly degraded when the number of the measurement sensors is low. Moreover, the localization performance is very susceptible to measurement bias and environment dynamics under such circumstances.

Real-time and reliable spectrum sensing constitutes a particularly important component of the Cognitive Radio (CR) and Radio Environmental Awareness (REA) aspects and technologies. This experiment evaluates key features of the spectrum sensing process (e.g. primary user detection, sensing complexity, sensing agility and swiftness, cooperation/spatial diversity gain etc.) by exploiting the possibilities provided by the iMinds/Imec testbed facilities.

The experiment focuses on evaluating and comparing the performance of different sensing techniques, varying from simple detection techniques (i.e. the energy detector) up to more complex techniques that incorporate the Estimated Noise Power (ENP) [36] approach and the Goodness-of-fit Higher Order Statistics Testing (GHOST) [37][38] detection approach. Moreover, the proposal focuses on the spatial diversity i.e. cooperative aspects of the presented sensing techniques.

In accordance with the above, it is expected that the obtained results, will ratify the theoretical performance of the ENP and GHOST sensing techniques. Moreover, it is expected that the results will

facilitate an in depth analysis of the sensing techniques behavior in terms of their complexity, spatial diversity gain, detection probability etc.

The SIRI experiments have recently also started on the iMinds testbed. Only initial support for creating accounts and pointing to the relevant documentation has been provided thus far.

Despite the large scale of the work conducted by the UKIM team, support required has been minimal. Each facility provider has spent a few days interacting with the experimenters to support the initial use of testbeds. This setup support mainly consisted of the creation of testbed accounts and introduction to facility use for the SIRI project. In the case of TCD, this support included a broader introduction to the Iris software defined radio system over the course of a several days, during which a member of the UKIM was able to physically be in Dublin. TCD has also spent additional time interacting with UKIM to gather results, prepare deliverables, and organize a final demonstration of the work. No additional hardware was required for the SIRI experiment, but the updates to TCD's testbed, discussed above, were pursued partly based on the experience of UKIM during the SIRI experiment and subsequent feedback related to facility usability.

5.6 Support for FACT*

The FACT experiment (Furthering Airborne Cognitive Technologies) aims to develop a novel multi-hop cognitive data-link for use in unmanned aerial systems (UAS). The solution uses cognitive radio equipment to improve this data-link by changing the technology, the channel and the transmit power. Since UAS testbeds do not yet exist, the w-iLab.t testbed is used to conduct the experiments in 2D. The mobile robot simulates the UAS, while the fixed nodes generate the interference.

Support was provided by iMinds to use the 802.15.4 sensor nodes in the testbed, as well as the mobile robots. Some issues were discovered during the experiment concerning the stability of the USB devices on some wireless nodes. This issue is being investigated at time of writing this document, but is most probably related to the drivers of the used operating system. This is very useful, as this will increase the stability of future sensor experiments on the w-iLab.t testbed. Additional support was provided to install several new USRP X310's in the testbed, along with a dedicated desktop computer (with 4x PCI-E).

Some experiment statistics:

- Number of reservation slots: 18
- Average duration of a reservation entry: 24 hours
- Average number of fixed wireless nodes: 14
- Average number of mobile nodes: 1
- Total duration of reservation entries: 18 days

5.7 Support for wiHeT*

The goal of this experiment is to get a better image of WiFi roaming on smartphones. We want to test if and how existing smartphones roam across WiFi networks with multiple access points (with the same SSID).

For this experiment, the wireless nodes at iMinds w-iLab.t are used. Two kinds of nodes are used in this experiment: wireless nodes at a fixed location and also mobile nodes. The mobile nodes can be configured in exactly the same way as the fixed nodes, but with the possibility to define a path which the mobile node will follow during the experiment.

Technical support was given by iMinds to setup this experiment:

- General support to make use of the facility. This includes creating an account, reserving some nodes and deploying an operating system on them.
- Hardware extensions: deployment of smartphone(s) on the mobile nodes in the w-iLab.t.
- Support to use the mobility framework of the testbed. This includes practical issues such as undocking the robot, drawing a specific mobility pattern and recharging of the battery.
- The experiment also required setting up several access points on the fixed nodes of the testbed. Since all access points have to be in the same subnet, and given the fact that the wireless nodes have no separate experiment network interface, the access points were configured as bridges on the testbed control network.
- iMinds also assisted in creating an OMF-enabled application to run the wiHeT experiment. This application is basically a 'wrapper' around the existing iPerf binaries.
- All technical issues related to configuring the smartphones and reading RSSI values from it, were done by Televic.
- Future hardware extensions planned: installation of different kinds of smartphones from different vendors to investigate their roaming behavior.

Some experiment statistics:

- Number of reservation slots: 88
- Average duration of a reservation entry: 7.8 hours
- Average number of fixed wireless nodes: 3
- Average number of mobile nodes: 3
- Total duration of reservation entries: 28 days
-

5.8 Discussion*

The CREW project accepted 7 out of 10 Open Call 3 proposals. The decision was based on the 2-page experiment description requested through the OC announcement. The 3 proposals that were rejected were either out of scope or not feasible within the CREW offering. From the 7 accepted experiments, two were particularly successful, one resulting in a journal submission and another one resulting in two conference demos. The 4 other experiments are still running and at least 3 will result in a final demo.

It is in general very hard to estimate the support needed for the experiments as the description of the experiment is only approximate as most of the experimenters did not use the facility before and their skill with hand-on work is sometimes limited. Second, the experimenters are not familiar with the testbeds. In the case of the OC3 experimenters, the summary of estimated and planned efforts is given in Table 12. It can be seen that for some experiments such as CARE, MUCO and SIRI, the actual overall support was lower than initially planned while in other experiments such as wiHeT, GAME-COG-NET has been higher. However, in our experience the support is not necessarily proportional to the complexity or the final output of the experiment. While CARE and MUCO have a slower progress, SIRI's results are impressive and have resulted in demos at several conferences while a paper is under preparation. Also, this was the only OC3 experiment that uses three testbeds – all the others use one or two. picoMESH for instance is a challenging experiment and required more support than estimated from one facility and none from another. This is because the experiment required serious debugging and took longer to have on the iMinds facility. Since the effort was much higher than initially estimated on both sides, the SME performing it might not continue on the second facility – TWIST.

Table 12 Estimated and actual effort spent for OC3 partners.

Partner	OC3 exp	Estimated	Actual
iMinds	CARE	0.5 MM (10 days)	3 days
iMinds	picoMESH	0.5 MM (10 days)	13 days
TUB	picoMESH	0.5 MM (10 days)	0 days
iMinds	FACT	0.5 MM (10 days)	12 days
IMEC	FACT	0.5 MM (10 days)	12 days
iMinds	wiHeT	1 MM (20 days)	30 days
JSI	GAME-GOG-NET	1 MM (20 days)	30 days
TUB	MUCO	1 MM	10 days
TCD	SIRI	1 MM (20 days)	20 days
IMEC	SIRI	0.5 MM (10 days)	0 days
JSI	SIRI	0.5 MM (10 days)	2 days
iMinds	SIRI	0.5 MM (10 days)	3 days

6 The CREW academy*

In 2014, CREW organized a new round of training days following the similar event in 2013. This time, the training days were dedicated to the OC3 experimenters. Out of 7 Open Call 3 experimenters, 5 participated to the training days, which we consider a success given that this call was not funded. One of the missing participants actually attended the training days already in 2013. All the tutorials given during the training days were filmed and made available on YouTube on the dedicated CREW channel (, https://www.youtube.com/channel/UCCur6L72i-XPkm5gKzqBAow).

The voice quality of the initial recording of the tutorials was poor, thus we considered doing a voice over version, which is now available, online. To determine the best tool for performing the voice-over, we made a survey of desktop screen recorders for different OSs with voice recording capability so that we use them to prepare voice over presentation for the CREW open access phase.

The final list of recorders corresponding to different operating systems is given in the following table:

Linux OS	Windows	MAC OS
RecordMyDesktop, SimpleScreenRecorder	CamStudio RECORDER	quicktime application

The following recommendations for the recording process were made:

1. Use high quality audio microphones rather than built in microphones.

2. Capture audio using bit rate 192kps and sampling rate 22050 sps.

3. Start screen recording and capture your presentation (at least 1280 pixel x 720 pixel) with the highest quality (e.g. with out compression if possible or use MP4 container with H.264 video codec).

4. After the recording is finished, insert pages, texts and application outputs using a video editor (for example Windows Movie Maker or similar programs).

5. All presentations must start with the CREW Island page attached to this email.

6. Finally, upload your presentation on myMinds Documents/CREWall-info relevant for core and open call partners.

7 The CREW participation in the Ofcom TV white space trial*

The Ofcom TV white space field trials with various devices were significantly delayed due to the time needed for Ofcom to qualify the geolocation databases. The first datatbase was only qualified in late May 2014, three further databases following in June 2014, and the first field tests with devices operating in TV white spaces provided by NICT only started in July 2014. Most of the trials so far, however, have been carried out by the Carlson wireless devices, which are, inter alia, used for the provision of continuous broadband wireless access in the Strand Campus of King's College using via white spaces backhaul links. Other trials with Carlson devices include testing long distance links for backhaul/PPDR use cases and some challenging indoor links within the Strand Campus of King's College. These have so far been only conducted at various locations around London while future trials (from November onwards) may also move to other cities in UK represented in the Acropolis consortium, e.g. Guildford and York.

Most of the other devices foreseen for Ofcom trials are still being finalized, e.g. KTS/Sinecom devices require some further work on the interaction with the database, and on the Eurecom devices there is still some work on the amplification/mixing/filtering. VESNA SNE-ESHTER, as a candidate device for advanced spectrum sensing in UHF band from CREW, has meanwhile been finalized and tested by JSI, and is ready for the deployment in one of the forthcoming trials, following some final adaptations to support long-term standalone monitoring of the spectrum usage by the various experimental transmissions.

From the CREW perspective, however, there have already been some DVB-T and PMSE coexistence tests carried out as part of Ofcom trials by the OC2 experimenter Instituto de Telecomunicações (IT) from Portugal. Although IT does not receive CREW funding for participation in the Ofcom trial, they can leverage on results from the CREW open call 2 experiment and there is still an strong interaction between CREW and IT via CRS-i.

8 FIRE Support Actions*

Initial contribution to FIRE were listed in deliverable D5.1. This section provides an update to these FIRE support actions. In addition to the activities listed there, following support was given to FIRE by CREW to following non-limitative list of events.

8.1 Attendance to FIRE events*

Support was/will be given within the FIRE context by CREW members actively participating in the following non-limitative list of FIRE specific events:

- FIRE week in Poznan (October 2011)
- Future Internet Week in Aalborg (May 2012)
- FIRE engineering workshop in Ghent (November 2012)
- CREW training days in Brussels (February 2013)
- Future Internet Assembly in Dublin (May 2013)
- Future Network and Mobile Summit in Lisbon (July 2013)
- Workshop on Cognitive Radio in Lisbon (September 2013)
- *ICT 2013 in Vilnius (November 2013) – the CREW booth won the award in cluster 4 at this conference
- *CREW training days 2nd edition in Ghent (January 2014)
- *Future Internet Assembly, Athens, Greece (March 2014)
- *European Conference on Future Internet, Munich, Germany (September 2014).
- We also attended several FIRE board and FIRE Forum meeting (Ghent October 2013, Munich September 2014)

An up-to-date overview of future and past events is available directly at the CREW website, via or http://www.crew-project.eu/pastevents respectively.

Attendance to this FIRE related events often also coincides with presentations and/or publications at these events. They are listed in deliverable D8.4 (for dissemination in Year 3) and for Year 4 an overview will given in the WP8 presentation of the Year 4 review meeting (19 November 2014, Ljubljana).

8.2 FIRE brochure*

CREW contributed to the latest version of the FIRE brochure. The FIRE brochure is distributed during different FIRE events and holds an overview of the FIRE facility projects, and the infrastructure that is made available.

An electronic version of the brochure can be downloaded from the FIRE website: http://www.ict-fire.eu/home/publications.html .

* In January 2014, we sent another update for the Brochure. In February, an update for the FIRE Magazine has been sent. The CREW consortium also provided use cases of CREW for SMEs for the FUSION web site:

http://www.sme4fire.eu/index.php/resources/crewusecases/tub

http://www.sme4fire.eu/index.php/resources/crewusecases/jsi

http://www.sme4fire.eu/index.php/resources/crewusecases/iminds-wings

9 Conclusions

In this document we have described the final demand-driven extensions of the CREW federation and FIRE support actions.

We explained how the federated CREW test facilities have been extended with a second set of new functionality which has been defined in a demand-driven and open way based on the gaps identified, for both the internal use cases (WP6) and the experiments of Open Call 2 (WP7).

To this end, we extended the Connectivity Brokerage Framework for better usage in different scenarios and for improved usability in specific cognitive radio scenarios. A suitable database system was selected and a extension with some IEEE 1900.6 compatible parts was made. Implementation of the Connectivity Brokerage Framework was then also investigated and tested for the w-iLab.t and Log-a-tec testbed.

Furthermore, a framework (ProtoStack/ Crime) that allows composing communication services in a dynamic way was proposed and different optimizations were performed within the different testbeds (e.g. OMF, IRIS, GRASS-RaPlat, USRP sensing engine improvement etc.).

To conclude the document, the specific support actions required for the Open Call 2 experiments were also described, as well as a short update on the FIRE support actions.

** The original deliverable submitted at the end of Y3 has been updated at the end of Y4 with a series of testbed extensions that were demand driven or planned and with information referring to Open Call 3. All the new sections and subsections have a * at the end of the name so that they can easily be identified in the table of contents and throughout the document.*

This updated deliverable has reported on the updates of the CREW portal, on the way it improved benchmarking functionality to provide a better understanding of the environment in which an experiment is carried out. Then, the Connectivity Agent has then been extended to support dynamic discovery and connection setup and the ProtoStack tool has been extended with learning functionality. Additional usability improvements have been performed for Iris and the redesign of the SNE-ISMUHF-TV has been completed and the new boards evaluated. Improvements to the wireless management network of the LOG-a-TEC testbed have also been performed as a result of external demand.

The CREW-GENI collaboration continued also in Y4 of the project with further work on the ontology and its integration into TaSoR and a paper preparation.

All OC2 demos needed various level of support in Y4 in view of the final review. We also provide a report on the OC3 experiments and the support we provided, each in separate sections. Finally, we report on the CREW Academy, the planned participation in the OfCom trial and the participation in FIRE activities.

Bibliography

[1] Plets, D., Joseph, W., Vanhecke, K., Tanghe, E. and Martens, L., "Coverage Prediction and Optimization Algorithms for Indoor Environments," *EURASIP Journal on Wireless Communications and Networking, Special Issue on Radio Propagation, Channel Modeling, and Wireless, Channel Simulation Tools for Heterogeneous Networking Evaluation*, vol. 1, 2012.

[2] SUrrogate MOdelling Tool Box, http://sumowiki.intec.ugent.be/Main_Page, last accessed on 27th of Sep, 2013

[3] Rabaey, J et al., "Connectivity Brokerage - Enabling Seamless Cooperation" in *Wireless Networks*. 2010.

[4] Parsa, S. and Banerjee, A., "Design and Implementation Guideline for the Connectivity Brokerage Distributed Repository (CBDR)", Available: https://bitbucket.org/aparsa/connectivitybroker, 2010. Berkeley Wireless Research Center (BWRC).

[5] Polastre, J., Szewczyk, R. And Culler, D., **"**Telos: enabling ultra-low power wireless research.", *IPSN '05: Proc. of the 4th international symposium on Information processing in sensor networks*, Los Angeles, California, US.

[6] Levis, P. et al., "T2: A Second Generation OS For Embedded Sensor.", Telecommunication Networks Group, Technische Universitaet Berlin, 2005.

[7] Ingels, M. et al, **"**A 5mm2 40nm LP CMOS 0.1-to-3GHz multistandard transceiver", in *2010 IEEE International Solid-State Circuits Conference ISSCC*, 2010.

[8] Nicollet, E., Pothin, S. and Sanchez, A., **"**Transceiver Facility Specification.", Wireless Innovation Forum, 2009. "SDRF-08-S-0008-V1_0_0_Transceiver_Facility_Specification.pdf", "http://groups.winnforum.org/p/cm/ld/fid=85".

[9] Latre, S., Van de Meerssche, W., Melis, S., Papadimitriou, D., De Turck, F. and Demeester, P., "Automated management of network experiments and user behaviour emulation on large scale testbed facilities", in *Network and Service Management (CNSM)*, 2010 Niagara Falls.

[10] Fortuna, C., "Dynamic Composition of Communication Services". Ljubljana, Slovenia : Jozef Stefan International Postgraduate School, 2013.

[11] OMF 6 Documentation. [Online] http://omf.mytestbed.net/projects/omf6/wiki/Wiki

[12] Dunkels, A., Osterlind, F. and He, Z., "An Adaptive Communication Architecture for Wireless Sensor Networks", in *Proceedings of the ACM Conference on Embedded Networked Sensor Systems*, 335–349, 2007, Sydney, Australia.

[13] Fortuna, C. and Mohorcic, M., "Dynamic composition of services for end-to-end information transport", *IEEE Wireless Communications Magazine*, 2009, Vol. 16.

[14] Tektronix, Fundamentals of Real-Time Spectrum Analysis,. (Tektronix, 2009), p. 7

[15] Liu et al. EURASIP Journal on Wireless Communications and Networking 2013, 2013:228

[16] Plets, D., Joseph, W., Vanhecke, K., Tanghe, E. and Martens, L., "Coverage Prediction and Optimization Algorithms for Indoor Environments," *EURASIP Journal on Wireless Communications and Networking, Special Issue on Radio Propagation, Channel Modeling, and Wireless, Channel Simulation Tools for Heterogeneous Networking Evaluation*, vol. 1, 2012.

[17] SUrrogate MOdelling Tool Box, http://sumowiki.intec.ugent.be/Main_Page, last accessed on 27th of Sep, 2013

[18] Rabaey, J et al., "Connectivity Brokerage - Enabling Seamless Cooperation" in *Wireless Networks*. 2010.

[19] Parsa, S. and Banerjee, A., "Design and Implementation Guideline for the Connectivity Brokerage Distributed Repository (CBDR)", Available: https://bitbucket.org/aparsa/connectivitybroker, 2010. Berkeley Wireless Research Center (BWRC).

[20] Polastre, J., Szewczyk, R. And Culler, D., "Telos: enabling ultra-low power wireless research.", *IPSN '05: Proc. of the 4th international symposium on Information processing in sensor networks*, Los Angeles, California, US.

[21] Levis, P. et al., "T2: A Second Generation OS For Embedded Sensor.", Telecommunication Networks Group, Technische Universitaet Berlin, 2005.

[22] Ingels, M. et al, "A 5mm2 40nm LP CMOS 0.1-to-3GHz multistandard transceiver", in *2010 IEEE International Solid-State Circuits Conference ISSCC*, 2010.

[23] Nicollet, E., Pothin, S. and Sanchez, A., "Transceiver Facility Specification.", Wireless Innovation Forum, 2009. "SDRF-08-S-0008-V1_0_0_Transceiver_Facility_Specification.pdf", "http://groups.winnforum.org/p/cm/ld/fid=85".

[24] Latre, S., Van de Meerssche, W., Melis, S., Papadimitriou, D., De Turck, F. and Demeester, P., "Automated management of network experiments and user behaviour emulation on large scale testbed facilities", in *Network and Service Management (CNSM)*, 2010 Niagara Falls.

[25] Fortuna, C., "Dynamic Composition of Communication Services". Ljubljana, Slovenia : Jozef Stefan International Postgraduate School, 2013.

[26] OMF 6 Documentation. [Online] http://omf.mytestbed.net/projects/omf6/wiki/Wiki

[27] Dunkels, A., Osterlind, F. and He, Z., "An Adaptive Communication Architecture for Wireless Sensor Networks", in *Proceedings of the ACM Conference on Embedded Networked Sensor Systems*, 335–349, 2007, Sydney, Australia.

[28] Fortuna, C. and Mohorcic, M., "Dynamic composition of services for end-to-end information transport", *IEEE Wireless Communications Magazine*, 2009, Vol. 16.

[29] Tektronix, Fundamentals of Real-Time Spectrum Analysis,. (Tektronix, 2009), p. 7

[30] Liu et al. EURASIP Journal on Wireless Communications and Networking 2013, 2013:228

[31] http://wireless.kernel.org/en/users/Drivers/ath9k/spectral_scan/

[32] https://wiki.freebsd.org/dev/ath_hal(4)/SpectralScan

[33] https://github.com/simonwunderlich/FFT_eval

[34] http://fosiao.com/content/Zigbee-and-wifi-rf-channels

[35] http://standards.ieee.org/getieee802/download/802.15.4d-2009.pdf

[36] V. Rakovic, V. Pavlovska, V. Atanasovski and L. Gavrilovska, "Cooperative spectrum sensing based on noise power estimation," 2013 International Symposium on Wireless Personal Multimedia Communications, June, 2013, Atlantic City, USA.

[37] D. Denkovski, V. Atanasovski and L. Gavrilovska, "GHOST: Efficient Goodness-of-fit HOS Testing Signal Detector for Cognitive Radio Networks," IEEE ICC 2012, Ottawa, Canada, June 10-15, 2012.

[38] D. Denkovski, V. Atanasovski and L. Gavrilovska, "HOS based goodness-of-fit testing signal detection," IEEE Communications Letters, Vol. 16, Issue 3, pp. 310-313, March 2012.

[39] Ranveer Chandra, Jitendra Padhye, Lenin Ravindranath, Alec Wolman "Beacon-Stuffing: WiFi Without Associations", 2007.

[40] Student project: Service Announcement Using Beacon Stuffing In 802.11 Networks http://www.tkn.tu-berlin.de/menue/tknteaching/student_projects/project_summaries/service_announcement_using_beacon_stuffing_in_80211_networks/

[41] Alec Woo, Terence Tong, and David Culler. 2003. Taming the underlying challenges of reliable multihop routing in sensor networks. In Proceedings of the 1st international conference on Embedded networked sensor systems. ACM, 14–27.

[42] Arsalan Tavakoli and David Culler. 2009. Hydro: A hybrid routing protocol for low-power and lossy net- works. (2009). http://tools.ietf.org/html/draft-tavakoli-hydro-01

[43] Kannan Srinivasan, Prabal Dutta, Arsalan Tavakoli, and Philip Levis. 2010. An empirical study of low- power wireless. ACM Transactions on Sensor Networks (TOSN) 6, 2 (2010), 16.

[44] E Kim, D Kaspar, C Gomez, and C Bormann. 2011. Problem Statement and Requirements for 6LoWPAN Routing. (2011). http://tools.ietf.org/html/draft-tavakoli-hydro-01

[45] Kannan Srinivasan, Prabal Dutta, Arsalan Tavakoli, and Philip Levis. 2010. An empirical study of low- power wireless. ACM Transactions on Sensor Networks (TOSN) 6, 2 (2010), 16.

[46] Gomez Carles, Boix Antoni, and Paradells Josep. 2010. Impact of LQI-based routing metrics on the perfor- mance of a one-to-one routing protocol for IEEE 802.15. 4 multihop networks. EURASIP Journal on Wireless Communications and Networking 2010 (2010).

[47] Nouha Baccour, Anis Koubaa, Luca Mottola, Marco Antonio Zuniga, Habib Youssef, Carlo Alberto Boano, and Mario Alves. 2012. Radio link quality estimation in wireless sensor networks: a survey. ACM Trans- actions on Sensor Networks (TOSN) 8, 4 (2012), 34.

[48] LuisSanchez, LuisMunoz, Jose Antonio Galache, PabloSotres, Juan R Santana, Veronica Gutierrez, Rajiv Ramdhany, Alex Gluhak, Srdjan Krco, Evangelos Theodoridis, and others. 2013. SmartSantander: IoT Experimentation over a Smart City Testbed. Computer Networks (2013).

[49] P. De Valck, I. Moerman, D. Croce, F. Giuliano, I. Tinnirello, D. Garlisi, E. De Poorter and B. Jooris, "Exploiting Programmable Architectures for WiFi/ZigBee Inter-Technology Cooperation," Eurasip, 2014.

[50] Denkovski, Daniel, et al. "Small-cells radio resource management based on Radio Environmental Maps." Computer Communications Workshops (INFOCOM WKSHPS), 2014 IEEE Conference on. IEEE, 2014.

[51] Cinkelj, Justin, et al. "Design Trade-offs for the Wireless Management Networks of Constraint Device Testbeds", ISWCS 2014, Barcelona, Spain.