Cognitive Radio Experimentation World

SEVENTH FRAMEWORK PROGRAMME

Project Deliverable D3.1
Basic operational platform

Contractual date of delivery:	30-09-11
Actual date of delivery:	30-09-11
Beneficiaries:	IBBT – IMEC – TCD – TUB – TUD – TCS – EADS
Lead beneficiary:	TCF
Authors:	Danny Finn (TCD), Justin Tallon (TCD), Luiz DaSilva (TCD), Jono Vanhie - Van Gerwen (IBBT), Stefan Bouckaert (IBBT), Ingrid Moerman (IBBT), Christoph Heller (EADS), Alejandro Sanchez (TCS), David Depierre (TCS), Sofie Pollin (imec), Peter VanWesemael (imec), Jan Hauer (TUB), Daniel Willkomm (TUB), Mikolaj Chwalisz (TUB), Nicola Michailow (TUD), Carolina Fortuna (JSI), Zoltan Padrah (JSI), Marko Mihelin (JSI)
Reviewers:	Daniel Willkomm (TUB), Jan Hauer (TUB), Mikolaj Chwalisz (TUB), Danny Finn (TCD)
Workpackage:	WP3 – Creating the Federation
Estimated person months:	42
Nature:	R
Dissemination level:	PU
Version	1.0

Abstract: This public document gives a detailed description of the functionality of the first operational federated platform. This platform will include a first version of the PORTAL and supports intra-country component combinations and basic data collection. This deliverable reports on the activities performed in all tasks of this work package.

Keywords: network testbeds, federation, wireless networks, cognitive radio, cognitive network, functionalities, capabilities, components, combination, interface, data format, portal, guidelines.

Revision history

Version	Date	Author	Description
0.0	01/06/2011	Alejandro SANCHEZ (TCS)	Initial draft from CREW documents template
0.1	01/08/2011	Alejandro SANCHEZ (TCS)	ToC updated for inclusion of common data format following Berlin meeting
0.2	08/08/2011	Alejandro SANCHEZ (TCS), David DEPIERRE (TCS)	TCF contributions added: • Added sections 3.7, 6.4.1 • New chapter 2 inserted • Section 6.3.1 filled
0.2.1	9/08/2011	Stefan Bouckaert (IBBT), Jono Vanhie-Van Gerwen (IBBT)	Added contributions related to the IBBT testbed
0.2.2	11/08/2011	Christoph Heller (EADS)	First paragraphs of conclusion added
0.2.3	16/08/2011	Alejandro SANCHEZ (TCS)	Merged contributions from IBBT: • Chapter 5 "Common portal" • Section 6.4.2 added Merged contributions from TCD and TUB
0.3	17/08/2011	Alejandro SANCHEZ (TCS)	Latest conclusion from EADS included. Version published.
0.3.1	19/08/2011	Sofie Pollin (imec)	Extended section 3.4 for imec
0.3.2	24/08/2011	Nicola Michailow (TUD)	Filled section 3.3
0.4	31/08/2011	Alejandro SANCHEZ (TCS)	New Federation figure Separated "Common data formats" document merged within chapter 3. List of acronyms reviewed Appendix added, references added
1.0	27/09/2011	Alejandro SANCHEZ (TCS)	Final version

Executive Summary

D3.1 establishes the first operational basic federation platform following the efforts for integration of the different testbeds carried out during the first year. The document compiles and describes all the functionalities that the consortium makes available for external partners' usage. The main features along with some usage guidelines are provided in the document for each individual testbed. Similarly for individual or standalone components, i.e. IMEC Sensing agent and TCF LTE multi-antenna sensing device, further insights into their behaviour, integration and leveraging in existing testbeds are provided. In the case of LTE multi-antenna detection, the document delves with the algorithmic details of the solution proposed. The content presented in D3.1 is new, but it relies heavily on information already published within other deliverables. For that reason this document should be read with previous deliverables at hand, particularly D2.1 and D2.2 that explain the testbed architectures.

An entire chapter is devoted to the topic of common data formats. As announced in the technical annex, finding common data representation formats in a project addressing experimentation is of paramount importance. The chapter exposes the rationale behind the chosen data format, describes the template and the data types selected and provides a set of examples where the format is applied.

The common portal, a pivotal element of the whole project is very quickly addressed since more information is available online.

Special attention has to be paid to the content dealing with the testbed components and combinations. These combinations, another core element of work package 3, together with the common data formats, enable the creation of the virtual components. To make the interaction between components possible, interfaces are described with great detail.

Two interfaces are introduced: The Transceiver Facility API interface, and the IMEC sensing platform interface. The details exposed from these interfaces are supposed to offer external experimenters, or anyone considering a combination of separated individual elements, all the necessary knowledge to properly use the available features, and exploit the capabilities. For the Transceiver Facility API, Use cases are provided that should enable the user to better know how to use the interface functions and variables.

Even if the notion of virtual component and more specifically the potential number of virtual components is bolstered by the interfaces, other combinations could be envisioned without using the interfaces. Thus, two examples of combinations are given that use the available Federation functionalities in a straightforward way. One is the simulation of the LTE detection algorithm using real data produced by the LTE testbed. The data is generated at the source (the LTE Base station cell) and replayed on a computer where the synchronization and detection algorithm runs. The second is insertion of the IMEC advanced sensing platform in the IBBT testbed to extend, complete and strengthen the native sensing capabilities of the testbed. These two examples give good insights into the added value the federation brings for experimentation.

List of Acronyms and Abbreviations

API	Application Programming Interface
ASIP	Application-Specific Instruction-set Processor
ADC	Analogue to Digital Converter
BLER	Block Error Rate
BAN	Body Area Network
BTS	Base Station Transceiver
CR	Cognitive Radio
CRAWDAD	Community Resource for Archiving Wireless Data At Dartmouth
CREW	Cognitive Radio Experimentation World
CP	Cyclic Prefix
DIFFS	Digital Front-end For Sensing
DL	Down Link
DSP	Digital Signal Processor
DVB-T	Digital Video Broadcast Terrestrial
eNB	Enhanced Node B
ECG	Electrocardiographie
EMG	Electromyographie
EVA	Extended Vehicular A
FARAMIR	Flexible and spectrum-Aware Radio Access through Measurements and modelling In cognitive Radio Systems
FCC	Federal Communications Commission
FDD	Frequency Division Duplex
FFT	Fast Fourier Transform
FIFO	First In First Out
FPGA	Field Programmable Gate Array
GIS	Geographic Information System
GPP	General Purpose Processor
GSR	Galvanic Skin Response
HAL	Hardware Abstraction Layer
HARQ	Hybrid Automatic Repeat reQuest
ISM	Industrial Scientific Medical
IEEE	Institute of Electrical and Electronics Engineers
I/O	Input/Output
I/Q	In-phase/Quadrature
IP	Intellectual Property

LTE	Long Term Evolution
MAC	Medium Access Control
MIMO	Multiple Input Multiple Output
MCU	Micro Controller Unit
OFDM	Orthogonal Frequency Division Multiplex
OFDMA	Orthogonal Frequency Division Multiple Access
PBCH	Physical Broadcast Channel
PCFICH	Physical Control Format Indicator Channel
PD	Probability of Detection
PDCCH	Physical Data Control Channel
PFA	Probability of False Alarm
PHICH	Physical Hybrid ARQ Indicator Channel
PHY	Physical Layer
PSS	Primary Synchronization Signal
QAM	Quadrature Amplitude Modulation
QPSK	4 Phase Shift Keying
RAT	Radio Access Technology or Technique
RF	Radio Frequency
SCALDIO	Scalable Radio
SDR	Software Defined Radio
SINR	Signal to Interference and Noise Ratio
SIMO	Single Input Multiple Output
SIMD	Single Instruction Multiple Data
SSH	Secure Shell
SSS	Secondary Synchronization Signal
SWF	Spatial Wiener Filter
TETRA	Terrestrial Trunked Radio
TDD	Time Division Duplex
TTI	Transmission Time Interval
TWIST	TKN Wireless Indoor Sensor Network Testbed
UE	User Equipment
UL	Up Link
USRP	Universal Software Radio Peripheral
E-UTRA	Evolved Universal Terrestrial Radio Access
VNC	Virtual Network Computing
WARP	Wireless Open Access Research Platform
WCDMA	Wireless Code Division Multiple Access

WF	Waveform
WIMAX	Wireless Microwave Access
WInnF	Wireless Innovation Forum
XML	Extensible Markup Language

Table of contents

1 Introduction

1.1 Scope

This document aims at providing a complete and detailed description of the first basic operational federated CREW testbed. It will detail, one by one, the implemented and supported functionalities of the first release of the CREW federation. D3.1 belongs to WP3 “Creating the Federation”. It is consequently addressing the details of the creation or set-up of the Federation. The necessary effort to bring together the separated testbeds into an integrated Federation is therefore covered here.

Chapter 1: Introduces D3.1

Chapter 2: Lists the implemented (supported) functionalities of the federated testbeds

Chapter 3: Describes the common data formats proposed in accordance with the experiments

Chapter 4: Short overview of the CREW common portal

Chapter 5: Interfaces descriptions and potential combinations of testbeds elements for advanced experiments.

1.2 Document purpose and intended audience

This document is intended to provide a main reference to anyone interested in the usage of the CREW Federation. It should provide enough information to clearly grasp the capabilities of the Federation in terms of available functionalities so a potential external user may be able to make an assessment on the feasibility (or not) of the experiment he or she could have in mind.

1.3 References and links to other workpackages and deliverables

Please note that this document is not self-contained and more details on the CREW Federation testbeds may be found in other published documents. These details are quickly presented here to avoid redundancy. The present document will refer to those when appropriate. Two fundamental documents complement D3.1, these are:

D2.1 Definition of Internal Usage Scenarios: which focus on the initial CREW usage scenarios for the Federation.

D2.2 Definition of the Federation: which gives a wider overview or high-level description of the CREW Federation and its basic functionalities.

2 CREW federation implemented functionalities

The figure below provides an overview of the CREW federation and the integrated testbeds, before the start of the project.

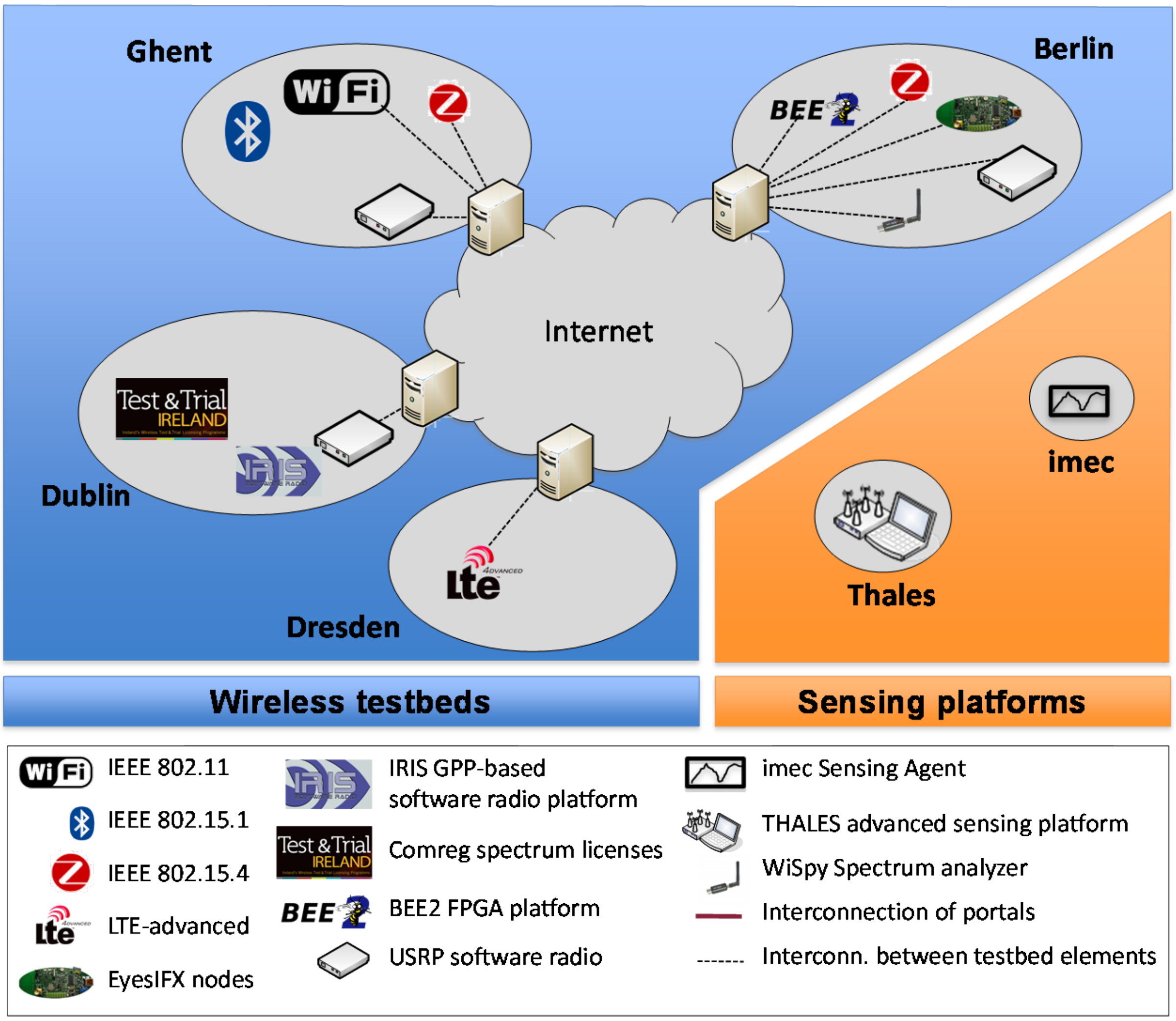


Figure 1: The CREW federated testbed

During the first year of the project, different components of the federation were integrated, and new functionality was designed. The resulting CREW federation as it exists today is shown in Figure 2. Besides the functionality that is readily available now, more components can be combined today on request. For example, while the imec sensing agent is only drawn in the Ghent testbed, such sensing agents could also be moved to other locations.

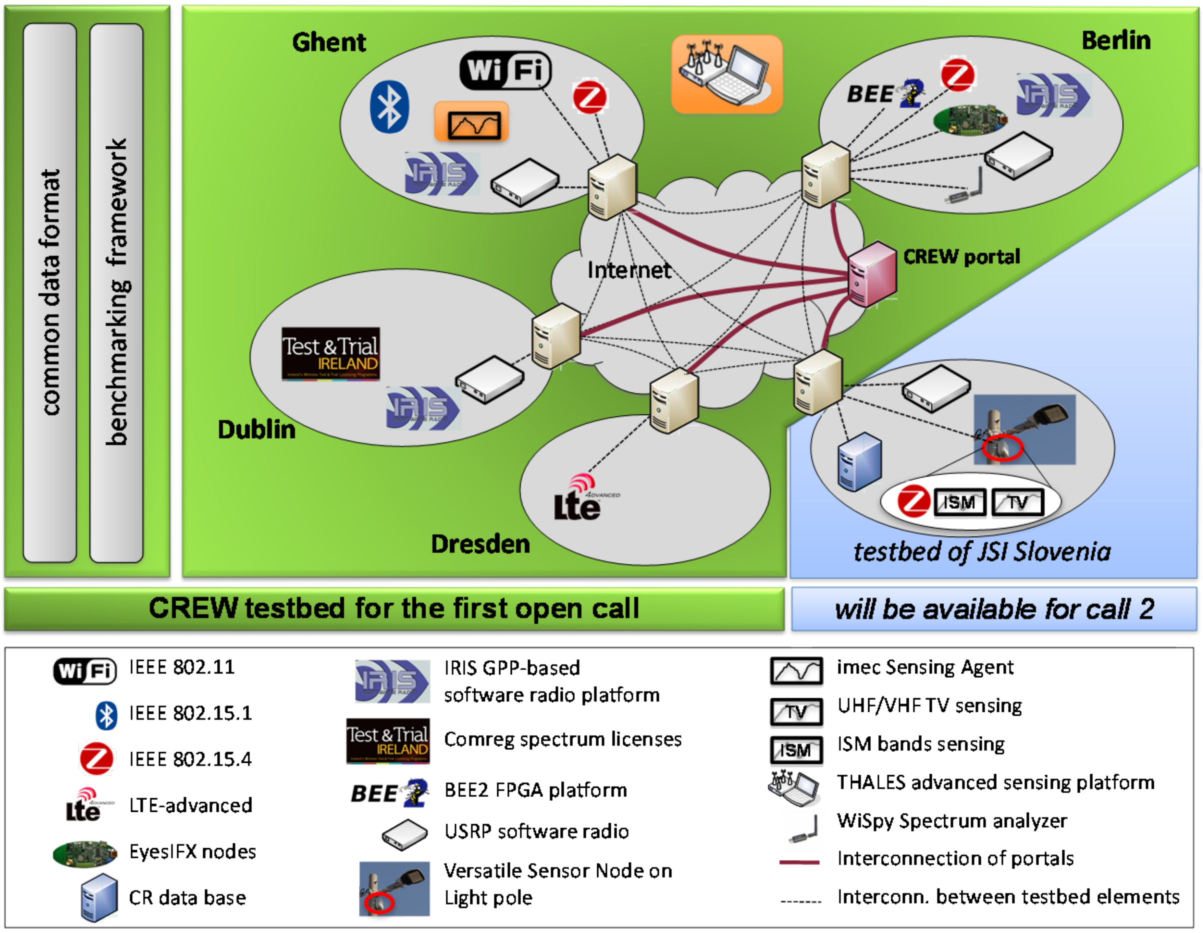


Figure 2 - CREW federation after Y1 of the project

2.1 TCD Iris testbed supported functionalities

The TCD Iris software defined radio testbed, has been already described in document D2.2 and consists of a highly reconfigurable software radio architecture that can be used to construct complex radio structures from a combination of C++ and XML. It is a GPP-based radio architecture and uses XML documents to describe the radio structure. This testbed provides a highly flexible architecture for real-time radio reconfigurability based on intelligent observations the radio makes about its surroundings.

As well as this software architecture, there exists a physical testbed that is integrated with the software that enables practical experimentation. The hardware components of the testbed at TCD consist of four Quad core machines, each of which has attached either a USRP 1, USRP 2 or USRP N210. The USRP (Universal Software Radio Peripheral) is a family of hardware used as an RF frontend for software radios. The USRP 1's have an 8MHz bandwidth, and the USRP 2's and USRP N210's have a 24MHz bandwidth (using Gigabit Ethernet to communicate between the USRP and the computer). The daughterboards available are the RFX2400 which are capable of transmitting and receiving between 2.3 and 2.9 GHz and the XCVR2450 dual band transceiver which can operate from 2.4 to 2.5 GHz and from 4.9 to 5.85 GHz. The structure of the testbed can be seen in Figure 3 below.

The entirety of the iris software can be downloaded and installed by following the instructions on the iris wiki https://ntrg020.cs.tcd.ie/irisv2/.

Use of the Iris testbed must be scheduled beforehand, this is done using the testbed google calendar. The calendar is called ctvr.testbed.

After obtaining an Iris testbed user account, the testbed can be fully, remotely accessed by external users:

2.1.1 Powering the USRPs

We have installed a remote power switch which allows us to remotely power each of the USRPs on and off. This switch can be controlled through a web interface.

Access the switch by navigating to http://ctvr-switch.cs.tcd.ie in your web browser.

2.1.2 VNC access

A useful approach for accessing the testbed remotely is through VNC. SSH into a node and start a VNC server as follows:

```
ssh nodeuser@ctvr-node07.cs.tcd.ie

vncserver :1 -geometry 1280x900
```

This will create a vncserver on display 1 of node 07 and with a 1280x900 screen resolution.

Once the server is running, use a VNC client to connect. In this case, we would connect to ctvr-node07.cs.tcd.ie:1.

When you are finished, kill the VNC server on the testbed node as follows:

```
vncserver -kill :1
```

Once the user has accessed the testbed remotely as shown above, radios and their respective components can be opened, edited and run over the air between different nodes in the testbeds.

Use of the Anritsu MG3700A signal generator and the Rhode and Schwarz FSVR real-time spectrum analyser is done in a similar way. Full details of how this is done, as well as further details on other aspects of testbed use, are available on the Iris testbed section of the CREW portal.

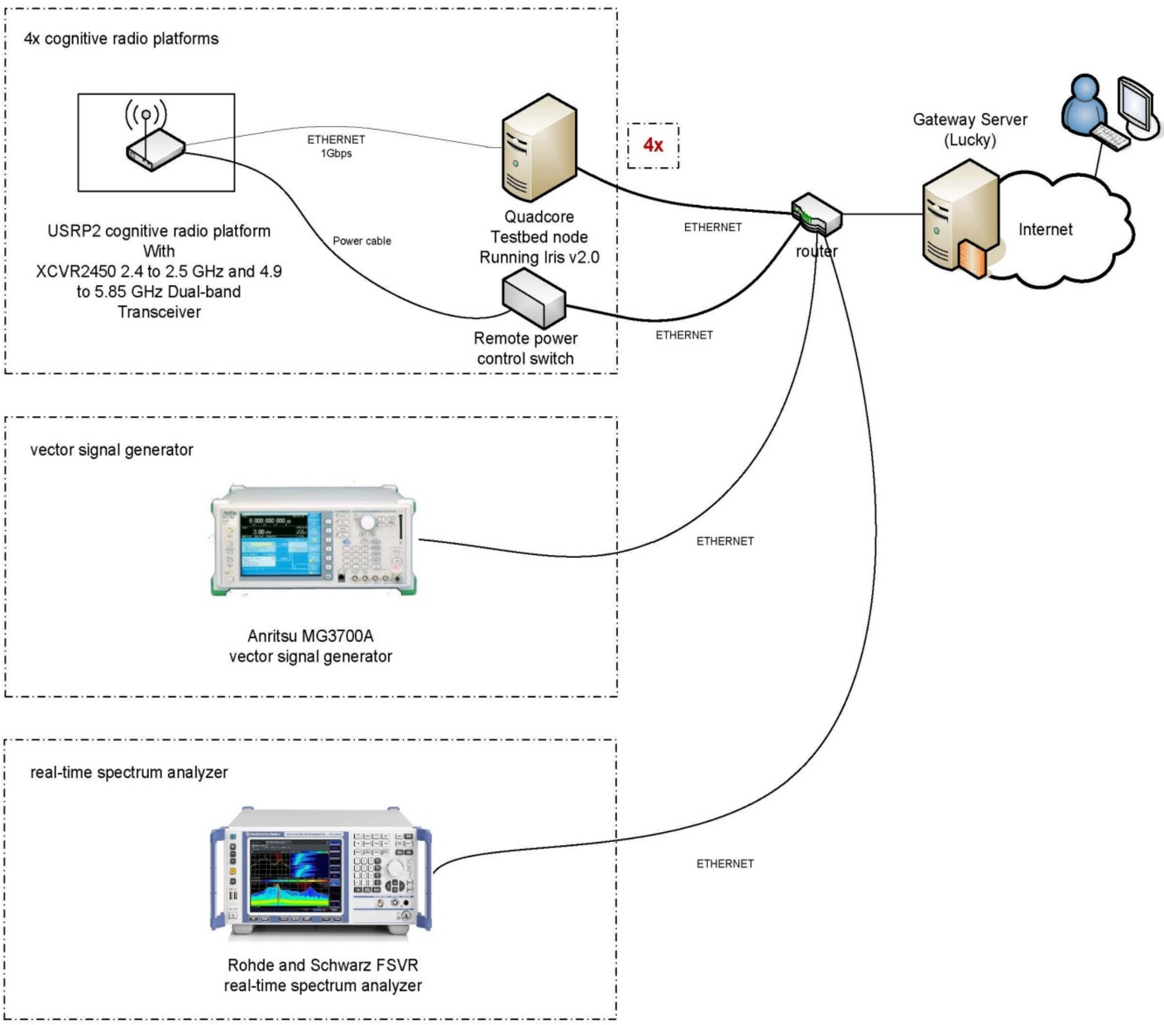


Figure 3: Overview of hardware available to experimenters in the TCD Iris lab

2.2 TUB TWIST testbed supported functionalities

The TUB TWIST testbed was already introduced in CREW deliverable D2.2, Section 1.5. As stated in that document, the TKN Wireless Indoor Sensor Network Testbed (TWIST) is a multi-platform, hierarchical sensor network testbed architecture developed at the Technische Universität Berlin (TUB). One instance is currently deployed at TUB campus: a total of 204 sensor nodes (102 eyesIFX and 102 Tmote Sky nodes) are distributed in a 3D grid spanning 3 floors of an office building, resulting in more than 1500 m² of instrumented office space.

In the following we give an overview of the hardware, tools, and functionalities that are available to the experimenters at the TUB testbed. An experimenter will have access to the following components:

(1) The TWIST sensornet testbed with 204 sensor nodes,

(2) One mobile robot that t can be programmed to follow certain trajectories in the TWIST building. Shimmer2 sensor nodes or WiSpy devices (see below) can be mounted on the robot, e.g. to record RF environmental maps, or perform experiments emulating body area networks (BANs) as well as experiments involving interaction between a mobile network and the fixed TWIST infrastructure,

(3) A set of low-cost USB spectrum analyzers with a custom software framework on a laptop, which may be deployed at various locations in the testbed or on the mobile robot to monitor spectrum usage during an experiment,

(4) A set of at least 8 shimmer2r sensor nodes to be used for mobile BAN experiments; a custom setup for synchronization of the nodes via digial I/O cabling is provided as well as a laptop with a pre-installed toolchain to program/access the shimmer2r nodes.

These components are depicted in Figure 4 and described in more detail below.

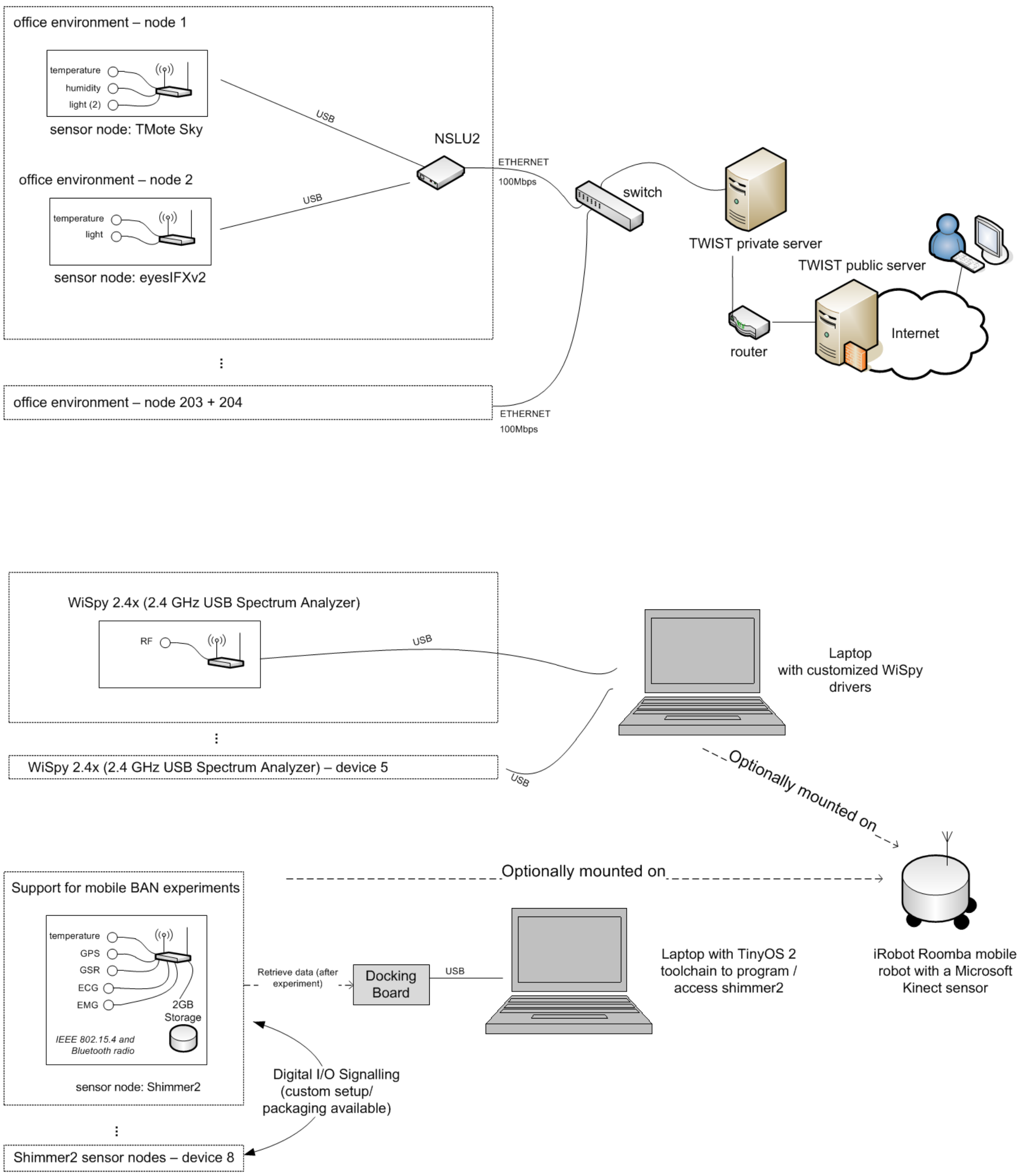


Figure 4: Overview of hardware available to experimenters in the TUB TWIST testbed

2.2.1 TWIST sensornet testbed

The first component, the TUB TWIST sensornet testbed, was introduced in CREW deliverable D2.2, Section 1.5 on a rather high level. Figure 5 gives a more concise view of the functionality that is offered to the user, and highlights how it is accessed as well as which parts can be modified / parameterized by the user during an experiment.

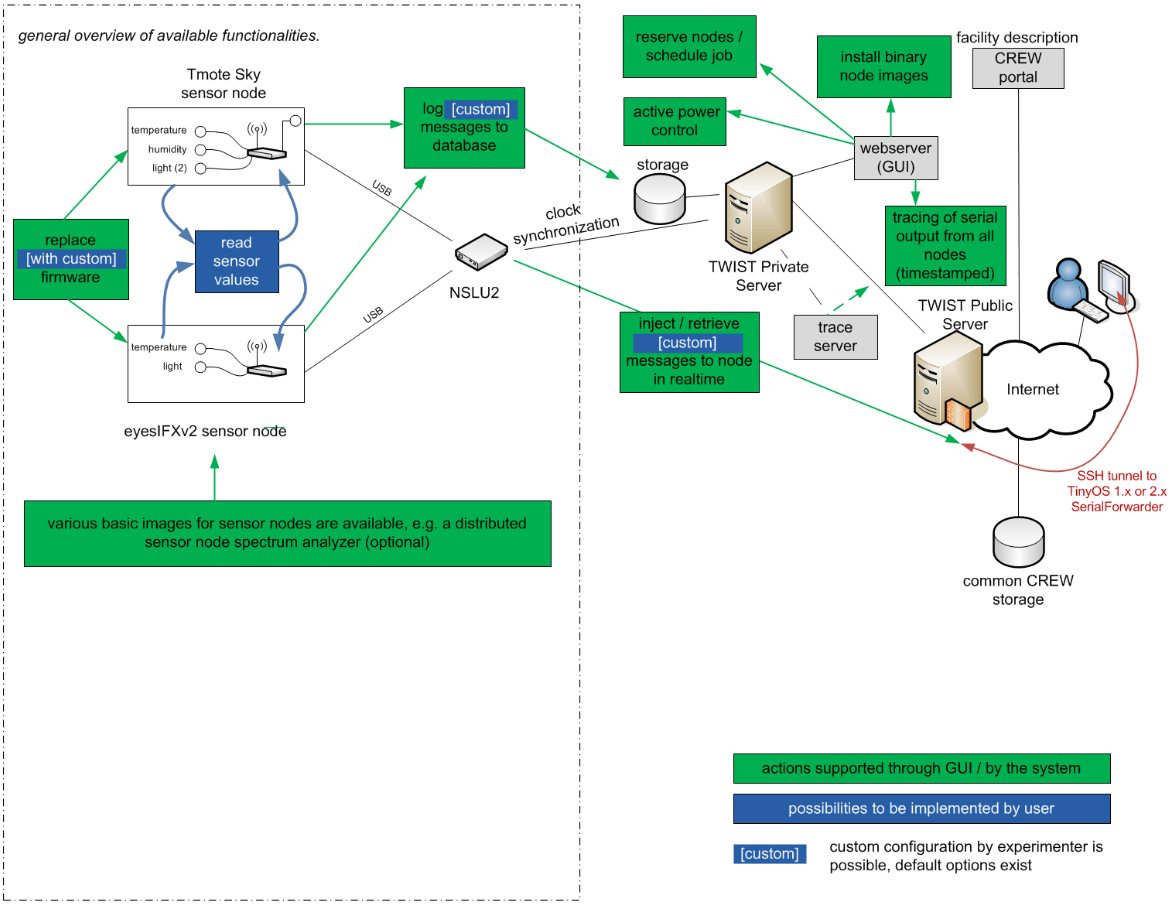


Figure 5: General overview of available TWIST functionality

A user can access the TWIST testbed via a web interface remotely. The first step of an experiment always involves scheduling of an experiment by reserving an adequate experiment "job" (a time slot and a set of resources, such as the type of sensornodes). Once the user's job becomes active, the user can upload node images (firmware image) to a set of nodes. The user may decide to upload different images to different nodes in parallel.

TWIST supports automatic tracing, i.e. whenever nodes output data (such as experiment results, traces, debug messages, etc.) over the serial line the data is automatically stored in the trace file, which the user may download after the experiment has been completed. In addition a user may want to interact with a set of sensor nodes over the USB control channel during the experiment in realtime (e.g. to change some node parameters). This is supported by providing access to each node's TinyOS 1.x or 2.x SerialForwarder, which establishes a connection over a TCP/IP stream in order to get serial access to the node.

A more detailed explanation on which TWIST functionality an experimenter can use and how they are accessed (job registration, installation of a node image, using the tracing server, etc.) is described in several detailed tutorials on the portal.

2.2.2 Mobile robot

Experimenters have access to a mobile iRobot Roomba robot which is coupled with a Microsoft Kinect sensor. The robot runs ROS (an open-source, meta-operating system) and it can be programmed to follow certain trajectories in the TWIST building. It can be controlled by the experimenter via custom scripts to perform repeatable mobility patterns. In contrast to the TWIST sensornet testbed the robot cannot be accessed remotely (via a webinterface), i.e. experimenters have to be present to start their experiments locally. Shimmer2 sensor nodes or WiSpy devices may be mounted on the robot, e.g. to record RF environmental maps, or perform experiments emulating body area networks (BANs) as well as experiments involving interaction between a mobile network and the fixed TWIST infrastructure.

2.2.3 USB spectrum analyser framework

The testbed infrastructure includes several commercial low-cost USB spectrum analysers: Wi-Spy 2.4x (Metageek). The Wi-Spy 2.4x is a spectrum analyser that scans for RF activity (RF power) in the 2.4 GHz spectrum. The Wi-Spy 2.4x allow users to quickly identify interference and analyse the quality of the signal. Compared to other spectrum sensing solutions these devices may not be very accurate, however, in certain cognitive radio scenarios they may be sufficient to be used as sensing agents. Wi-Spys may, for example, be deployed at various locations in the testbed or mounted on the mobile robot to monitor spectrum usage during a sensornet experiment.

The infrastructure includes the USB spectrum analysers in conjunction with a customized software framework, which provides an experimenter with fine-grained control over the parameter setting (e.g. select only a subset of the entire 2.4 GHz ISM band). The laptop with the pre-installed software framework is also part of the infrastructure and can be utilized during the experiments.

2.2.4 BAN sensor nodes

The testbed infrastructure includes a set of shimmer2 sensor nodes which may be used in BAN (Body Area Network) scenarios. Like the popular Telos sensor node platform, Shimmer2 integrates the Texas Instruments MSP430 MCU and the IEEE 802.15.4-compliant CC2420 transceiver. In addition, the Shimmer2 platform also incorporates a Bluetooth radio. Every Shimmer2 node has an integrated 3-axis accelerometer (Freescale MMA7260Q), which we can be utilized to monitor the subject's movement pattern. Furthermore, we provide a set of medical sensors (ECG, GSR, EMG) that may be used to develop realistic medical application scenarios. Our Shimmer nodes are also equipped with a 2 GB MiniSD card, which is sufficient to store all traces that accumulate during an experiment.

The shimmer2 sensor nodes are provided together with a laptop with a pre-installed toolchain for installing TinyOS 2 node images via the shimmer programmer board. The laptop also allows users to conveniently access the measurement traces on the MiniSD card after an experiment. Together with the remaining TWIST infrastructure, an experimenter may, for example, investigate cognitive radio techniques that couple mobile (BAN) with a static (TWIST) networks or low-cost sensing agents (WiSpy). The BAN sensor nodes can also be mounted on the mobile robot, for example, to emulate a mobile BAN.

Finally, the BAN sensor nodes are provided together with an optional custom setup that allows users to connect the BAN nodes via a digital I/O control channel (via dedicated cabling). This additional channel may be used, for example, to achieve tight time-synchronization between the nodes (at the order of microseconds).

2.3 TUD LTE+ Testbed supported functionalities

A list of the components that are available in the LTE+ testbed can be found in D2.2, section 2.3. Experiments can be conducted either in the indoor lab or with the two outdoor sites on the roof of the university building.

Base station (eNB) and mobile terminal (UE) nodes each are connected to a host PC and configured with text files in XML format. The host computer also manages measurements of the received signals and stores them in dumps. At the eNBs, a GPS unit is used for synchronization, while the UEs employ GPS for position tracking. Additionally, UEs can be powered by a mobile power supply if necessary.

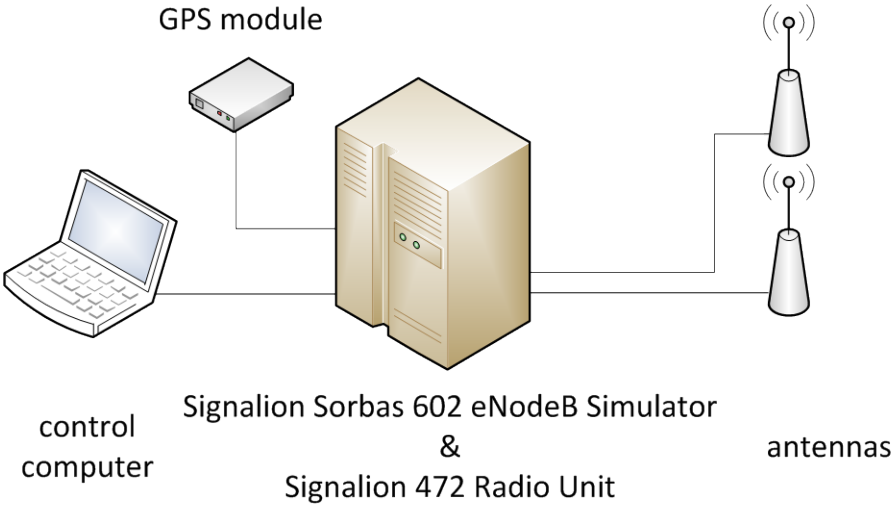


Figure 6: Configuration of a baste station node (eNB)

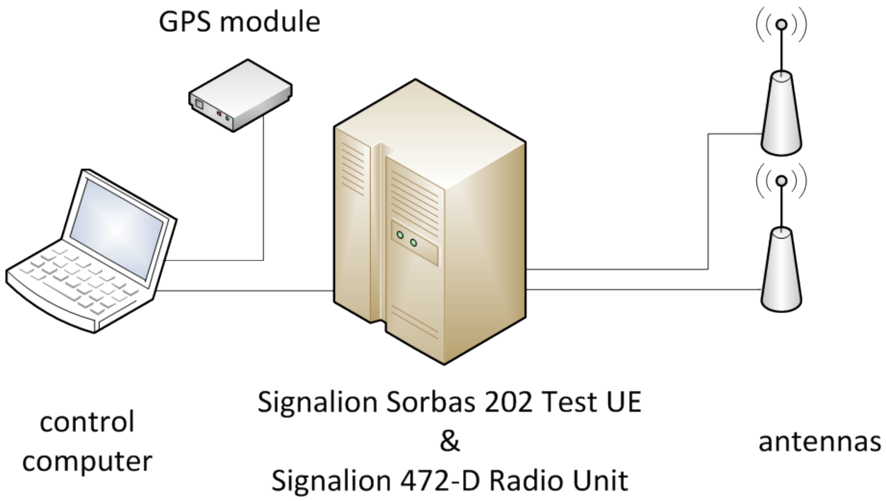


Figure 7: Configuration of a mobile terminal node (UE)

It is important to distinguish if a downlink (DL) or an uplink (UL) experiment is desired. Further, when considering the supported functionalities, it is necessary to be aware that the testbed provides only basic compliance with LTE Rel. 8 and that there are several deviations: Particularly in DL, the frame structure and control channels slightly differ from what is stated in the specifications:

- PDCCH is always on the 2nd OFDM-symbol
- PHICH is not in the first OFDM symbol and has a different structure and content
- PCFICH is not supported
- PBCH is not supported

The uplink operates with OFDM. Also, 5 MHz and 10 MHz mode are not supported, thus the testbed operates in 20 MHz mode only. At the moment of writing this document, the testbed hardware occupies E-UTRA Band 7 at 2.6 GHz. However due to regulatory constraints we expect to switch to frequencies around 2.1 GHz within the next 9 months.

2.3.1 Uplink functionality

In uplink experiments, it is possible to serve up to 4 UEs. The UEs use 1 antenna for transmission, while the eNBs can receive with 1 or 2 antennas. The resolution for scheduling a transmission is 1 ms, which corresponds to 1 TTI (transmission time interval). Scheduling can be done for a total duration of several minutes. The number of occupied PRBs is either 10, 20, 30 or 40 (cf. Table 1). QPSK, 16QAM and 64QAM modulation are supported.

2.3.2 Downlink functionality

In downlink, up to 4 UEs and up to 4 eNBs can be used simultaneously. The eNBs can transmit with up to 2 antennas and the UEs can receive with up to 2 antennas, thus up to 2 streams per UE can be sent. Time resolution is 1 ms corresponding to 1 TTI (same as UL). The number of occupied PRBs can be 12, 24, 36 or 48 (cf. Table 2).

2.3.3 Signal measurement functionality

The evaluation of an experiment happens via dumps of the received signals at the UEs / eNBs. While in the UL, signal dumps can be recorded for all eNBs in synch, the dumping process needs to be initiated manually and out of synch in the DL.

The signal dumps contain the received time samples as well as additional control information. Further processing in Matlab allows derivation of indicators like SINR, BLE, etc. in semi-realtime/offline.

Table 1: Supported transport formats for uplink

Idx	Size (bits)	#PRBs	Mod	Idx	Size (bits)	#PRBs	Mod	Idx	Size (bits)	#PRBs	Mod
0	80	10	QPSK	23	7840	20	16QAM	48	13760	40	16QAM
1	240	10	QPSK	26	240	30	QPSK	49	15680	40	16QAM
2	480	10	QPSK	27	720	30	QPSK	55	4000	10	64QAM
3	800	10	QPSK	28	1440	30	QPSK	56	5360	10	64QAM
4	1280	10	QPSK	29	2400	30	QPSK	57	6080	10	64QAM
5	1680	10	QPSK	30	3840	30	QPSK	58	6640	10	64QAM
6	2080	10	16QAM	31	5040	30	QPSK	59	6960	10	64QAM
7	2560	10	16QAM	32	6240	30	16QAM	60	7920	10	64QAM
8	2960	10	16QAM	33	7680	30	16QAM	61	8000	20	64QAM
9	3440	10	16QAM	34	8880	30	16QAM	62	10720	20	64QAM
10	3920	10	16QAM	35	10320	30	16QAM	63	12000	20	64QAM
13	160	20	QPSK	36	11760	30	16QAM	64	13280	20	64QAM
14	480	20	QPSK	40	960	40	QPSK	65	14080	20	64QAM
15	960	20	QPSK	41	1920	40	QPSK	66	15840	20	64QAM
16	1600	20	QPSK	42	3200	40	QPSK	67	12000	30	64QAM
17	2560	20	QPSK	43	5120	40	QPSK	68	16080	30	64QAM
18	3360	20	QPSK	44	6720	40	QPSK	69	18000	30	64QAM
19	4160	20	16QAM	45	8320	40	16QAM	70	19920	30	64QAM
20	5120	20	16QAM	46	10240	40	16QAM	71	20880	30	64QAM
21	5920	20	16QAM	47	11840	40	16QAM	72	23760	30	64QAM
22	6880	20	16QAM								

Table 2: Supported transport formats for downlink

Idx	Size (bits)	#PRBs	Mod	Idx	Size (bits)	#PRBs	Mod	Idx	Size (bits)	#PRBs	Mod
1	288	12	QPSK	24	4992	24	16QAM	47	16416	36	64QAM
2	576	12	QPSK	25	6144	24	16QAM	48	18720	36	64QAM
3	960	12	QPSK	26	7104	24	16QAM	49	19872	36	64QAM
4	1536	12	QPSK	27	8256	24	16QAM	50	21312	36	64QAM
5	2016	12	QPSK	28	9408	24	16QAM	51	23904	36	64QAM
6	2496	12	16QAM	29	10944	24	64QAM	54	384	48	QPSK
7	3072	12	16QAM	30	12480	24	64QAM	55	1152	48	QPSK
8	3552	12	16QAM	31	13248	24	64QAM	56	2304	48	QPSK
9	4128	12	16QAM	32	14208	24	64QAM	57	3840	48	QPSK
10	4704	12	16QAM	33	15936	24	64QAM	58	6144	48	QPSK
11	5472	12	64QAM	36	288	36	QPSK	59	8064	48	QPSK
12	6240	12	64QAM	37	864	36	QPSK	60	9984	48	16QAM
13	6624	12	64QAM	38	1728	36	QPSK	61	12288	48	16QAM
14	7104	12	64QAM	39	2880	36	QPSK	62	14208	48	16QAM
15	7968	12	64QAM	40	4608	36	QPSK	63	16512	48	16QAM
18	192	24	QPSK	41	6048	36	QPSK	64	18816	48	16QAM
19	576	24	QPSK	42	7488	36	16QAM	65	21888	48	64QAM
20	1152	24	QPSK	43	9216	36	16QAM	66	24960	48	64QAM
21	1920	24	QPSK	44	10656	36	16QAM	67	26496	48	64QAM
22	3072	24	QPSK	45	12384	36	16QAM	68	28416	48	64QAM
23	4032	24	QPSK	46	14112	36	16QAM	69	31872	48	64QAM

2.4 IMEC sensing platform supported functionalities

The use of the imec sensing engine was already introduced in CREW deliverable D2.2. In that deliverable, various ways to make use of the sensing engine were introduced. In this deliverable, we use the same approaches and elaborate a bit more on how to implement these various possible experiments.

2.4.1 Integrating the sensing solution in the experimenter's own testbed through the USB interface and use of the driver

The imec sensing engine is equipped with a USB interface. All configuration, control and measurement data is communicated via this interface. The host which controls the sensing engine needs to have the libusb library, which provides user applications a uniform access to USB devices on different operating systems, installed. A Hardware Abstraction Layer (HAL) and Application Programmers Interface (API) have been developed for usage of the sensing engine with a Linux host PC running Ubuntu. Both HAL and API are developed in ANSI C code which will enable portability to other operating systems.

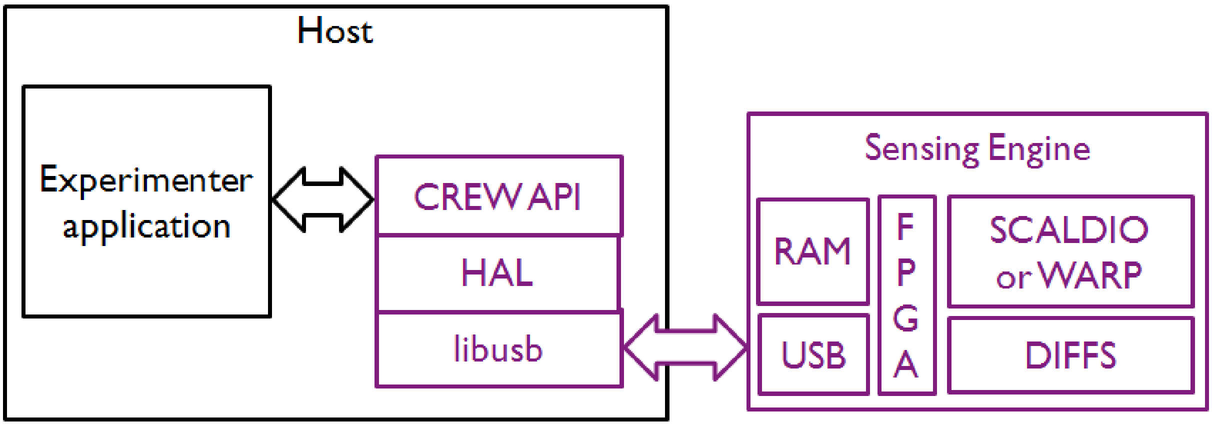


Figure 8: Sensing Engine Host integration

Figure 8 shows an overview of the system: the Host PC runs the Experimenter application which needs to access the Sensing Engine through the API.

The interfaces in the HAL are described in more detail in Section 5.2.2 of this deliverable.

2.4.2 Leveraging on the integration of the sensing engine in CREW testbeds

During the federation activities of the CREW project, the sensing engine will be federated with the CREW testbeds. This will allow new modes to use the sensing engine. For instance, when multiple sensing engines are integrated in the w-iLab.t testbed of IBBT, it will be possible to use the IBBT tools for creating a scenario, a test and benchmarking. This will enable testing a larger set of usage scenarios, such as, for instance, distributed sensing. The sensing engine can then be accessed through the interfaces available in the hosting testbed. This is discussed more in Section 2.5.4 and Section 5.3.2 for the case of the integration in the Wilab.t testbed.

2.4.3 Reprogramming the sensing engine with specific functionality

The sensing engine is very flexible and programmable. While basic functionality is there, after the first year of the CREW project, one can envision that researchers will be interested in testing newly developed functionality on the platform. The functionality that is there at the end of the first year of the project is (discussed in more details in D6.1):

- Sweeping of the 2.4 GHz ISM band for integration in the w-iLab.t testbed and for doing sensing experiments in the 2.4 GHz ISM band.
- Sensing of the OFDMA resource allocation of LTE. (currently not supported through API)
- Feature-detection of DVB-T.

To write new functionality for the sensing engine, one should write a program for the 32-slot SIMD processor (with optimized instruction set that was designed using the Coware toolflow. Special instructions/hardware accelerators exist for:

- Auto/Cross correlation and signal power
- Parallel FFT (128 complex values).

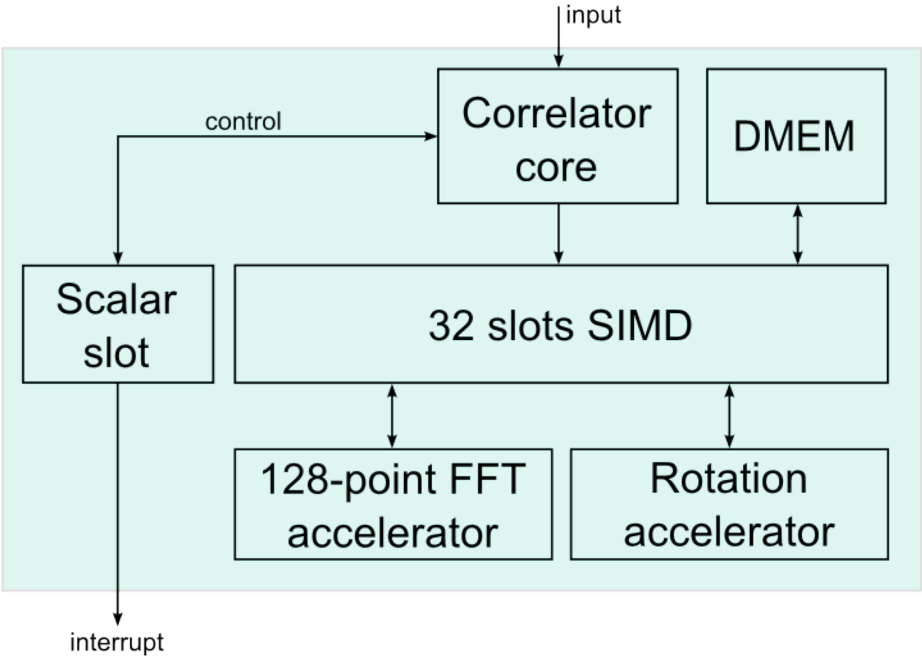


Figure 9: SIMD processor in the Sensing Engine Processor with accelerators.

In addition to the SIMD processor, one can configure the automatic gain control, the IQ imbalance and DC Offset compensation blocks, and the flexible filter branch. A high level overview of the processor is given in Figure 10.

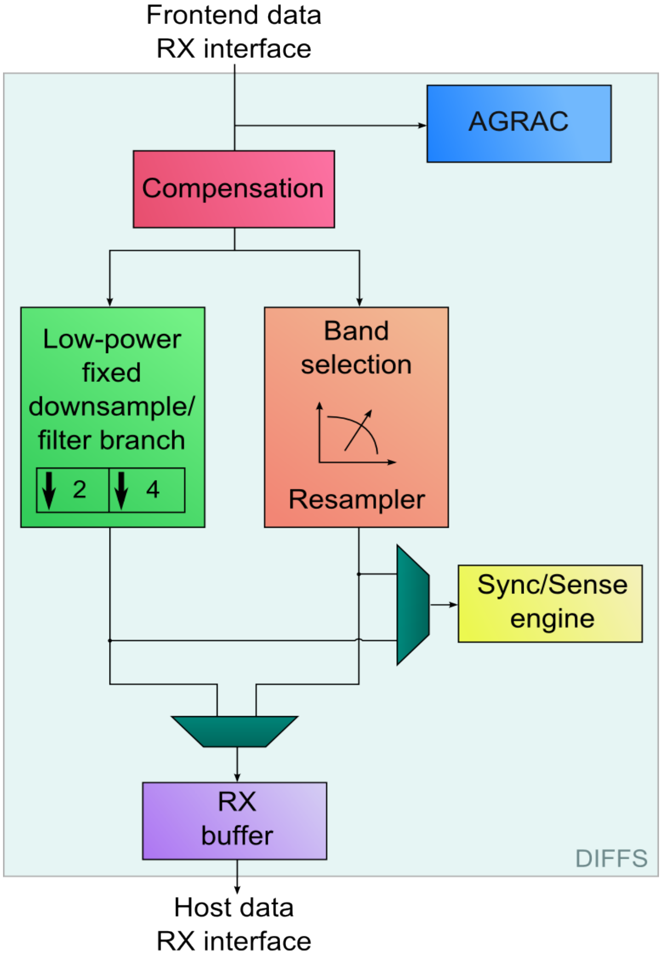


Figure 10: Sensing Engine processor high level view.

More information on how to interface with the chip is given in Section 5.2.2 and Table 4.

2.4.4 Making use of samples from the sensing engine to test algorithms

If it is not possible, or too much effort, to run the sensing functionality on the hardware, one could imagine testing functionality using I and Q samples obtained from the sensing engine. Various samples have been collected during the CREW experiments that were held in Dublin, Dresden and Berlin. These data sets are available through the CREW portal and the common data format is described in Section 3.

2.4.5 Mixing and matching the hardware of the sensing engine with the experimenter's own hardware components

The sensing engine, as used in the CREW federated testbed, can contain the SCALDIO or the WARP front-end. It could be possible to experiment with other front-ends, so as to compare the performance of different front-end solutions.

For instance, the current board is designed to operate with the imec Scaldio-2b board, the imec Scaldio-2c board and the Rice university WARP board. Each can be used for different sensing scenarios. With the Scaldio-2b board, it is possible to scan from 1MHz to 6 GHz with a single sensing engine. With the WARP board it is possible to sense the ISM bands, using up to 10 different sensing engines. With the WARP board, it is also possible to replay interference in the ISM bands. The different scenarios can be implemented by connecting different RF front-ends to the sensing engine processor board. It is possible to connect also different front-ends, provided the connector matches.

2.5 IBBT w-iLab.t testbed supported functionalities

The IBBT w-iLab.t infrastructures were already introduced in CREW deliverable D2.2, Section 2.5. As stated in this document, the w-iLab.t is a wireless Wi-Fi and sensor network testbed infrastructure, currently deployed across three 90 *m* x 18 *m* floors of the IBBT office building in Ghent, Belgium. At 200 places throughout the office spaces, meeting rooms and corridors, wireless hardware is mounted to the ceiling.

At the moment of writing this deliverable, an extension to the w-iLab.t infrastructure is being set up in a building in Zwijnaarde, Belgium, located approximately 5 *km* away from the current instance of w-iLab.t. This extension to the infrastructure will be available to the experimenters joining the CREW project after the first open call for experimenters (experiments are expected to start in January 2012).

The following paragraphs give an overview of the hardware, tools, and functionalities that are available to the experimenters at the two testbed locations (IBBT office and Zwijnaarde). When executing experiments in the office building, experimenters should take into account that interference from other 2.4 *GHz* and 5 *GHz* ISM test set-ups and production networks is likely. Measurements performed at the Zwijnaarde location show that minimal 2.4 *GHz* and 5 *GHz* ISM interference is suffered at this location, which stems from the fact that this testbed is located on top of a cleanroom facility, which is encapsulated in metal and concrete. Moreover, at this location, there are no offices with production (or experimental) Wi-Fi networks in the immediate environment of the testbed nodes. For this reason, the Zwijnaarde location will also be labelled "pseudo-shielded" in what follows.

Figure 11 shows a schematic overview of the hardware that is available at the two testbed locations (office environment and pseudo-shielded environment). The equipment on the right of the figure that is *not* displayed in a box, includes the computer of the experimenter (located anywhere on the internet), and a set of routers, switches and links that enable the user to take control over the actual testbed devices (drawn in the boxes). Note that the drawing represents the switches, links and routers in a simplified way, for the sake of clarity. From top to bottom, the boxes show:

1. the equipment available at the IBBT office (first two boxes, 200 nodes),
2. a similar yet enhanced set of nodes located in Zwijnaarde (boxes 3,4; 40 nodes),
3. 10 imec sensing nodes (cf. Section 2.4 of this document),

4. 8 USRP cognitive radio platforms and the hardware needed to drive these components.

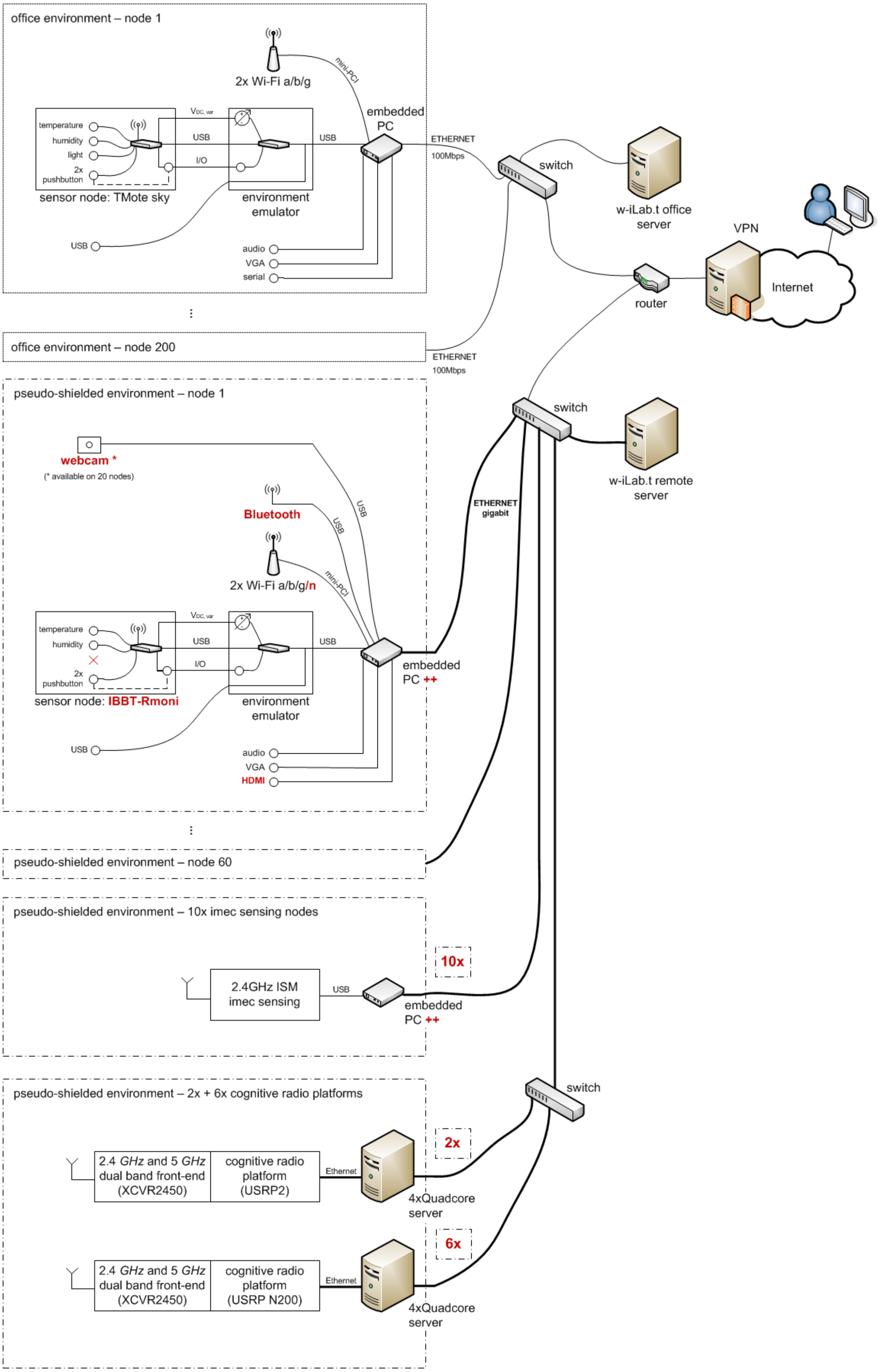


Figure 11: Overview of hardware available to experimenters in the IBBT w-iLab.t

All of the above hardware may be used by experimenters to perform cognitive networking experiments. However, the w-iLab.t offers more than just the hardware: a wide selection of software tools is offered to help experimenters to define, run, and monitor their experiments. Furthermore, functionality is in place to assist researchers in logging, visualizing and processing their results in real-time or after the experiment.

Figure 12 shows a schematic overview of the functionality that is offered to the user who is interested in using the nodes that are labelled (1) and (2) in the classification above. It is important to remember that w-iLab.t offers the hardware (nodes) and tools: the behaviour of the nodes may be entirely programmed by the experimenter. For example: each of the embedded PCs is equipped with 2 Wi-Fi interfaces. On top of these embedded devices, an experimenter may install any set of drivers, protocols and applications, meaning that, for example, an embedded PC may be configured purely as a server (e.g. webserver, data collection server), or as a Wi-Fi access point (e.g. with cognitive protocols for channel selection), or as a Wi-Fi client connecting to an access point (possibly using multiple Wi-Fi interfaces), or as a gateway (e.g. to the sensor network, or to the backbone), or any other functionality such a device might have. As such, the functionality of (task carried out by) the testbed is endless and up to the imagination of the experimenter.

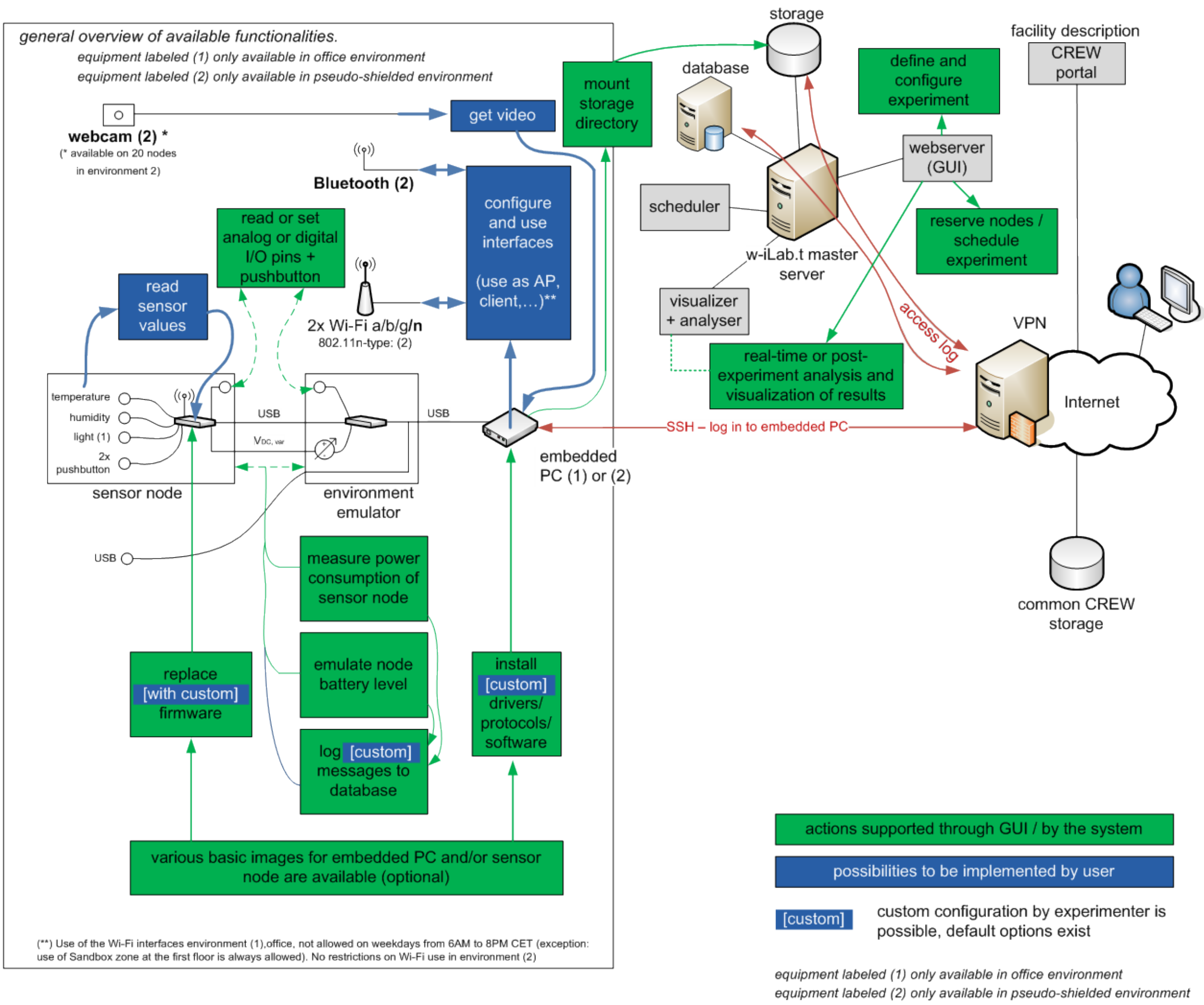


Figure 12: schematic overview of the functionality related to the nodes available to the experimenters

The remainder of this section offers a concise description of the most important functionalities, some of which are highlighted in Figure 12. The additional functionalities not listed in the figure, are equally relevant, with some of these specifically targeting the hardware listed above under (3) and (4). Although this information will help experimenters to pinpoint *what* is possible when using the IBBT infrastructure, in-depth information on *how* to use the functionality is not included in this document. For information on how to access and/or modify the functionalities, and for further details on the hardware and software components (including datasheets, tutorials, and code examples), the reader of this document is invited to the CREW portal at www.crew-project.eu. As the w-iLab.t is continuously being updated and expanded, the portal will also always offer the most recent version of the available equipment and services.

2.5.1 Install (custom) firmware, software, drivers, protocols on embedded PCs and sensor nodes

The functionality included in Figure 12 includes the following. Experimenters developing cognitive radio protocol stacks can make full use of the embedded PCs, sensor nodes, or a combination of these devices. The experimenters may fully configure the Linux-based embedded platforms. This includes installing their own applications, networking protocols, MAC layer protocols, kernel, and/or drivers. The limitations related to the configurability of the embedded PC's are basically the same as experimenters would meet when installing a computer system on their desktops. Sensor nodes may be flashed with custom firmware images. Tools are available to help experimenters to easily install their solutions on top of a single, a selection of, or all nodes available in the testbed. Alternatively, users may perform manual configurations, e.g. on the embedded PC's via SSH login.

Also worth mentioning is the possibility to log results to a storage server, visualise results with a "visualizer", and analyse results with an "analyser". Advanced functionality related to these functions includes real-time interaction with the I/O pins available on the sensor nodes via an "environment emulator". This same environment emulator enables users to perform real-time power consumption measurements of the sensor nodes, emulate battery levels, and add timestamps to logging information as to guarantee time-synchronisation of the logging output of the sensor nodes and the testbed system. Details are available on the CREW portal.

As can be seen in the figures, different technologies are available to experiment with, including Wi-Fi based on the IEEE 802.11a/b/g/n standards, IEEE 802.15.4, and Bluetooth.

2.5.2 Use of the cognitive radio platforms: USRP hardware with 2.4 GHz ISM front-end

USRP devices are available in the Zwijnaarde testbed. The devices may be used by the experimenters. Sample code will be provided. It is, for example, possible to install the IRIS software radio on top of this hardware.

2.5.3 Use of the CREW benchmarking framework: reproducible environments and performance comparison

The CREW benchmarking framework, as described in CREW deliverable D2.2 Section 3.5, is operational inside the IBBT testbed. The benchmarking framework assists experimenters in their experiments, by providing the means *(i)* to recreate repeatable wireless environments, and *(ii)* compare results.

The repeatable wireless environments are a set of wireless devices, transmitting a predefined traffic or interference pattern. The repeatability of these wireless settings is verified through repeated experiments; these repeated experiments and analysis thereof are supported by the framework. For the first open call, IBBT offers a repeatable home environment comprising a set of 802.15.4 based sensor nodes, Wi-Fi access points and Wi-Fi stations. Experimenters may use this setting, or create similar or completely new settings based on this provided scenario and can use the provided analysis tools to verify the repeatability of the created environment. Details on the offered reproducible environment are provided in CREW deliverable D4.1.

The benchmarking framework furthermore allows experimenters to reliably compare the performance difference of different iterations of their cognitive solution. Comparing a given solution with a completely different solution is also possible. A benchmarking API is offered to the experimenters. This benchmarking API may be used by experimenters to let the benchmarking framework interact with their own custom software / drivers / protocol stacks, once they are installed on the devices offered by the w-iLab.t. The API is available for experimenters in two flavours, one for sensor nodes as TinyOS modules, and one for the embedded PC's as linux scripts. Both APIs offer similar functionality, with specific extensions for the respective platforms. Details on the benchmarking functionality can be found on the CREW portal.

2.5.4 Use of imec sensing agents

The imec sensing engines are integrated in the IBBT w-iLab.t. Users may use the information collected by the sensing engines during their experiments, either as an external monitoring platform, or, as a way to provide input parameters to their protocols. Details are available in Section 2.4 of this document, and on the portal.

2.6 THALES Multi-antenna LTE detection procedure

2.6.1 Introduction

The goal of the detection procedure is to detect all the significant base stations surrounding the LTE sensing equipment. However, the rather straightforward algorithms implemented in mobile terminals do not allow detecting weak base stations. Multi-antenna receivers and smart antenna processing allow increasing detection performance.

For the two use cases described in D2.3 (US13 “Reliable sensing of cellular systems” and US51 “Impact of cognitive networking on a cellular primary system”) sharp detection must be performed. Indeed, in a CR system, the imposed sensing sensitivity is classically very high (for example, the FCC imposed TV bands devices to be capable of sensing analogue TV signals, digital TV signals and wireless microphone signals at a level of -114 dBm within defined receiver bandwidths [3]). Moreover, in a metrology approach, it can be shown that low base stations may impact in a significant way the mobile performance and thus need to be detected by a reliable tool. It is proved that these stations with a SIR down to -15 dB should be detected.

In section 2.6.2.5 the spatial detector is described as well as the two-step synchronization procedure considered the standard

- Detection of the Primary Synchronization Signal, acquisition of the slot synchronization and identification of the cell identity within the cell identity group,
- Detection of the Secondary Synchronization Signal, acquisition of the frame synchronization and identification of the physical layer identity of the cell, the CP length and the duplex mode.

2.6.2 Reference-based multi-antenna detection

2.6.2.1 Mathematical notations and signal modeling

In this section, we present the multi-antenna reference-based algorithm used to detect the PSS and the SSS broadcast by the significant base stations. It consists of evaluating at each time position, relevant statistics to be compared to a given threshold.

Let us first give some algorithmic notations.

The discrete time signal received at time *n*, on antenna *m* is written $x^{(m)}[n]$. Thus, the received snapshot vector on the antenna array can be written as:

$$\underline{\mathrm{x}}[n] = \begin{bmatrix} x^{(1)}[n] \\ \vdots \\ x^{(M)}[n] \end{bmatrix} \qquad \text{Eq. 2-1}$$

M being the total number of antennas of the array.

The presence or the absence of the synchronization signal at time *n* can be formulated as the following composite hypothesis-testing problem:

- Hypothesis H_1: Presence of the synchronization signal

The discrete time signal $\underline{x}$ can be written as:

$$\underline{x}[n+k] = \underline{h}d[k] + \sum_{p\neq 0} \underline{h}_p\, d[k-p] + \underline{n}[n+k], \text{for } k = 0,\dots,N-1 \qquad \text{Eq. 2-2}$$

- Hypothesis H_0: Absence of the synchronization signal

$$\underline{x}[n+k] = \underline{n}[n+k], \text{for } k = 0,\dots,N-1 \qquad \text{Eq. 2-3}$$

Here:

- N is the length of the synchronization sequence,
- n corresponds to the current time index at which the two hypotheses have to be tested,
- $d[k]$ is the synchronization sequence,
- $\underline{h} + \sum_{p\neq 0} \underline{h}_p\, z^{-p}$ is the unknown transfer function of the discrete time equivalent propagation channels between the transmitter and the M antenna of the receiver,
- $\underline{n}[n]$ represents the contribution of the background noise and of the other active cells.

The two hypotheses are composite in the sense that the joint probability distribution of the sequence $\{\underline{x}[n+k], k = 0,\dots,N-1\}$ depends on several unknown parameters. It is thus impossible to derive and implement optimum detection procedures in the most general case. In order to motivate the following sub-optimum algorithms, we first address the case where the noise $\underline{n}[n]$ is temporally white and the transfer function $\underline{h} + \sum_{p\neq 0} \underline{h}_p\, z^{-p}$ is reduced to the vector $\underline{h}$, which implicitly implies the existence of a single path propagation channel between each active base station and the receiver. In this context, it is possible to derive the maximum likelihood ratio test whose performance will be studied in the following. This test is called "The optimal spatial detector" in the following. When the propagation channels between the active base stations and the LTE sensing equipment are frequency selective, the above assumption is not motivated. We thus suggest a heuristic modification of the optimal spatial detector, as shown below, to obtain better performance.

2.6.2.2 Optimal spatial detector

We consider the following simplified hypotheses testing problem:

Hypothesis H_1:

$$\underline{x}[n+k] = \underline{h}d[k] + \underline{n}[n+k], \text{for } k = 0,\dots,N-1 \qquad \text{Eq. 2-4}$$

Hypothesis H_0:

$$\underline{x}[n+k] = \underline{n}[n+k], \text{for } k = 0,\dots,N-1 \qquad \text{Eq. 2-5}$$

where $\underline{n}[n]$ is assumed to be temporally white, but possibly spatially correlated with an unknown covariance matrix. In order to derive a relevant test, we propose to use the maximum likelihood methodology. The likelihood ratio can be written:

$$c(n) = \left(\frac{det(\mathbf{R}_0)}{det(\mathbf{R}_1)}\right)^N \frac{\exp\left(-\sum_{k=0}^{N-1}\left(\underline{x}[n+k] - \underline{h}d[k]\right)\mathbf{R}_1^{-1}\left(\underline{x}[n+k] - \underline{h}d[k]\right)^H\right)}{\exp\left(-\sum_{k=0}^{N-1}\underline{x}[n+k]\,\mathbf{R}_0^{-1}\underline{x}^H[n+k]\right)} \qquad \text{Eq. 2-6}$$

where $\mathbf{R}_0$ and $\mathbf{R}_1$ are the covariance matrices of the noise under the hypotheses H$_0$ and H$_1$. These two unknown matrices as well as the vector $\underline{\mathrm{h}}$ are nuisance parameters that have to be estimated under each hypothesis in the maximum likelihood sense:

After some easy calculations, we get that under H$_0$:

$$\widehat{\mathbf{R}}_0 = \widehat{\mathbf{R}}_{xx}(n) \qquad \text{Eq. 2-7}$$

while under H$_1$:

$$\widehat{\underline{\mathrm{h}}} = \frac{1}{\|d\|^2}\hat{\underline{\mathrm{r}}}_{xd}(n) \qquad \text{Eq. 2-8}$$

$$\widehat{\mathbf{R}}_1 = \widehat{\mathbf{R}}_{xx}(n) - \frac{1}{\|d\|^2}\hat{\underline{\mathrm{r}}}_{xd}(n)\hat{\underline{\mathrm{r}}}_{xd}^H(n) \qquad \text{Eq. 2-9}$$

with:

$$\|d\|^2 = \sum_{k=0}^{N-1} |d[k]|^2 \qquad \text{Eq. 2-10}$$

$$\hat{\underline{\mathrm{r}}}_{xd}(n) = \sum_{k=0}^{N-1} \underline{\mathrm{x}}[n+k]d^*[k] \qquad \text{Eq. 2-11}$$

$$\widehat{\mathbf{R}}_{xx}(n) = \sum_{k=0}^{N-1} \underline{\mathrm{x}}[n+k]\underline{\mathrm{x}}^H[n+k] \qquad \text{Eq. 2-12}$$

Replacing into Eq. 2-6 the vector $\underline{\mathrm{h}}$ and the matrices $\mathbf{R}_0$ and $\mathbf{R}_1$ by their estimates, we get, after some calculations that the maximum likelihood ratio is given by:

$$c(n) = \frac{\hat{\underline{\mathrm{r}}}_{xd}^H(n)\widehat{\mathbf{R}}_{xx}^{-1}(n)\hat{\underline{\mathrm{r}}}_{xd}(n)}{\|d\|^2} \qquad \text{Eq. 2-13}$$

It is interesting to remark that Eq. 2-13 can be interpreted as the correlation of the synchronization sequence with the output of the spatial filter $\hat{\underline{\mathrm{r}}}_{xd}^H(n)\widehat{\mathbf{R}}_{xx}^{-1}(n)$ driven by the received signal $\underline{\mathrm{x}}$. Indeed, $c(n)$ can be written as:

$$c(n) = \frac{1}{\|d\|^2}\sum_{k=0}^{N-1}\left(\hat{\underline{\mathrm{r}}}_{xd}^H(n)\widehat{\mathbf{R}}_{xx}^{-1}(n)\underline{\mathrm{x}}[n+k]\right) d^*[k] \qquad \text{Eq. 2-14}$$

It is worth mentioning that under hypothesis H$_1$, $\hat{\underline{\mathrm{r}}}_{xd}^H(n)\widehat{\mathbf{R}}_{xx}^{-1}(n)$ can be interpreted as an estimate of the so-called spatial Wiener filter defined as the minimum variance estimate of $d[k]$ based on the observation $\underline{\mathrm{x}}[n+k]$. Therefore, the present detector evaluates under hypothesis H$_1$ the optimum spatial mean-square estimate of sequence $d[k]$, and checks the relevance of the hypothesis by correlating this estimate with the actual sequence.

In order to improve the performance of the above test, we propose to use the periodicity of the synchronization sequences. To do so, the above hypothesis-testing problem has to be modified as follows:

Hypothesis H$_1$:

$$\underline{\mathrm{x}}[n + mT_{frame}] = \underline{\mathrm{h}}d[k] + \underline{\mathrm{n}}[n + mT_{frame}], \text{for } m = 0, \ldots, N_{frame} - 1 \qquad \text{Eq. 2-15}$$

Hypothesis H$_0$:

$$\underline{x}[n + mT_{frame}] = \underline{n}[n + mT_{frame}], \text{for } m = 0, \dots, N_{frame} - 1 \quad \text{Eq. 2-16}$$

Where:

- N_{frame} is the number of observed frames,
- T_{frame} is the duration of a frame,
- $\underline{h}$, $\mathbf{R}_0$ and $\mathbf{R}_1$ are assumed to be frame varying (i.e. depending on *m*).

The maximum likelihood test consists in comparing to a threshold the quantity:

$$\tilde{c}(n) = \frac{1}{N_{frame}} \sum_{m=0}^{N_{frame}-1} -\ln\left(1 - c(n + mT_{frame})\right) \quad \text{Eq. 2-17}$$

If all the criteria $c(n + mT_{frame})$ are small compared to 1, this test can be reduced to:

$$\overline{c}(n) = \frac{1}{N_{frame}} \sum_{m=0}^{N_{frame}-1} c(n + mT_{frame}) \quad \text{Eq. 2-18}$$

In the following, we will call "instantaneous criterion" the criterion of Eq. 2-13 and "integrated criterion" the criterion of Eq. 2-18 . We will show in the following that the integrated criterion provides an averaging effect, improving the performance of the test.

2.6.2.3 Asymptotic value of the criterion at the synchronization positions

At a synchronization position, it is possible to calculate the value of the criterion, assuming that the matrix $\mathbf{R}_{xx}$ and the vector $\underline{r}_{xd}$ are perfectly estimated. Indeed, at the synchronization position:

$$\mathbf{R}_{xx} = \pi_d \underline{h}\, \underline{h}^H + \mathbf{R}_1 \quad \text{Eq. 2-19}$$

and

$$\underline{r}_{xd} = \pi_d \underline{h} \quad \text{Eq. 2-20}$$

where π_d is the power of the sequence $d[k]$.

The instantaneous criterion becomes:

$$c(n) = \frac{\pi_d \underline{h}^H \mathrm{R}_1^{-1} \underline{h}}{1 + \pi_d \underline{h}^H \mathrm{R}_1^{-1} \underline{h}} = \frac{\mathrm{SINR_{SWF}}}{1 + \mathrm{SINR_{SWF}}} \quad \text{Eq. 2-21}$$

where the quantity $\mathrm{SINR_{SWF}} = \pi_d \underline{h}^H \mathrm{R}_1^{-1} \underline{h}$ is the signal to noise plus interference ratio at the Spatial Wiener Filter output.

2.6.2.4 False alarm probability

The false alarm probability corresponding to a threshold *S* is the probability that the criterion $c(n)$ is higher than *S* for a time position *n* which does not correspond to a synchronization position. Under the assumption that the noise $\underline{n}[n]$ is temporally white, it is possible to evaluate in closed form the probability distribution of $c(n)$ under the null hypothesis. Indeed, according to [2], under the null hypothesis, the probability distribution of $c(n)$ is equal to:

$$p(c) = \frac{(N-1)!}{(M-1)!\,(N-M-1)!}\, c^{M-1}(1-c)^{N-M-1} \quad \text{Eq. 2-22}$$

where M represents the number of antennas of the array.

The false alarm probability can then be computed as:

$$\mathrm{PFA_c}(S) = \int_0^1 p(c)\mathrm{d}c \qquad \text{Eq. 2-23}$$

As for the integrated criterion, an approximated analytical formula giving the false alarm probability can still be derived by remarking that the probability distribution in Eq. 2-22 can be approximated by:

$$p(c) = \frac{N^M}{(M-1)!} c^{M-1} e^{-Nc} \qquad \text{Eq. 2-24}$$

which corresponds to a χ^2 distribution with *2M* degrees of freedom and an expectation equal to . Then the distribution of the mean of independent instantaneous criteria can easily be deduced as:

$$p(c) = \frac{N^M}{(M-1)!} c^{M-1} e^{-Nc} \qquad \text{Eq. 2-25}$$

Then we get the false alarm probability for the integrated criterion:

$$\mathrm{PFA_{\bar{c}}}(S) = \int_0^1 p_{N_{frame}}(c)\mathrm{d}c \qquad \text{Eq. 2-26}$$

The detection of the PSS is achieved by computing the instantaneous synchronization criterion described in Eq. 2-13 at each time position. If, as usual, the signal is observed over several frames, it is better to take advantage of the PSS periodicity using the Eq. 2-18 integrated criterion.

2.6.2.5 Application to LTE standard

On the downlink, LTE is an OFDMA (Orthogonal Frequency Division Multiple Access) system with an inter-carrier spacing of 15 kHz or 7.5 kHz. **In this document only the 15 kHz case will be studied.** Depending on the target service, the data rate can be adjusted by changing the bandwidth as specified in Table 3.

Table 3 : LTE downlink characteristics depending on channel bandwidth

Channel bandwidth [MHz]	1.4	3	5	10	15	20
FFT size	128	256	512	1024	1536	2048
Number of subcarriers (excluding DC)	72	180	300	600	900	1200
Occupied bandwidth [MHz]	1.095	2.715	4.515	9.015	13.515	18.015
Maximum data rate [Mbps]	5.76	14.4	24.0	48.0	72.0	96.0
Sampling rate [Mcps]	1.92	3.84	7.68	15.36	23.04	30.72

The OFDM symbols are gathered in slots of 0.5 ms, while a frame consists of 20 slots (10 ms). The number of OFDM symbols in a slot depends on the length of the CP (Cyclic Prefix). Indeed, two modes are possible depending on the propagation condition:

- The normal CP mode in which a slot contains 7 OFDM symbols (the length of the first OFDM symbols CP is longer in order to keep the slot duration constant).
- The extended CP mode in which a slot contains 6 OFDM symbols.
- The frame structure is summarized in Figure 13, in the 5 MHz bandwidth case.

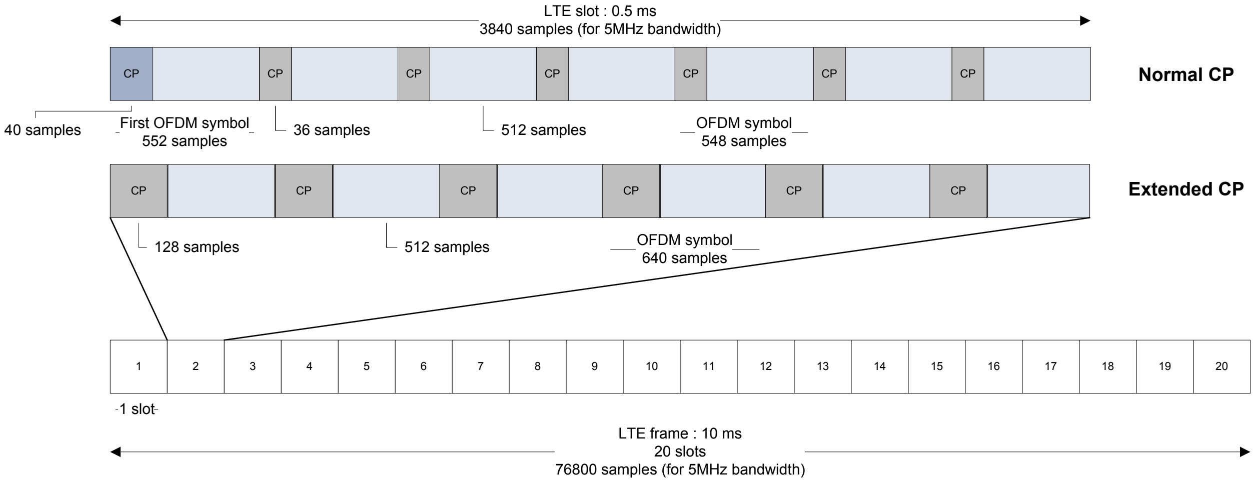


Figure 13: LTE frame structure in the 5 MHz bandwidth case

Synchronization sequences

In order for the UE to get synchronized with the LTE network, two synchronization sequences are broadcast by the BTS:

- The Primary Synchronization Signal (PSS)
- The Secondary Synchronization Signal (SSS)

The detection of these two signals not only enables time synchronization but also provides the physical layer identity of the cell and the cyclic prefix length, and informs the UE whether the cell uses FDD or TDD.

The PSS and SSS structure in time is shown in Figure 14 in the FDD case and in Figure 15 in the TDD case. Both synchronization signals are transmitted twice per frame. For the PSS, the same sequence is transmitted each time. For the SSS, a different sequence is sent on slot 1 and 11, in the FDD case (or on slot 2 and 12 on the TDD case). The sequences that are transmitted for the PSS and SSS in a given cell are used to indicate the physical layer cell identity. There are 504 physical layer cell identities split into 168 groups of 3 identities. Three different PSS are used to indicate the cell identity within the group and 168 SSS are used to indicate the identity of the group. The goals of the synchronization signals are thus:

- For the PSS, to acquire the slot timing synchronization and the cell identity within the group. Three different PSS have to be tested.
- For the SSS, to acquire the frame timing synchronization, the identity of the group and therefore the physical layer identity of the cell, the CP length and the duplex mode. 168 different SSS have to be tested on 8 different timing positions (2 for the CP length × 2 for the duplex mode × 2 for the frame synchronization).

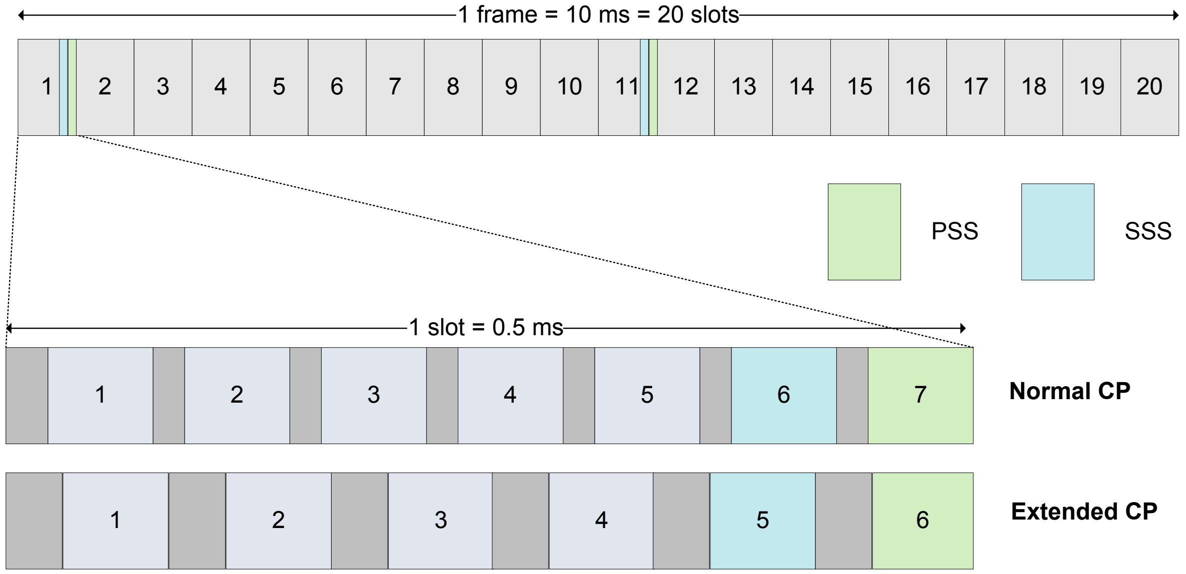


Figure 14: PSS and SSS frame structure in the FDD case

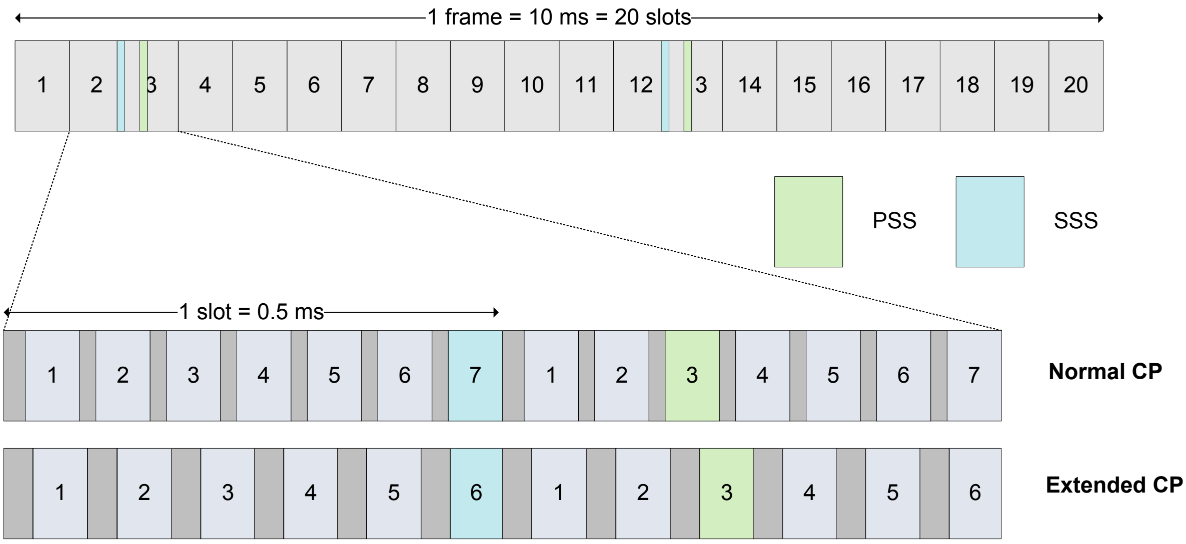


Figure 15: PSS and SSS frame structure in the TDD case

In the frequency domain, the PSS and SSS are transmitted on the 62 central subcarriers allowing the synchronization to be performed independently of the signal bandwidth.

The detection of the primary synchronization signal is achieved by computing the criterion described in Eq. 2-18 at each time position over half a frame (which is the period of the PSS). Three criteria must be computed with three different PSS corresponding to the cell identity within the group. The three criteria are then compared to a threshold. Each position for which one of the criteria is greater than the threshold is considered as a possible detection of a primary synchronization sequence transmitted by a base station. This allows to detect the most powerful stations and to get the slot synchronization and the cell identity within the group.

The key point is here to choose the value of the threshold: it has to be chosen in order to achieve an acceptable false alarm rate, while providing satisfying probability detection. In general, the Eq. 2-26 is not relevant because it is based on very strong assumptions, which are not verified in practice. Therefore, we propose in the following to evaluate the threshold by means of Monte-Carlo simulations.

The detection of the secondary synchronization sequence is achieved using the same algorithm for each time position at which a PSS has been detected. For each time position, 168 different SSS have to be tested, in order to identify the identity of the group. Each SSS is tested on 8 different timing positions (2 for CP length deduction × 2 for duplex mode deduction × 2 for frame synchronization deduction). On the 168×8 criterion values, the maximal one is taken. No thresholding is applied in this step.

Moreover, it is at this step that the multiple paths are handled. Indeed, during the detection of the PSS, two time positions above the threshold can correspond to two paths coming from the same base station. If after the secondary synchronization, we observe that two possible detected base stations have the same physical layer cell identity, we will conclude that it is in fact two different paths coming from the same base station.

3 Common Data Collection/Storage Methodology Design

3.1 Introduction

Many usage scenarios that take place can be 'recorded'. In our federation data recorded in one testbed will be usable in other testbeds to support emulated usage scenarios (e.g. primary user data recorded in testbed A feeds into a sensing device in testbed B). To this end this task will define data of interest, common structures for storing data and create a federation database for storage of any collections made. The availability of this 'fluid' data will also be of key interest in the benchmarking processes in WP4 as it will enable us to compare and contrast how different approaches deal with given data sets. Making the data openly available in itself is also of huge importance as the data can be used to validate theoretical ideas or indeed as actual input to systems being tested, emulated, or simulated. Whenever possible, we will contribute collected data to open repositories. FARAMIR plans to build environmental maps of primary users and the CREW data can potential contribute to this. There is the potential to also contribute outside the EU to such repositories as CRAWDAD, a Community Resource for Archiving Wireless Data At Dartmouth, in the USA. To underpin this work, data logging facilities will need to be added to some testbeds. Task 4.4 in WP4 will also have an impact here.

3.2 Background on IEEE 1900.6

The IEEE 1900.6 standard "defines the information exchange between spectrum sensors and their clients in radio communication systems. The logical interface and supporting data structures used for information exchange are defined abstractly without constraining the sensing technology, client design, or data link between the sensor and client." [4] For the definition of common data formats in the CREW project, especially the definition of supporting data structures is of interest.

The scope of the 1900.6 standard is limited to the definition of spectrum sensing related parameters and data structures. However, since a significant part of the data relevant for the CREW common data format is also spectrum sensing related, we decided to use the 1900.6 definitions where applicable and extend the available data structures where necessary.

Clause 6 "Information description" of the IEEE 1900.6 standard [4] contains the information relevant to the CREW common data format. Here also a list and definitions of the defined data structures can be found. Within the CREW common data format task we extend the 1900.6 definitions with the additional data structures required. Such additional data structures are, e.g., related to meta information, experiment specification, and non-sensing related parameters such as throughput, bit/frame error rates, etc.

3.3 Definition of the data of interest

The results of an experiment should be stored in such a way that it is possible to utilize the outcome further for our CREW-specific goals (analysis & benchmarking, performance comparison of different CR equipment, "replaying" etc.). To structure this process we group all relevant information into the following 3 categories 1. "experiment abstract", 2. "meta-information" and 3. "experiment-trace(s)".

Typically experimenters make several iterations of an experiment and obtain several traces. Some of these iterations may have involved slight changes in the experimental setup. These changes are often not enough to justify calling it a new experiment. Our specification allows us to define a common set of meta-information (the devices involved, the location, etc.) which can be “refined” for each individual trace (iteration). Specifically, the experimenter defines a template (in category 2) which is valid, unless it is overwritten in the specification of an individual trace (in category 3).

The suggested file format for experiment specification and meta-information is JSON, which can be transcoded to XML but is easier human-readable. We plan to offer a web interface with forms that the user fills in and produce the JSON representation automatically (this would likely be implementation work for CREW Year 2).

Measurement traces may be in a proprietary format, as long as they easily are convertible to CSV (with open processing tools). We recommend Matlab mat-files as our preferred file format, as the "Experiment abstract and meta-information" of an experiment can then also be added in each file for convenience. For this we plan to define Matlab structures (a template) to be used and provide a tool that parses the JSON representation into Matlab structures, so a user will only need to enter the information once. If the trace file is too big it can be split into parts, which has to be explained.

In the following we present a template for our envisioned data structure with additional "best practice" comments. Examples for data from different internal usage scenarios of the CREW project can be found in Section 3.4. Examples using the json format can be found in Appendix B: BEE2 example .json.

1 **Experiment Abstract:** This is the structured description of the entire experiment. It provides all basic information for understanding how the experiment was performed and who did it. This information is valid for the entire experiment set.

1.1 Title: A few words with the core information.

1.2 Unique CREW Tag: The tag for further references to other CREW experiments or publications.

1.3 Author: List of all experiment authors.

1.3.1 Name

1.3.2 Email

1.3.3 Address

1.3.4 Phone

1.4 Release Date: The release date of the experiment. It should be kept as specified in ISO 8601.

1.5 Experiment summary: Detailed textual description of the experiment. What was done, why, what should be expected. How many experiment iterations were performed…

1.6 Collection methodology: Textual description of how the data was collected.

1.7 Further documentation:

1.7.1 Description: Short description of the related documentation.

1.7.2 Bibtex: The list of bibtex entires for the related publications.

1.8 Related experiments: The list of the related experiments in the CREW database. For example if there are new or previous experiments based on this, experiments closely related to each other. Here also information on the relation should be added, e.g. "repetition of experiment on a larger scale", or "repetition due to problems in previous experiments"

1.9 Notes: Case specific experiment notes. For example problems with data that was discovered during evaluation or known limitations for further usage of the data

2 **Meta-information:** Meta-information is "the information required for describing, understanding, and evaluating information". This is very well described in IEEE 1900.6 in Section 5.3.1 and in Table on page 18. To give IEEE 1900.6 support it is necessary to provide support for all parameters defined there. There is also an interface for retrieving data from sensor itself or Data Archive (DA from IEEE 1900.6)

The information in this category may be refined in category 3 with details per individual trace.

2.1 Device: Detailed description of the device types involved in the experiment.

2.1.1 Name: Device name

2.1.2 Description: Short textual description of the used device

2.1.3 Datasheets: Links to the device datasheets. The specific datasheet information is also required for access in IEEE 1900.6.

2.1.4 Software: Specific software (hopefully with source code) used for measurements. For example custom USRP code.

2.1.4.1 Description: What kind of software was used, what's the execution environment, etc.

2.1.4.2 Operating system: The operating system used on the device. It is possible that hardware supports multiple operating systems, firmware version, in such case it is necessary to specify which one was used.

2.1.4.3 Application Name: Names of the applications used.

2.1.4.4 Code: References to the source code to download.

2.2 Location: A description of the area where the measurement was performed. Was it indoors/outdoors, location map, placement of the devices in the building, etc.

2.2.1 Layout: Arrangement of devices and other objects in the environment.

2.2.2 Mobility: Mobility information of devices or objects in the environment.

2.3 Time: general date / time range, will be refined in category 3 per trace.

2.4 Radio Frequency

2.4.1 Operating range: The frequency range on which the devices are operating. For example 2.4 GHz ISM band.

2.4.2 Interference sources: Possible sources of interference.

2.5 Parameters: Definition of parameters that can be changed between different iterations of the experiment.

2.5.1 Description

2.5.2 Name: Name of the parameters. For further usage in iteration description.

2.5.3 Unit: Unit of measurement for the parameter.

2.6 Trace description

2.6.1 Description: Detailed description of the generated trace files. It is left open in which format the actual measurement trace is stored as long as it is sufficiently well described and can be converted to CSV, which we consider the baseline. We also recommend other formats, such as matlab timeseries objects.

2.6.2 Collected metrics: List of the types of collected data. For example signal power, time, location.

2.6.2.1 Name

2.6.2.2 Unit of Measurement: The unit used for data storage. For example Hz, dBm. The measurement units (frequency, power etc.) are described in IEEE 1900.6 in Section

6.2 on page 73 and should be referenced here)

2.6.2.3 Accuracy: Information about device accuracy for this metric. Should be extracted from datasheet.

2.6.3 Processing tools: Conversion, evaluation, etc.

2.7 Signal generation: If any how the signal was generated, trace, source files.

2.7.1 Description: Description how signal was generated.

2.7.2 Trace: Trace file of the generated file suitable for replaying the experiment.

3 **Experiment Iterations:** Each trace may include sensing data, but may also cover other parameters if they were dynamic and captured during the experiment (time or location data). A measurement trace must conform to the specification above in category 1. It may deviate from the specification in category 2, but then it must by explicitly stated below in which ways it differs, i.e. the specification for an individual trace can overwrite/refine/augment any of the existing fields described in category 2)

3.1 Description: A brief description/relation to the other traces, e.g. this was the n^{th} iteration of the experiment.

3.2 Time: The actual start and end time of this experiment trace.

3.2.1 Start time

3.2.2 End time

3.3 Parameters: Values of the parameters defined in category 2.

3.3.1 Name

3.3.2 Value

3.4 Trace-file: List of trace files containing the traces in the format described in 2.5.3

3.5 If necessary the fields from category 2 can be redefined here. For example the location information if changed during the experiment. The change can be represented here without the need of creation of the new description.

3.4 Examples

3.4.1 BAN Example

1 Experiment Abstract

1.1 Title: Urban RF noise measurements with a IEEE 802.15.4 Body Area Network

1.2 Unique CREW Tag: 2011-1-Hauer

1.3 Author(s): Jan Hauer

1.3.1 Contact information : hauer@tkn.tu-berlin.de, Einsteinufer 2*5, 10587* Berlin, Germany

1.4 Release Date: 2011.06.25 I would suggest using uniform date time encoding across the entire CREW data: here, in 2.3 and in measurements. My suggestion would be UTC: 1994-11-05T13:15:30Z (see here for more http://www.w3.org/TR/NOTE-datetime)

1.5 Experiment summary: *2.4 GHz RF noise measurements on 3 shimmer2r sensor nodes attached to a person, who was walking through a central shopping district in Berlin, Germany. The subject was walking for 30 minutes, we monitored RSSI on all 16 IEEE 802.15.4 channels (2400, 2405, ... MHz) in a round-robin fashion. Because we were* *monitoring the unlicensed 2.4 GHz band there are likely many signals from different devices. Our setup is completely passive, simply recording RF noise and location data.*

1.6 Collection methodology: *RF noise samples are collected by periodically reading the RSSI register of the shimmer2r radio (CC2420 radio). An RSSI reading represents the average signal power over 192 microseconds in dBm. Once an RSSI reading has been obtained, we switch to the next channel (2400->2405->…->2480->2400->2405 MHz, etc.) and collect the next sample. All RSSI values are stored on an SD card and extracted after the experiment. Location data is collected via a GPS-daughterboard, which allows to obtain GPS coordinates and time once per second. GPS location and time data is also stored on the SD card.*

1.7 Further documentation : *none*

1.8 Related experiments: *none*

1.9 Notes: *During the first 30 seconds of an experiment the GPS data is unavailable, because the GPS needs time for calibration.*

2 Meta-information

2.1 Devices

2.1.1 Datasheets:

- *Shimmer2r platform:* http://www.shimmer-research.com/wp-content/uploads/2011/06/Shimmer-2R-Technical-Data-Sheet.pdf
- *GPS board:* http://www.shimmer-research.com/wp-content/uploads/2011/02/GPS-Spec-Sheet.pdf
- *CC2420 Radio:* http://www.ti.com/lit/gpn/cc2420

2.1.2 Software

2.1.2.1 Description: maybe it would be nice to have the information here a little bit more structured

2.1.2.2 *Operating system: TinyOS 2*

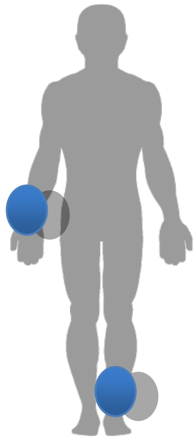
2.1.2.3 *Driver: shimmer2r GPS vX.X*

2.1.2.4 *Application: ApplicationName*

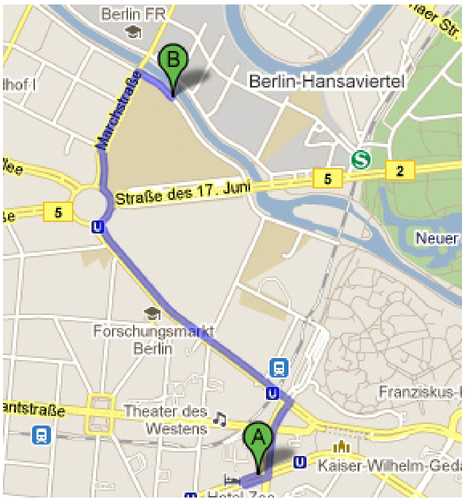
2.1.2.5 Code: *Language: nesC URI: http://github.com/XXXX*

2.2 Space

2.2.1 Layout: *Sensor nodes are attached to right hand and left foot of the subject as shown in the following schematic:*



The subject was walking on the pedestrian path from Kurfuerstendamm 192, 10879 Berlin, Germany, to Einsteinufer 25, 10551 Berlin, taking the route as shown in the following picture:



It was a typical urban main street environment with cars passing frequently and other pedestrians walking on the pedestrian path. There were houses next to the streets, a mix between apartment building and office buildings. The path can be seen in google-streetview: http://maps.google.de/maps?saddr=Kurf%C3%BCrstendamm,+Charlottenburg+10789+Berlin&daddr=Einsteinufer+26,+10587+Berlin&hl=de&ie=UTF8&ll=52.515646,13.326845&spn=0.019482,0.055661&sll=52.501646,13.311052&sspn=0.038716,0.111322&geocode=Fd0gIQMdUVPLACn3VCVx_1CoRzFZUEg_VssRdQ%3BFRlUIQMdHlnLACkxTgW3BFGoRzG28WgySvDIFg&mra=iwd&z=15&layer=c&cbll=52.515978,13.326451&panoid=8GDeVURfbqcbmeSuxI1aAQ&cbp=12,332.11,,0,13.95

2.2.2 Mobility: *could be Fixed, Mobile or Fixed and Mobile. Depending on the case, Speed could be 0 km/h (Stationary), 4 km/h (human walking speed), etc. This would again bring in a little more structure which could further be used to link to process vocabularies for describing experiments.*Execution:

Iteration 1: StartTime: 2011-05-22T17:15:00Z EndTime: XXX DurationTime: 785msec

Iteration 2: StartTime: 2011-05-22T18:15:00Z EndTime: YYY DurationTime: 900msec

2.3 RF Frequency

2.3.1 Operating range(s): *2.4 GHz ISM Band, 2400 – 2483 MHz*

2.3.2 Interference sources: *There was likely uncontrolled interference from various 2.4 GHz band devices. Measuring the power of their signal was a main goal of the experiments.*

2.4 Trace description

2.4.1 Collected metrics

2.4.1.1 *RF Power*

2.4.1.1.1 *on one of 16 IEEE 802.15.4 channels averaged over 192us*

2.4.1.1.2 *UoM: dBm*

2.4.1.1.3 *+- 6 dB (RSSI accuracy) datasheet*

2.4.1.2 *Time (GPS time)*

2.4.1.2.1 *in unix time + microseconds*

2.4.1.2.2 *+-40 ppm accuracy*

2.4.1.3 *Location*

2.4.1.3.1 *GPS coordinates*

2.4.1.3.2 *+- 5m (GPS accuracy)*

2.4.2 Format: each experiment corresponds to two standard matlab timeseries objects with the following properties (timeseries struct members):

a)

- Name: 'RSSI on node in right hand'
- Time: [nx1 double]
- TimeInfo: [1x1 tsdata.timemetadata]
  - Units: seconds.microseconds
- Data: [nx1 double]
- DataInfo: [1x1 tsdata.datametadata]
  - Units: ”dBm”

b)

- Name: 'RSSI on node on left foot'
- Time: [mx1 double]
- TimeInfo: [1x1 tsdata.timemetadata]
  - Units: seconds.microseconds
- Data: [mx1 double]
- DataInfo: [1x1 tsdata.datametadata]
  - Units: ”dBm”

c)

- Name: ‘GPS Location’
- Time: [nx1 double]
- TimeInfo: [1x1 tsdata.timemetadata]
  - Units: seconds.microseconds
- Data: [nx1 double]
- DataInfo: [1x1 tsdata.datametadata]
  - Units: ”GPS coordinates”

2.4.3 Processing tools: none

2.4.4 Signal generation: none

3 Experiment Trace(s)

Trace 1:

Description: *first iteration of the experiment*

Time: *22.05.2011 at 17:15-18:15 CET*

Trace-specific meta-information : *none*

Trace-file: *LINK*

Trace 2:

Description: *second iteration of the experiment*

Time*: 23.05.2011 at 16:05-17:05 CET*

Trace-specific meta-information:

Space.layout: in addition to the two nodes attached to right hand and left foot we used a third node attached to the chest

Format: we have an additional RSSI trace for the new node

Trace-file: LINK

3.4.2 BEE2 Example

The calibration process requires two steps. First the transmitter is calibrated and than the receiver. We consider this as two separate experiments because the two different devices are used for measurements and both give output in completely different formats. Both experiments are still tightly connected to each other. That is why both are mentioned in each other references, and most of the description is common.

1 Experiment Abstract

1.1 Title *Transmitter calibration of the radio Front Ends for BEE2*

1.2 Unique CREW Tag: *2011-1-Chwalisz*

1.3 Author(s): *Mikołaj Chwalisz*

1.3.1 Contact information: chwalisz@tkn.tu-berlin.de*, Einsteinufer 25, 10587 Berlin, Germany*

1.4 Release Date: *07.04.2011*

1.5 Experiment summary: *The calibration is a process aimed to give a meaningful comparison between measurements made by one device, with known magnitude and correctness, and a second device. This step is essential to be able to compare results with other experiments, especially with custom made devices. The other goal of the calibration is to determine the condition of the instrument to perform measurements. This also includes the ability to transfer defined measurement units.*
In order to calibrate the receiver, it is necessary to have a calibrated transmitter.

In this experiment we try to calibrate BEE2 Front End as the transmitter based on signal received by the R&S FSV Spectrum Analyzer

1.6 Collection methodology: *Devices where set to one frequency and the power level of the generic OFDM was measured. Whole experiments where done with cable connection. Transmitting device is set to one center frequency*

1.7 Further documentation: *The measurements where published in master thesis of Mikołaj Chwalisz.*

1.7.1 Bibtex: *@MastersThesis{ chwalisz2010mscthesis, title = "{Development of a testbed for spectrum diversity measurements in the ISM band}", author = "Miko{\l}aj Chwalisz", school = {Warsaw University of Technology \& Technische Universit{\"a}t Berlin}, month = "March", year = "2011"}*

1.8 Related experiments: *This is part of calibration process of BEE2 Radio Front Ends. The other part has CREW Tag: 2011-2-Chwalisz*

1.9 Notes

2 Meta-information

2.1 Devices*:*

- *BEE2 Board: The Berkeley Emulation Engine 2 (BEE2) was developed to be a reusable, modular, and scalable framework for designing high-end reconfigurable computers at the Berkeley Wireless Research Center (BWRC). It is supposed to help solving computationally intensive problems such as: emulation and design of wireless communication systems, real-time scientific computation, high-performance real-time digital signal processing*
- *Radio Front End: The radio capabilities for BEE2 board in the CogRad testbed are provided by the radio Front End. It is made of the baseband board performing data processing, control and digital to analog conversion. The daughter card is used to perform up/down signal conversion to 2.4 GHz.*
  http://bwrc.eecs.berkeley.edu/Research/Cognitive/prototyping_platform.htm
- R&S FSV Spectrum Analyzer

2.1.1 Datasheets:

- *BEE2: C. Chang, J. Wawrzynek, and R.W. Brodersen, BEE2: a high-end reconfigurable computing system, Design Test of Computers, IEEE 22 (2005), no. 2, 114–125.*
  http://bee2.eecs.berkeley.edu/wiki/BEE2wiki.html
  http://bee2.eecs.berkeley.edu/
- *Front End: Fred Burghardt, Cognitive Radio Testbed Users Manual, April 2009.*
  http://bwrc.eecs.berkeley.edu/Research/Cognitive/prototyping_platform.htm
- *R&S FSV:* http://www2.rohde-schwarz.com/file/FSV_dat-sw_en.pdf

2.1.2 Device Software

2.1.2.1 Description: MSSGE (Matlab / Simulink / System Generator / EDK) toolchain for FPGA designs. Used CASPER libraries and code from BWRC (Berkeley)

2.1.2.2 Code Software like FPGA designs is available, please contact author.

2.2 Space

2.2.1 Layout: Cable connection between devices.

2.2.2 Mobility: None

2.3 Time *Couple of seconds per measurement*

2.4 RF Frequency

2.4.1 Operating range(s) *2.4 GHz ISM band, 2400 – 2483 MHz*

2.4.2 Interference sources: *None, cable, the no loss connection is assumed*

2.5 Trace description

2.5.1 Collected metrics: *Power measurements in dBm, detector and trace mode of spectrum analyzer defined in every trace file.*

2.5.2 Data accuracy: *Total measurement uncertainty: 0.28dB*

2.5.3 Format *text file with the following structure:*

- *Parameter listing:*
    - *Name; Value; (Unit)*
    - *Values;Number of values;*
    - *Vector: Frequency;dBm*
- *Additional PNG file with spectrum analyzer screen shot*

2.5.4 Processing tools: *Basic analysis GUI and matlab scripts available, please contact author.*

2.5.5 Signal generation: *For signal generation the FE was used. One or two OFDM symbols stored in FE's FPGA fabric and send repeatedly. Resulting in constant OFDM stream. Matlab file with I/Q samples is available as well as the scripts to create it.*

3 Experiment Trace(s)

3.1 Trace 1:

3.1.1 Description *10dB Attenuator added into cable*

3.1.2 Time *20.01.2011 at 16:05 CET*

3.1.3 Trace-specific meta-information *none*

3.1.4 Trace-file *fec att10dB count500 swt1ms clrw.DAT*

3.2 Trace 2:

3.2.1 Description *signal was averaged over 500 sweeps*

3.2.2 Time *20.01.2011 at 16:15 CET*

3.2.3 Trace-specific meta-information *none*

3.2.4 Trace-file *fec att0dB count500 swt1.1ms rbw100khz avg.DAT*

3.4.3 Receiver calibration

1 Experiment Abstract

1.1 *Title Receiver calibration of the radio Front Ends for BEE2*

1.2 *Unique CREW Tag: 2011-2-Chwalisz*

1.3 *Author(s): Mikołaj Chwalisz*

1.3.1 *Contact information: chwalisz@tkn.tu-berlin.de, Einsteinufer 25, 10587 Berlin, Germany*

1.4 Release Date: *07.04.2011*

1.5 Experiment summary: *The calibration is a process aimed to give a meaningful comparison between measurements made by one device, with known magnitude and correctness, and a second device. This step is essential to be able to compare results with other experiments, especially with custom made devices. The other goal of the calibration is to determine the condition of the instrument to perform measurements. This also includes the ability to transfer defined measurement units.*
In order to calibrate the receiver, we take the knowledge of the signal strength of the Front End from experiment 2011-1-Chwalisz and take it as the input for receiver calibration.

1.6 Collection methodology: *Devices where set to one frequency and the power level of the generic OFDM was measured. Whole experiments where done with cable connection. Transmitting device is set to one center frequency*

1.7 Further documentation: *The measurements where published in master thesis of Mikołaj Chwalisz.*

1.7.1 Bibtex: *@MastersThesis{ chwalisz2010mscthesis, title = "{Development of a testbed for spectrum diversity measurements in the ISM band}", author = "Miko{\l}aj Chwalisz", school = {Warsaw University of Technology \& Technische Universit{\"a}t Berlin}, month = "March", year = "2011"}*

1.8 Related experiments *This is part of calibration process of BEE2 Radio Front Ends. The other part has CREW Tag: 2011-1-Chwalisz*

1.9 Notes

2 Meta-information

2.1 Devices*:*

- *BEE2 Board: The Berkeley Emulation Engine 2 (BEE2) was developed to be a reusable, modular, and scalable framework for designing high-end reconfigurable computers at the Berkeley Wireless Research Center (BWRC). It is supposed to help solving computationally intensive problems such as: emulation and design of wireless communication systems, real-time scientific computation, high-performance real-time digital signal processing*

- *Radio Front End: The radio capabilities for BEE2 board in the CogRad testbed are provided by the radio Front End. It is made of the baseband board performing data processing, control and digital to analog conversion. The daughter card is used to perform up/down signal conversion to 2.4 GHz.* http://bwrc.eecs.berkeley.edu/Research/Cognitive/prototyping_platform.htm

2.1.1 Datasheets:

- *BEE2: C. Chang, J. Wawrzynek, and R.W. Brodersen, BEE2: a high-end reconfigurable computing system, Design Test of Computers, IEEE 22 (2005), no. 2, 114–125.* http://bee2.eecs.berkeley.edu/wiki/BEE2wiki.html http://bee2.eecs.berkeley.edu/

- *Front End: Fred Burghardt, Cognitive Radio Testbed Users Manual, April 2009.*

http://bwrc.eecs.berkeley.edu/Research/Cognitive/prototyping_platform.htm

2.1.2 Device Software

2.1.2.1 Description*: MSSGE (Matlab / Simulink / System Generator / EDK) toolchain for FPGA designs. Used CASPER libraries and code from BWRC (Berkeley).*

2.1.2.2 Code *Software like FPGA designs is available, please contact author.*

2.2 Space

2.2.1 Layout*: Cable connection between devices.*

2.2.2 Mobility: *None*

2.3 Time *Couple of seconds per measurement*

2.4 RF Frequency

2.4.1 Operating range(s) *2.4 GHz ISM band, 2400 – 2483 MHz*

2.4.2 Interference sources: *None, cable, the no loss connection is assumed*

2.5 Trace description

2.5.1 Collected metrics: *Calculated FFT data from I/Q measurements in dB. Exact relation to the dBm is to be defined by this experiment.*

2.5.2 Data accuracy:

2.5.3 Format *matlab file with the following structure:*

- *frequency: Center frequency [double]*
- *Unit: MHz*
- *fe_id: [double]*
- *name: file name [string]*
- *fe: front end name [string]*
- *fs: sampling frequency [double]*
- *Unit: Hz*
- *spectrum: FFT series [MxN double]*
- *Unit: dB*
- *frequency_series: [Mx1 double]*
- *Unit: Hz*

2.5.4 Processing tools: *Basic analysis GUI and matlab scripts for loading the data is available*

2.5.5 Signal generation: *For signal generation the FE was used. One or two OFDM symbols stored in FE's FPGA fabric and send repeatedly. Resulting in constant OFDM stream. Matlab file with I/Q samples is available as well as the scripts to create it. Refer also to Experiment 2011-1-Chwalisz*

3 Experiment Trace(s)

3.1 Trace 1:

3.1.1 Description *10dB Attenuator added into cable, FE gains set to:*

- *agc = 130*
- *pga = 14*

3.1.2 Time *20.01.2011 at 16:55 CET*

3.1.3 Trace-specific meta-information *none*

3.1.4 Trace-file *memdump_0_FEA_agc_130_pga_14_att10db_b.fft*

3.2 Trace 2:

3.2.1 Description *10dB Attenuator added into cable, FE gains set to:*

- *agc = 130*
- *pga = 14*

3.2.2 Time *20.01.2011 at 17:10 CET*

3.2.3 Trace-specific meta-information *none*

3.2.4 *Trace-file memdump_0_FEA_agc_130_pga_14_att0db_b.fft*

3.4.4 Dublin Sensing Experiment

1 Experiment Abstract

1.1 Title: Sensing of DVB-T signals transmitted in the 2.4 GHz ISM band with a range of sensing devices

1.2 Unique CREW Tag: 2011-1-sensing_dublin

1.3 Author(s): Sofie Pollin, Peter Van Wesemael

1.3.1 Contact information : pollins@imec.be, Kapeldreef 75, 3001 Leuven, Belgium

1.4 Release Date: xxxx.xx.xx

1.5 Experiment summary: *An 8 MHz DVB-T signal is transmitted in the 2.4 GHz ISM band in a large meeting room at CTVR in Dublin. The goal is to familiarize with the results and output formats of various sensing solutions. Some of those are capable of sensing in the 2.4 GHz ISM band only. Some of them have special algorithms for detecting DVB-T signals. Hence, the decision to sense DVB-T signals in the 2.4 GHz ISM band. The DVB-T signals were transmitted at various transmit powers. Also, there were scenarios with users present, without users present, and with users walking in the meeting room. Ambient interference from other devices in the 2.4 GHz ISM band is present during the experiment.*

1.6 Collection methodology: *Sensing is done with a range of sensing solutions:*

- *imec Advanced Spectrum Sensing*

Low power/low cost SDR RFIC prototype

Input range from 0.1 up to 6 GHz

Programmable channel bandwidth from 1 up to 40 MHz

On-chip 65MS/s 10b ADC

5 mm^2 – 40nm TSMC technology

- *USRP1 (Ettus Research)*

Highly flexible low cost RF transceiver.

For these experiments RFX2400 daughterboard used. It operates between 2.3 and 2.9 GHz.

Can sample up to 8Msamples/sec.

Experiment parameters

- *Iris*

Component based architecture for software defined radio

Designed and developed in CTVR, Trinity College Dublin

Highly reconfigurable

Parameters and components of radio can be changed in real time.

USRP1 front-end used in experiments

- *Wi-Spy 2.4x (MetaGeek, LLC.)*

Low-cost spectrum sensor for 2.4 GHz ISM band

We used Kismet Spec-tools for Linux OS to acquire power spectral density estimates in a non-proprietary format

Spectrum dumps are performed as fixed bandwidth sweeps of the entire ISM 2.4 GHz band

- *AirMagnet Spectrum XT*

USB product designed for troubleshooting and deploying WLAN networks

ISM 2.4 GHz/ 5 GHz

internal or external antenna

- *TelosB*

Sensor network hardware platform developed at UC Berkeley

Uses the IEEE 802.15.4-compliant CC2420 transceiver, which can measure RF energy in 2.4 GHz ISM band

IEEE 802.15.4 channel (resolution) bandwidth is 2 MHz,

Traces were collected for a range of scenarios:

- *Slow On/Off Pattern (60 s On / 60 s Off)*
- *Fast On/Off Pattern (10 ms On / 100 ms Off)*
- *Change of TX Power (-4 dBm / -15 dBm / -30 dBm)*
- *Change of Distance between TX and Sensing Nodes*
- *Change of Center Freq. (2.404 GHz : 8 MHz : 2.496 GHz)*

1.7 Further documentation : *Our setup and data is also described in paper '*Christoph Heller, Stefan Bouckaert, Ingrid Moerman, Pollin Sofie; Van Wesemael Peter, Danny Finn, Daniel Willkomm, Jan-Hinrich Hauer, "A Performance Comparison of Different

Spectrum Sensing Techniques, “ WinnComm 2011.’

1.8 Related experiments: *Follow-up experiments in Dresden and Dublin with CREW Tag: 2011-4-sensing_dresden and 2011-6-sensing_berlin*

1.9 Notes:

2 Meta-information

2.1 Devices

2.1.1 Datasheets:

- ***imec Advanced Spectrum Sensing***
  The imec spectrum sensing solution is a prototype, hence no datasheet is available. More information can be found in these leaflets and publications:
  http://www2.imec.be/content/user/File/Brochures/GR2011_Leaflet_Spectral%20Sensing.pdf
  http://www2.imec.be/content/user/File/Brochures/GR2010_Leaflet%20Scaldio.pdf
  http://www2.imec.be/content/user/File/Brochures/GR2011_Leaflet_COBRA.pdf
  M. Ingels et al. A 5mm2 40nm LP CMOS Transceiver for a Software-De_ned Radio Platform. IEEE Journal of Solid-State Circuits, 45(12):2794{2806, 2010.
- ***USRP1 (Ettus Research)*** http://www.ettus.com/downloads/ettus_ds_usrp_v7.pdf
  http://www.ettus.com/downloads/ettus_daughterboards.pdf
- ***Iris***
  L. E. Doyle, P. Sutton, K. Nolan, B. Ozgul, J. Lotze, T. Rondeau, S. Fahmy, H. Lahlou, and L. A. DaSilva, ***“Experiences from the Iris testbed in dynamic spectrum access and cognitive radio experimentation****,” IEEE Symposia on New Frontiers in Dynamic Spectrum Access Networks (DySPAN), Singapore, 6-9 April, 2010.*
  Paul Sutton, Jorg Lotze, Hicham Lahlou, Baris Ozgul, Keith Nolan, Linda Doyle, Suhaib Fahmy, Juanjo Noguera, "Multi-Platform Demonstrations using the Iris Architecture for Cognitive Radio Network Testbeds", Crowncom 2010, Cannes, France

- ***Wi-Spy 2.4x (MetaGeek, LLC.)***
  http://files.metageek.net/marketing/Wi-Spy_2.4x/Wi-spy_24x_medium.pdf
- ***AirMagnet Spectrum XT***

  http://airmagnet.flukenetworks.com/assets/datasheets/AirMagnet_SpectrumXT_Datasheet.pdf
- ***TelosB***

  http://www.willow.co.uk/TelosB_Datasheet.pdf

2.1.2 Software:

2.1.2.1 Description:

For the overall processing, comparison and evaluation of the results Matlab is used. The signal generator used for the test signal generation is also controlled from Matlab. The different devices use different software:
Imec advanced spectrum sensing: c-code controlled from a Matlab environment

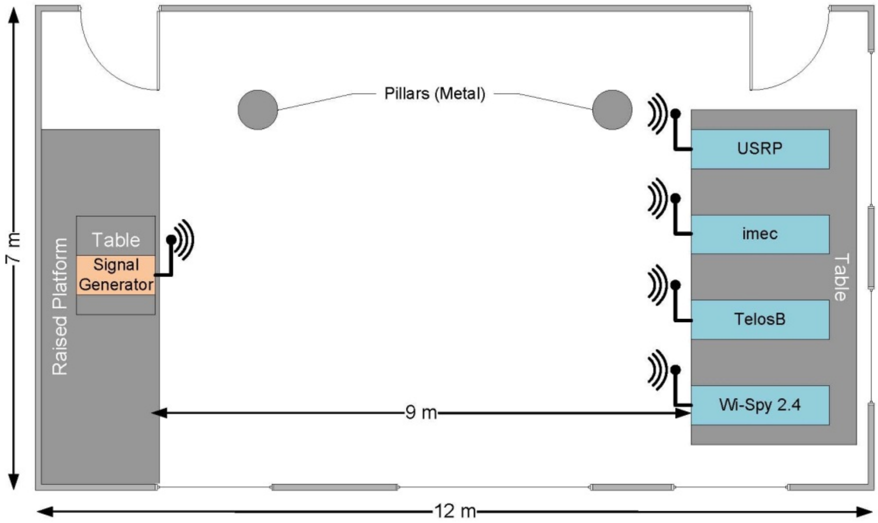
USRP1: Iris for I/Q data acquisition followed by Matlab for processing
WiSpy 2.4x: Kismet Spec-tools
AirMagnet: AirMagnet Spectrum XT TelosB: TinyOS 2 application

2.1.2.2 Code: crew.intec.ugent.be or https://svn.atlantis.ugent.be/svn/CREW/Data/Dublin-Jan2011/

2.2 Space

2.2.1 Layout:

The transmitter and sensing agents are set up in a large meeting room.



2.2.2 Mobility: *none*

2.3 Time: 2011-01-11 – 2011-01-13

2.4 RF Frequency

2.4.1 Operating range(s): *2.4 GHz ISM Band, 2400 – 2483 MHz*

2.5 Interference sources: *There was likely uncontrolled interference from various 2.4 GHz band devices.*

2.6 Trace description

2.6.1 Collected metrics (*what type of data did we collect, signal power, time, location, ...)*

- ***imec Advanced Spectrum Sensing***
  Raw IQ time domain samples sampled at 40 MSamples/s. with a 20 MHz analog signal bandwidth.
- ***USRP1 (Ettus Research)***

  Output type: IQ samples
- *Bandwidth: 8MHz**Wi-Spy 2.4x (MetaGeek, LLC.)***

  Spectrum dumps are performed as fixed bandwidth sweeps of the entire ISM 2.4 GHz band

  The resolution bandwidth is 327 KHz, sweep time is 507 ms
- ***AirMagnet Spectrum XT***

  CSV log files: 1 report/second

- ***TelosB***

  Take one RSSI sample per channel (signal power averaged over 192 us)

  Output data -> total: 2 ms per sample (sampling frequency 500 Hz)

2.6.2 Data accuracy:

- ***imec Advanced Spectrum Sensing***

  RBW / Sweep time 260 Hz / 1s

- ***USRP1 (Ettus Research)***

  Resolution BW: 7.81 kHz

  Sensing time: 5.12ms

- ***Wi-Spy 2.4x (MetaGeek, LLC.)***
  Resolution bandwidth 327 kHz

  Sweep time 300ms

- ***AirMagnet Spectrum XT***

  amplitude accuracy: +/- 2 dB

  Resolution bandwidth 156.3 kHz

  sweep time: 64 ms per 20 MHz

- ***TelosB***

  Sweep over spectrum in steps of 2 MHz (e.g. 2400->2402->2404 MHz)

  Take one RSSI sample per channel (signal power averaged over 192 us)

3.5.1 Format: Format (*detailed description of the trace format: It is left open in which format the actual measurement trace is stored as long as it is sufficiently well described and can be converted to CSV, which we consider the baseline. We also recommend other formats, such as matlab timeseries objects. The measurement units (frequency, power etc.) are described in IEEE 1900.6 in Section 6.2 on page 73 and should be referenced here)*

2.6.3 Processing tools: none

2.6.4 Signal generation:

- Source: Anritsu MG3700A RF Signal Generator
- Characteristic: DVB-T Signal
- Center Frequency: 2.477 GHz
- Bandwidth: 8 MHz
- CP Ratio: 1/4
- Power: -4 dBm

3 Experiment Trace(s)

3.1 Trace 1: https://svn.atlantis.ugent.be/svn/CREW/Data/Dublin-Jan2011/imec_scaldio2b/

3.1.1 Description: imec sensing agent traces

3.1.2 Time *2011-01-12 11h10m09s – 18h27m07s*

3.1.3 Trace-specific meta-information *none*

3.1.4 Trace-file: IMEC_1.1_onoff.tgz, IMEC_1.1_signal.tgz, IMEC_1.1_silent.tgz, IMEC_1.2_onoff.tgz, IMEC_1.2_signal.tgz, IMEC_1.3_onoff.tgz, IMEC_1.3_signal.tgz, IMEC_1.4_signal.tgz, IMEC_1.5_signal.tgz, IMEC_2.1_signal-15dBm.tgz, IMEC_2.2_signal-15dBm.tgz, IMEC_2.3_signal-15dBm.tgz, IMEC_3.1_signal-30dBm.tgz, IMEC_3.2_signal-30dBm.tgz, IMEC_3.3_signal-30dBm.tgz, IMEC_4.1_signal.tgz, IMEC_4.2_signal.tgz, IMEC_4.3_signal.tgz, IMEC_5.1_signal.tgz, IMEC_5.2_signal.tgz, IMEC_6.1_signal.tgz, IMEC_7.1_signal.tgz

3.2 Trace 2: https://svn.atlantis.ugent.be/svn/CREW/Data/Dublin-Jan2011/Airmagnet%20Spectrum%20XT/Description: imec sensing agent traces

3.2.1 Time *2011-01-12 11h10m09s – 18h27m07s*

3.2.2 Trace-specific meta-information *none*

3.2.3 Trace-file: CREW_1_1_onoff_2011-01-12_14-49-12.csv, CREW_1_1_signal_2011-01-12_12-02-05.csv, CREW_1_1_silent_2011-01-12_11-15-36.csv, CREW_1_2_onoff_2011-01-12_15-08-13.csv, CREW_1_2_signal_2011-01-12_12-24-18.csv, CREW_1_2_silent_2011-01-12_15-24-37.csv, CREW_1_3_onoff_2011-01-12_15-17-20.csv, CREW_1_3_signal_2011-01-12_12-36-20.csv, CREW_1_4_signal_2011-01-12_15-59-40.csv, CREW_1_5_signal_2011-01-12_16-05-45.csv, CREW_2_1_signal_2011-01-12_16-22-55.csv, CREW_2_2_signal_2011-01-12_16-30-26.csv, CREW_2_3_signal_2011-01-12_16-38-03.csv, CREW_3_1_signal_2011-01-12_16-43-29.csv, CREW_3_2_signal_2011-01-12_16-51-01.csv, CREW_3_3_signal_2011-01-12_16-58-16.csv, CREW_4_1_signal_2011-01-12_17-11-50.csv, CREW_4_2_signal_2011-01-12_17-15-05.csv, CREW_4_3_signal_2011-01-12_17-19-28.csv, CREW_5_1_signal_2011-01-12_17-23-05.csv, CREW_5_2_signal_2011-01-12_17-28-49.csv, CREW_5_3_signal_2011-01-12_17-31-53.csv, CREW_6_1_signal_2011-01-12_17-42-05.csv, CREW_7_1_signal_2011-01-12_18-20-12.csv

3.3 Trace 3: https://svn.atlantis.ugent.be/svn/CREW/Data/Dublin-Jan2011/IrisReadings/I_and_Q_data/

3.3.1 Time *2011-01-12 11h10m09s – 18h27m07s*

3.3.2 Trace-specific meta-information *none*

3.3.3 Trace-file: measurement1.2onoff, measurement1.2silent, measurement2.1onoff, measurement2.3signal, measurement3.3signal, measurement4.2signal, measurement4.3signal, measurement5.2signal, measurement5.3signal, measurement6.1signal

3.4 Trace 4: https://svn.atlantis.ugent.be/svn/CREW/Data/Dublin-Jan2011/IrisReadings/PSD_estimates/

3.4.1 Time *2011-01-12 11h10m09s – 18h27m07s*

3.4.2 Trace-specific meta-information *none*

3.4.3 Trace-file: measurement1.2onoff, measurement1.2silent, measurement2.1onoff, measurement2.1signal, measurement2.2signal, measurement2.3signal, measurement3.3signal, measurement4.2signal, measurement4.3signal, measurement5.2signal, measurement5.3signal, measurement6.1signal

3.5 Trace 5: https://svn.atlantis.ugent.be/svn/CREW/Data/Dublin-Jan2011/telos/

3.5.1 Time *2011-01-12 11h10m09s – 18h27m07s*

3.5.2 Trace-specific meta-information *none*

3.5.3 Trace-file: CREW_Measurement_1.1_signa.txt, CREW_Measurement_1.1_silent.txt,

CREW_Measurement_1.2_OnOff_1channe.txt,
CREW_Measurement_1.2_OnOff_5channels.txt,
CREW_Measurement_1.2_Silent_1channe.txt,
CREW_Measurement_1.2_Silent_5channels.txt, CREW_Measurement_1.2_signa.txt,
CREW_Measurement_1.3_OnOff_1channe.txt,
CREW_Measurement_1.3_OnOff_5channels.txt, CREW_Measurement_1.3_signa.txt,
CREW_Measurement_1.4_Signal_1channe.txt,
CREW_Measurement_1.4_Signal_5channels.txt, CREW_Measurement_1.4_signa.txt,
CREW_Measurement_1.5_Signal_1channe.txt,
CREW_Measurement_1.5_Signal_5channels.txt,
CREW_Measurement_2.1_Signal_1channe.txt,
CREW_Measurement_2.1_Signal_5channels.txt,
CREW_Measurement_2.2_Signal_1channe.txt,
CREW_Measurement_2.2_Signal_5channels.txt,
CREW_Measurement_2.3_Signal_1channe.txt,
CREW_Measurement_2.3_Signal_5channels.txt,
CREW_Measurement_3.1_Signal_1channe.txt,
CREW_Measurement_3.1_Signal_5channels.txt,
CREW_Measurement_3.2_Signal_1channe.txt,
CREW_Measurement_3.2_Signal_5channels.txt,
CREW_Measurement_3.3_Signal_1channe.txt,
CREW_Measurement_3.3_Signal_5channels.txt,
CREW_Measurement_4.1_Signal_1channe.txt,
CREW_Measurement_4.1_Signal_5channels.txt,
CREW_Measurement_4.2_Signal_1channe.txt,
CREW_Measurement_4.2_Signal_5channels.txt,
CREW_Measurement_4.3_Signal_1channe.txt,
CREW_Measurement_4.3_Signal_5channels.txt,
CREW_Measurement_5.1_Signal_1channe.txt,
CREW_Measurement_5.1_Signal_5channels.txt,
CREW_Measurement_5.2_Signal_1channe.txt,
CREW_Measurement_5.2_Signal_5channels.txt,
CREW_Measurement_5.3_Signal_1channe.txt,
CREW_Measurement_5.3_Signal_5channels.txt,
CREW_Measurement_6.1_Signal_1channe.txt,
CREW_Measurement_6.1_Signal_5channels.txt,
CREW_Measurement_7.1_Signal_16channels.txt,
CREW_Measurement_7.1_Signal_1channe.txt,

3.6 Trace 6: https://svn.atlantis.ugent.be/svn/CREW/Data/Dublin-Jan2011/wispy/

3.6.1 Time *2011-01-12 11h10m09s – 18h27m07s*

3.6.2 Trace-specific meta-information *none*

3.6.3 Trace-file: measurement1.1signal.txt, measurement1.1silent.txt,

measurement1.2onoff.txt, measurement1.2signal.txt, measurement1.2silent.txt, measurement1.3onoff.txt, measurement1.3signal.txt, measurement1.4signal.txt, measurement1.5signal.txt, measurement2.1signal.txt, measurement2.2signal.txt, measurement2.3signal.txt, measurement3.1signal.txt, measurement3.2signal.txt, measurement3.3signal.txt, measurement4.1signal.txt, measurement4.2signal.txt, measurement4.3signal.txt, measurement5.1signal.txt, measurement5.2signal.txt, measurement5.3signal.txt, measurement6.1signal.txt, measurement7.1signal.txt,

4 Common portal

At multiple points in this deliverable, the CREW portal is referenced. The CREW portal is a public website, containing all information and external links needed for experimenters to be able to understand the functionality of the CREW platform. The portal can be reached via the public CREW website, located at www.crew-project.eu.

The goal of the portal is twofold:

1. Present experimenters with a high-level overview of the available infrastructures, so one or multiple cognitive components (i.e. wireless testbeds, sensing engines, software etc.) of use for a specific experiment may be identified.
2. Present all details needed to start experiments using the cognitive components of choice.

The CREW portal is updated, whenever CREW components (hardware and/or software) are added or modified. For an up-to-date version of the portal, the reader is referred to www.crew-project.eu/portal. A snapshot of the portal as of 30/09/2011 is included in Appendix A: CREW Portal of this document.

5 Testbeds components and combinations

5.1 Mix and match components approach, the "virtual components"

As introduced in chapter 3 of D2.2, the combination of components from different testbeds is one of the modes of operation of the CREW federation. This mode and the general "virtual components" approach is one of the fundamental added values of the Federation: providing new capabilities with the existing features and capabilities. New functionalities are provided with as little as possible integration effort. This combination of components requires the definition of interfaces. Depending on the type of components combined and the targeted functionality different sort of interfaces are possible, ranging from simple usage conventions to elaborated software application programming interfaces (APIs).

The following subsections will detail the interfaces identified and used within CREW together with the components and their combinations.

5.2 Component interfaces

5.2.1 Transceiver Facility API[1]

5.2.1.1 Concept and approach

D2.2 section 3.3 introduced the concept and the rationale behind the Transceiver facility [5]. As stated there, the goal is to provide a solution to the problem of the multiplicity of interface specifications for radio transceivers programming, command and control. Depending on the industry (military, commercial, public safety) or the market segment (e.g. base station or mobile device manufacturer) very different specifications are available. These specifications may apply at different levels of the protocol stack, involve one or more hardware devices or address a completely different set of parameters and properties of the radio equipment. [6], [7], [8] are examples of these specifications.

The transceiver facility tries to provide a generic and standardized solution from an SDR perspective. Figure 21 in D2.2 highlights the fundamental SDR principle of "waveform" and "platform" separation (also called the "waveform" and "platform" paradigm).

The term "waveform" refers to the radio application. A "waveform" could also be seen as the radio standard e.g. WCDMA, GSM, WiMAX, TETRA or the radio protocol stack (physical layer, media access control and any other applicative layer). The term is thus encompassing all the software (either general computing or specific signal processing code) but eventually also dedicated hardware specially designed for a radio access technology or communication standard (e.g. high performance ASIC for FFT or encryption engine).

The term "platform" is on the other hand referring to any hardware component not specifically designed for the radio standard but used by it.

The "waveform" and "platform" separation principle appears quite straightforward for general computing tasks, for example those performed by the MAC. The "waveform" is the software and the "platform" a GPP on which the MAC software executes. The paradigm tends to be less clear if we look at the physical signal processing layer, where most of the computing will be performed by a DSP but potentially, for performance and consumption requirements, specific integrated hardware modules will also take in charge some of the radio protocol necessary tasks. In that case the "waveform" will be the signal processing software running on the DSP but also the IP of the integrated circuit and the "platform" will only be the DSP.

[1] Most of this chapter summarizes contents of the Transceiver Facility Specification official document. However, owing to the valuable feedback provided by the USRP2 implementation carried out within CREW modifications are possible. Those are mainly complements to the existing content i.e. new API services. Slight deviations from the official standard are also possible. In that cases the description presented here replaces and supersedes the official document contents.

The frontier blurs further when we look at the radio transceiver, typically composed by a digital and an analogue part. The radio transceiver was traditionally and it is still for most commercial applications (consumer products) highly dependent of the radio access technology, typically depending on its bandwidth, band and data rates. Nevertheless the SDR technology, by moving more and more hardware functionalities to the software realm, is enabling the design of generic transceiver with high bandwidth and large frequency range RF front-ends. These transceivers are seen as "platforms" with a given set of features and performances through which a group of "waveform" may access the radio channel.

The Transceiver facility sets the frontier and defines a set of interfaces between the two sides. The next figure depicts the level at which the Transceiver facility interface sits.

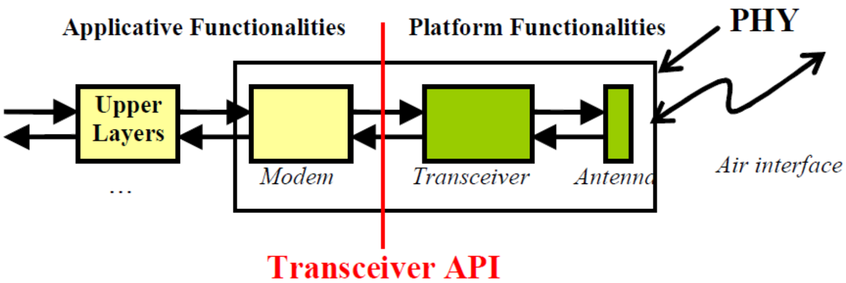


Figure 16: Transceiver interface sits between the "waveform" and the "platform"

5.2.1.2 Transceiver functionality

This section is intended to provide a summary of the main functionalities that any implementation of the Transceiver should provide. It also aims at identifying the key concepts being the foundations for the interfaces presented in the next section.

The transceiver is seen as depicted in figure Figure 17, it takes a baseband signal and transforms it in a radio signal that can access the air through the antenna. Therefore the transceiver is defined as the processing stage that transposes, for transmission, a *baseband signal* into a *radio signal*, and, for reception, a *radio signal* into a *baseband signal*. This processing stage is necessary to implement a given radio access technology. The Transceiver is not part of the waveform but of the platform.

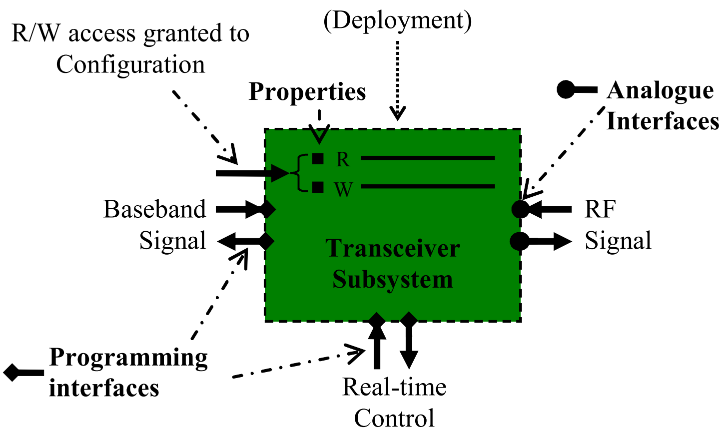


Figure 17: Transceiver conceptual view and external interfaces

The transceiver has a set of properties, a number of interfaces with the modem (or physical layer part of the "waveform") and other platform devices through programming interfaces (for data exchange and real-time control). It can also be configured. This will depend on the capabilities of the implementation (ability to support several frequency ranges, sampling ratios, bandwidths).

The two key functionalities of the Transceiver are the transmit and receive channels[2]. In the case of simplex waveforms (i.e. Transmit-only or Receive-only), only the corresponding type of channel is used, and a Transceiver providing only this functionality would match the requirements of such a waveform. In case of duplex waveforms both types of channels shall be provided.

5.2.1.3 Key concepts

5.2.1.3.1 Up-Conversion and Down-Conversion

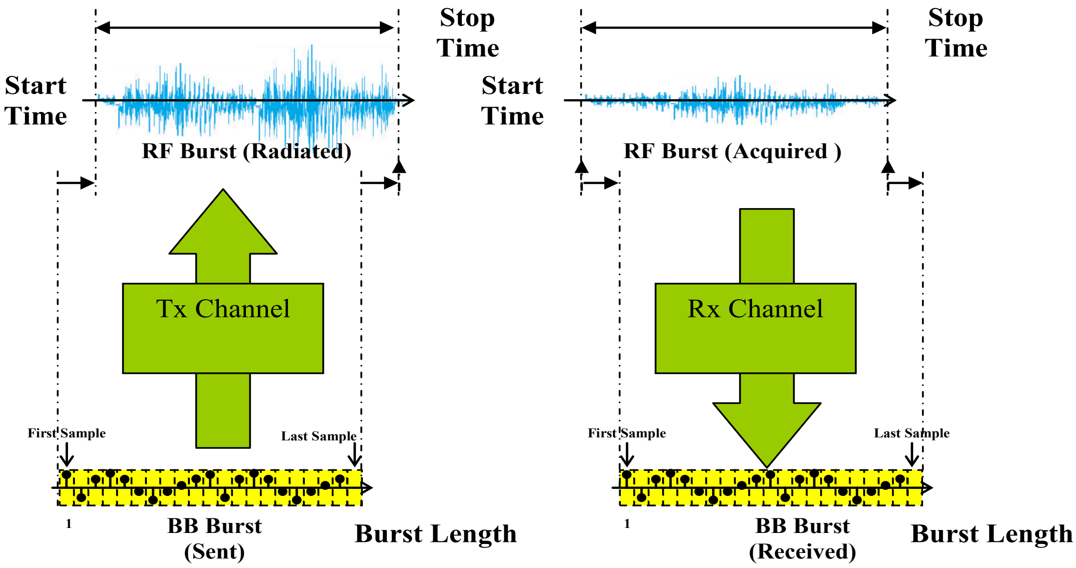
For a transmission a Transmit channel will perform an *Up-conversion* of a baseband signal to a radio signal while a Receive channel will perform a *Down-conversion* of a radio signal to a baseband signal The baseband signal is an analytical signal (sampled complex I&Q data) sampled at a sampling frequency (specific to the considered waveform).

Basically the *Up-conversion* carries out the whole operation of generating a RF signal modulated with or carrying the information contained in the baseband signal. Common processing steps for such an operation are: sampling rate adaptation, D/A conversion, generation of phase/quadrature signal implementing Hilbert transform, transposition from base-band to carrier frequency, channel filtering and transmission power control.

Likewise *Down-conversion* consists of extracting the baseband information contained in the radio signal to an analytic signal. This typically involves: transposition from carrier frequency to base-band, channel filtering, A/D conversion, analytic filtering generating analytic signal from the real signal, sampling rate adaptation and automatic gain control.

5.2.1.3.2 Burst

The notion of *Burst* is fundamental for the whole Transceiver approach. The Transceiver operation is seen a as the transmission and reception of RF signal in bursts of a given duration (defined by start and stop times) with an attached set of radio properties (carrier frequency, bandwidth). In Transmission the Transceiver is taking the samples of a baseband analytical signal or baseband burst at its input and up-converting them. In reception it is down-converting the RF signal burst at the antenna into a set of samples composing the baseband burst. The Transceiver is therefore working on a *burst basis.*



[2] It is worth highlighting that the notions of *Tx Channel* and *Rx Channels* have **nothing to do** with the radio propagation *channel*.

Figure 18: Transceiver works on a *Burst* basis

The Transceiver API provides services for creating bursts by specifying a *Time Profile* and a *Tuning Profile*, both composing the *Burst Profile*. The waveform uses these services to create the bursts according to its needs.

- Time Profile: It is defined by a *Start Time* and a *Stop Time.*
- Tuning profile: *Carrier frequency*, *transmission power* and *receive packet size* are direct tuning parameters. Additionally, the API services offer an additional parameter called *Preset*. The number and type of configuration settings included in the *Preset* are Transceiver implementation dependent. Most of the Tuning Parameters may not be directly specified at run-time. The *Preset* parameter is intended for those Transceiver parameters that will not be changed by the waveform on a real-time basis. Baseband sampling frequency, channel bandwidth, co-channel rejection, signal dynamic, in-band ripple are typical examples of such parameters. Each specific modulation type used by a given waveform will request an adapted set of *Preset* tuning parameters. For many waveforms a single set of *Preset* tuning parameters is enough.

The invocation of all burst control operations has to be sufficiently anticipated to allow the Transceiver to react. Several burst profiles may be provided in advance.

Some waveforms may require "on-the-fly" modifications of the *Bursts* (indeed the *Burst Profile*) after those are already running (Transceiver is in transmitting or receiving state). The typical example is related to the burst duration that can be unknown at *Burst* creation (push-to-talk radio, synchronization procedure). For these scenarios dedicated services in the Transceiver API are available enabling the update of some of the parameters of an on-going *Burst*.

5.2.1.3.3 Baseband signal exchange

The baseband signal is exchanged between the Transceiver Channel and the waveform modem using packets of baseband samples. A baseband *Burst* is thus decomposed into one or many baseband *Packets*. These packets are transferred between the waveform and the Transceiver by means of dedicated services in the programming APIs. The size of the *Packet* is defined at *Burst* creation for the Receive channel. For transmission the *Packet* size may vary from one *Packet* to another given the independent invocation of the data transferring or pushing API services.

The Baseband FIFO is a dedicated storage space within the Transceiver implementation where the samples are temporarily stored. It acts as a buffer for transmission and reception of the samples composing the baseband Bursts.

The next pictures (extracted from the official specification) provide with a complete view of the key Transceiver functionalities introduced so far.

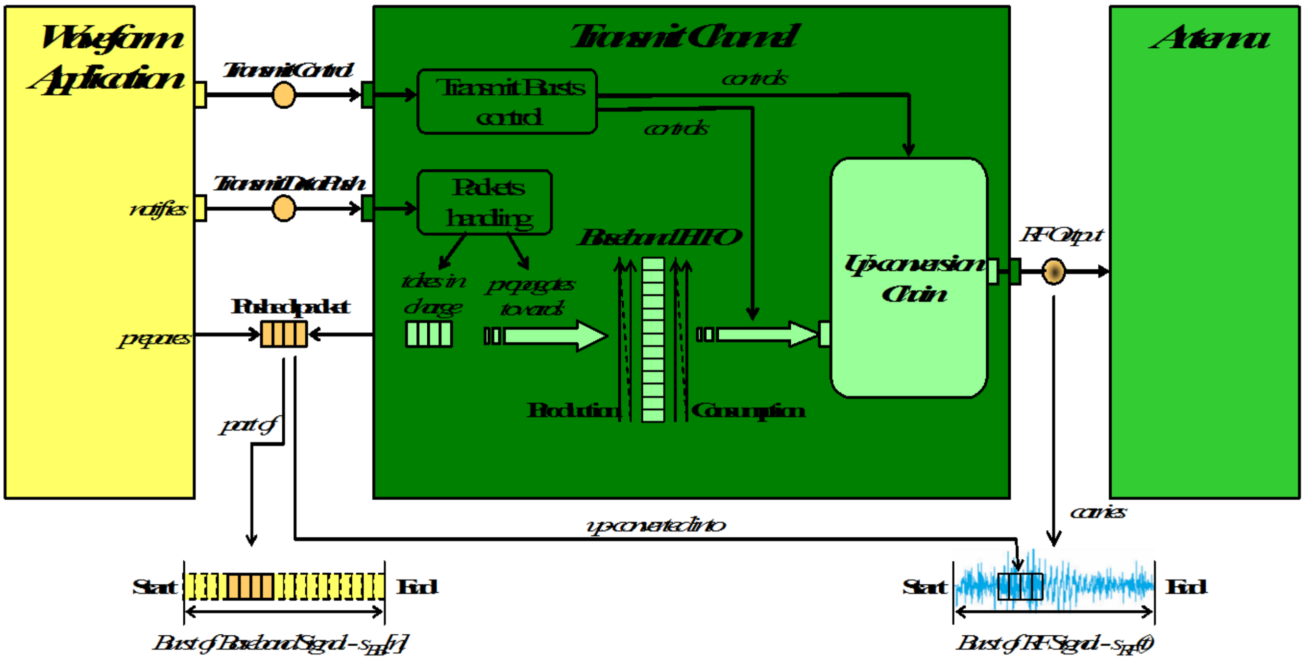


Figure 19: Transceiver functionalities of a *Transmit channel*

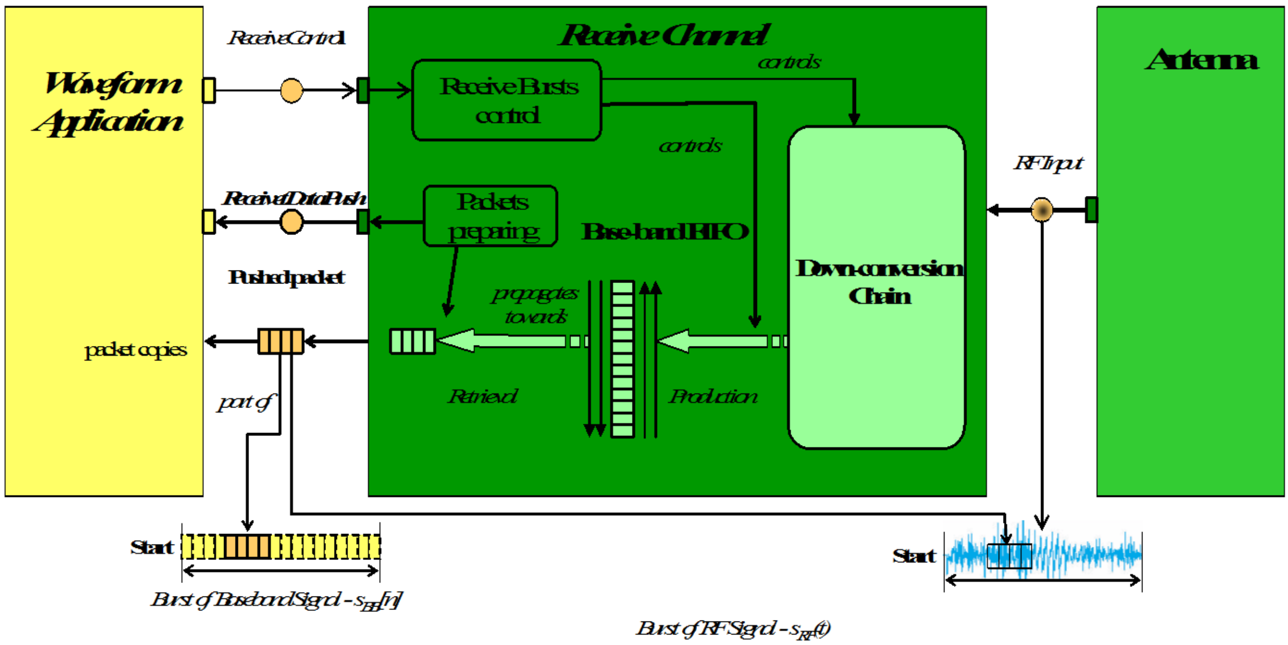


Figure 20: Transceiver functionalities of a *Receive channel*

5.2.1.3.4 Time management mechanisms

The way the *StartTime* and the *StopTime* of the *Time Profile* (belonging to the *Burst Profile*) are specified is a critical concept in the Transceiver. Several approaches are available for providing the transceiver with such information. The choice of the approach will be depending on the waveform characteristics and the Transceiver platform capabilities. Those approaches called *Time modes* are explained here.

Undefined Time mode

It is only applicable for the *StopTime*. In some scenarios the *StopTime* of a given *Burst* may not be known at *Burst* creation time. It will be provided afterwards (e.g. after a synchronization has been achieved). For these cases the *StopTime* parameter adopts the *Undefined* value, meaning that the transmission or reception will last until further notice.

Immediate time mode

For some waveforms *Burst* are created in an asynchronous basis. Others require a radio access to be performed as soon as possible without further timing details or accuracy. In those cases the *Immediate* time mode is preferred for the *StartTime* and *StopTime*. Examples could be, the first reception of a handset following a switch on to start a ranging procedure, or the half-duplex requirements of a push-to-talk radio where there is no protocol requesting a well-defined slotted frame.

Absolute time mode

This mode makes the assumption that both sides of the interface i.e. waveform modem and transceiver understand the same time base (they will typically share a common reference clock). *StartTime* and *StopTime* are parameters of the Time profile conveying an absolute time such a date, hour or second at which the Transceiver should start/stop transmission/reception of *Bursts*.

Event based time mode

This is the more flexible (and complex) time mode. The shared time references between waveform modem and Transceiver are limited to a set of well-defined *Events* and their occurrences. Upon the occurrence of one of these events the waveform knows the state of the Transceiver and may issue *Burst* creation requests based on that *Event*.

The *Events* defined in the Specification document are the TransmitStartTime, TransmitStopTime, ReceiveStartTime and ReceiveStopTime. These *Events* happen each time the Transceiver starts/stops a transmission/reception. The way the waveform is aware of the *Event* occurrence, for example that a transmission started, is known by means of a dedicated service of the programming API.

Some waveform may even not need to know the event occurrence and issue their *Burst* creation requests in a sequential basis, based up on the fact that each *Burst Time Profile* refers to the previous radio activation.

The Event based time mode is composed of four identifiers in the specification:

- Event Source Id: The reference *Event* (TransmitStartTime, TransmitStopTime, ReceiveStartTime or ReceiveStopTime)
- Event Count Origin: The *Event* occurrence for starting counting events.
- Event Count: An integer number signaling the number of *Events* having to occur before starting any action[3].
- Time Shift: An amount of time to wait before performing any action after the *Event* occurrence.

5.2.1.4 Interfaces

The following sections describe the APIs enabling the control/command message exchange and data transfer between the waveform and the transceiver platform. Some examples are provided illustrating use cases of the interface.

[3] Event Count Origin and Event Count parameters are defined in the official Specification. Nevertheless, Transceiver proof-of-concept implementations have shown that unnecessary complexity stems from their usage. Complete Time Profile information may be provided without using them.

The description of the methods (a.k.a. services) provided here is by no means exhaustive. The errors cases each method may have to deal with are not described. More information may be found within the official document. Moreover, the handling of the error cases is implementation dependent and may vary from one transceiver to another.

5.2.1.4.1 Interface methods

The number of available methods on the API is quite reduced following the goal of simplicity. Up to 8 methods are thus proposed: four for the receive channel and four more for the transmit channel. The usage of all the methods for one waveform is not mandatory. Indeed the usage is depending on the waveform requirements.

Transmit methods

`createTransmitCycleProfile`	**Transmit Burst creation**	

Method	`createTransmitCycleProfile(`
	`Time requestedTransmitStartTime,`
	`Time requestedTransmitStopTime,`
	`UShort requestedPresetId,`
	`Frequency requestedCarrierFrequency,`
	`AnaloguePower requestedNominalRFPower)`	

Parameters	`requestedTransmitStartTime`	Burst Start Time
	`requestedTransmitStopTime`	Burst Stop Time
	`requestedPresetId`	Tuning configuration present
	`requestedCarrierFrequency`	Carrier frequency
	`requestedNominalRFPower`	RF power

This method is used to request the transceiver to configure in order to start a TX burst transmission. The Time Profile of the Burst is defined by the Time parameters and the Tuning Profile by the Preset, Carrier Frequency and RF power. The Transceiver will perform the necessary configuration steps to set-up the Tuning Profile. Afterwards, and once the system time machine reaches the Start Time of the Time Profile, the transceiver will start transmitting a Burst with whatever data it is available on its internal FIFO buffer.

`configureTransmitCycle`	**Enables re-configuration of previously requested transmit Burst**	

Method	`configureTransmitCycle(`
	`Ulong targetCycleId,`
	`Time requestedTransmitStartTime,`
	`Time requestedTransmitStopTime,`
	`Frequency requestedCarrierFrequency,`
	`AnaloguePower requestedNominalRFPower)`	

Parameters	`targetCycleId`	Identifier for the previously requested Burst
	`requestedTransmitStartTime`	New Burst Start Time
	`requestedTransmitStopTime`	New Burst Stop Time
	`requestedCarrierFrequency`	New Carrier frequency
	`requestedNominalRFPower`	New RF power

This method is used to modify the Time Profile and some of the parameters of the Tuning Profile when a createTransmitCycleProfile() command has been previously issued.

`setTransmitStopTime`	**Enables the setting of the Stop Time for an on-going Transmit Burst**

Method	`setTransmitStopTime(`
	`Ulong targetCycleId,`
	`Time requestedTransmitStopTime)`

Parameters	`targetCycleId`	Identifier for the previously requested Burst
	`requestedTransmitStopTime`	On-going Burst Stop Time

This method is dedicated to the setting of the Stop Time of a Burst that is already on-going and that had an undefined Stop Time. The methods allow for stopping Transmission Burst with and unknown duration at the creation invocation time.

`pushBBSamplesTx`	**Enables the data exchange from the waveform to the Transceiver**	

Method	`pushBBSamplesTx(`
	`BBPacket thePushedPacket,`
	`Boolean endOfBurst)`	

Parameters	`thePushedPacket`	I&Q data packet
	`endOfBurst`	End of Burst signalling boolean

This method is designed for carrying data between waveform and Transceiver. The endOfBurst flag lets the Transceiver know the last packet of a set of packets composing a single Burst.

Receive methods

`createReceiveCycleProfile`	**Receive Burst creation**	

Method	`createReceiveCycleProfile (`
	`Time requestedReceiveStartTime,`
	`Time requestedReceiveStopTime,`
	`Ulong requestedPacketSize,`
	`UShort requestedPresetId,`
	`Frequency requestedCarrierFrequency)`	

Parameters	`requestedTransmitStartTime`	Burst Start Time
	`requestedTransmitStopTime`	Burst Stop Time
	`requestedPacketSize`	Packet size for transferring incoming data
	`requestedPresetId`	Tuning configuration present
	`requestedCarrierFrequency`	Carrier frequency

This method is used to request the transceiver to configure in order to start a RX burst reception. The Time Profile of the Burst is defined by the Time parameters and the Tuning Profile by the Preset and Carrier Frequency. The Transceiver will perform the necessary configuration steps to set-up the Tuning Profile. Afterwards, and once the system time machine reaches the Start Time of the Time Profile, the transceiver will start receiving a Burst. The I&Q data will be stored within its internal FIFO buffer. The PacketSize parameter sets the size of the packets that will be exchanged between the Transceiver and the waveform.

`configureReceiveCycle`	**Enables re-configuration of previously requested receive Burst**	

Method	`configureReceiveCycle (`
	`Ulong targetCycleId,`
	`Time requestedReceiveStartTime,`
	`Time requestedReceiveStopTime,`
	`Ulong RequestedPacketSize,`
	`Frequency requestedCarrierFrequency)`	

Parameters	`targetCycleId`	Identifier for the previously requested Burst
	`requestedReceiveStartTime`	New Burst Start Time
	`requestedReceiveStopTime`	New Burst Stop Time
	`RequestedPacketSize`	New Packet Size
	`requestedCarrierFrequency`	New Carrier frequency

This method is used to modify the Time Profile and some of the parameters of the Tuning Profile when a

createReceiveCycleProfile() command has been previously issued.

`setReceiveStopTime`	**Enables the setting of the Stop Time for an on-going Receive Burst**	

Method	`setReceiveStopTime (`
	`Ulong targetCycleId,`
	`Time requestedReceiveStopTime)`	

Parameters	`targetCycleId`	Identifier for the previously requested Burst
	`requestedReceiveStopTime`	On-going Burst Stop Time

This method is dedicated to the setting of the Stop Time of a Burst that is already on-going and that had an undefined Stop Time. The methods allow for stopping Reception Burst with and unknown duration at the creation invocation time.

`pushBBSamplesRx`	**Enables the data exchange from the Transceiver to the waveform**	

Method	`pushBBSamplesRx(`
	`BBPacket thePushedPacket,`
	`Boolean endOfBurst);`	

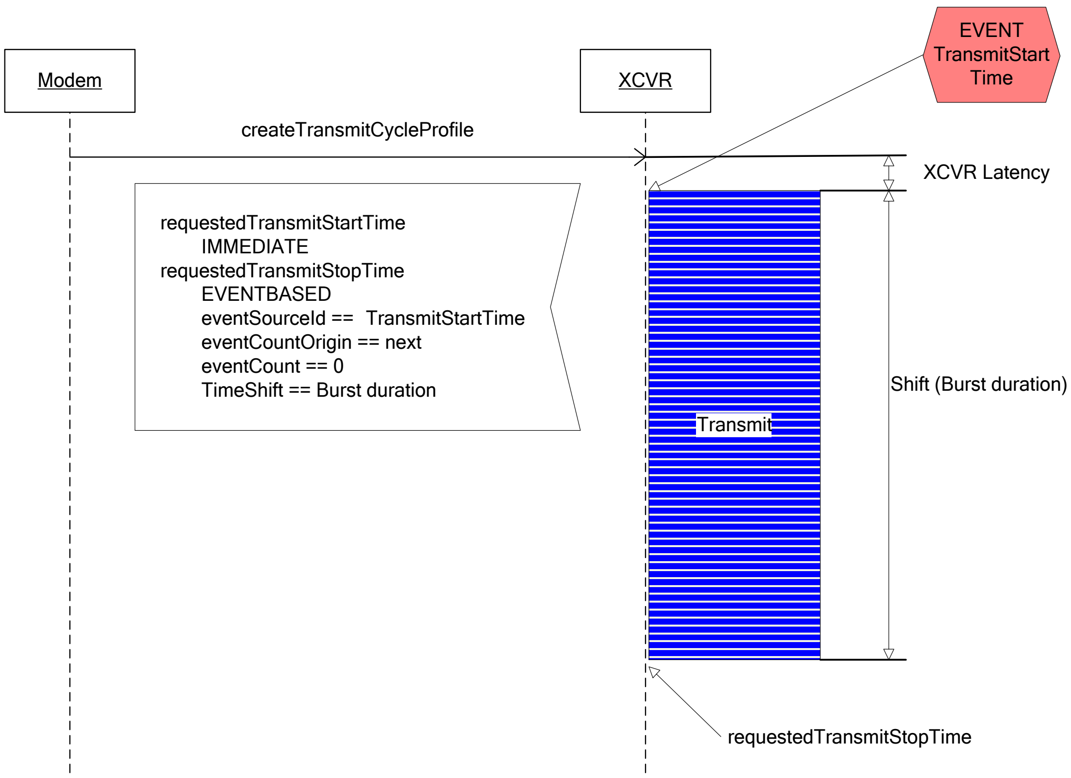
Parameters	`thePushedPacket`	I&Q data packet
	`endOfBurst`	End of Burst signalling boolean

This method is designed for carrying data between Transceiver and waveform. The endOfBurst flag lets the waveform know the last packet of a set of packets composing a single Burst.

5.2.1.4.2 Example use cases for Bursts Time Profile configuration

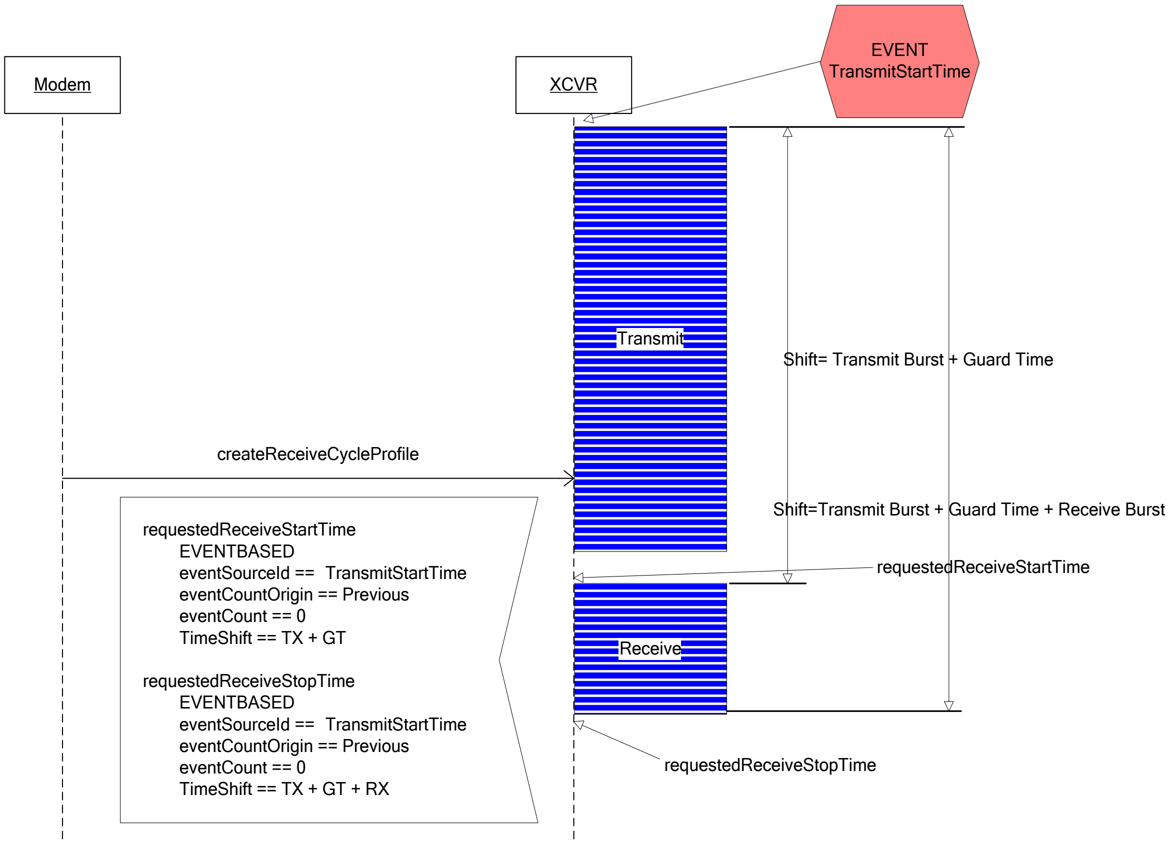
For reference a few uses cases for a TDD waveform are presented here. These use cases are basically depicting the usage and combination of the different Time Profile modes.

Setting the Transceiver for a Transmit Burst, starting immediately for a well-defined duration



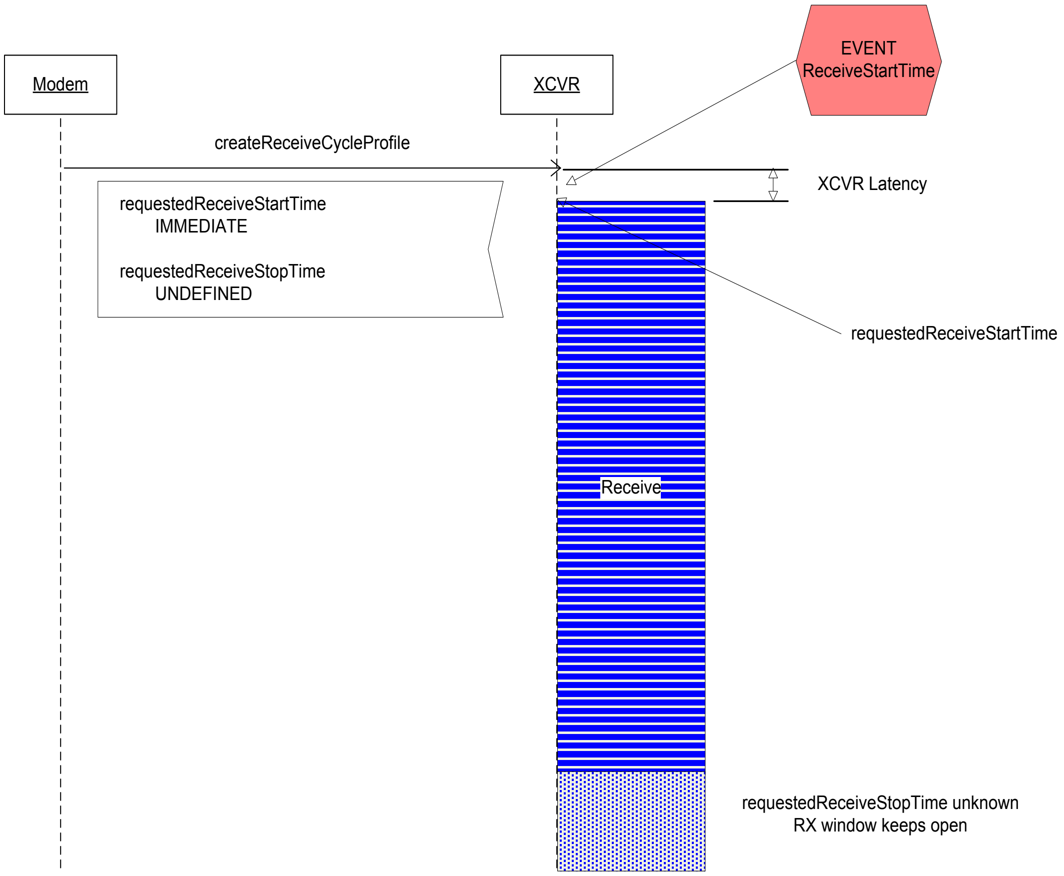
The above diagram depicts how the **`requestedTransmitStartTime`** parameter is set to *Immediate Time* mode in order to request the Transceiver to open a transmit window for a Burst *as soon as possible*. The assumption is made that the exact timing or instant of the time base at which the transmission starts has no relevance. The exact duration of the Burst, on the other hand, has to be well defined. Thus the **`requestedTransmitStopTime`** is set in *Event Based Time* mode using the *TransmitStartTime* as reference event source and a **`TimeShift`** equal to the targeted Burst duration. **`eventCountOrigin`** is set to **`next`**, meaning that the next occurrence of the selected event source will be the reference event. **`eventCount`** is set to zero.

Setting the Transceiver for Receive Burst with accurate timing regarding the previous Transmit Burst



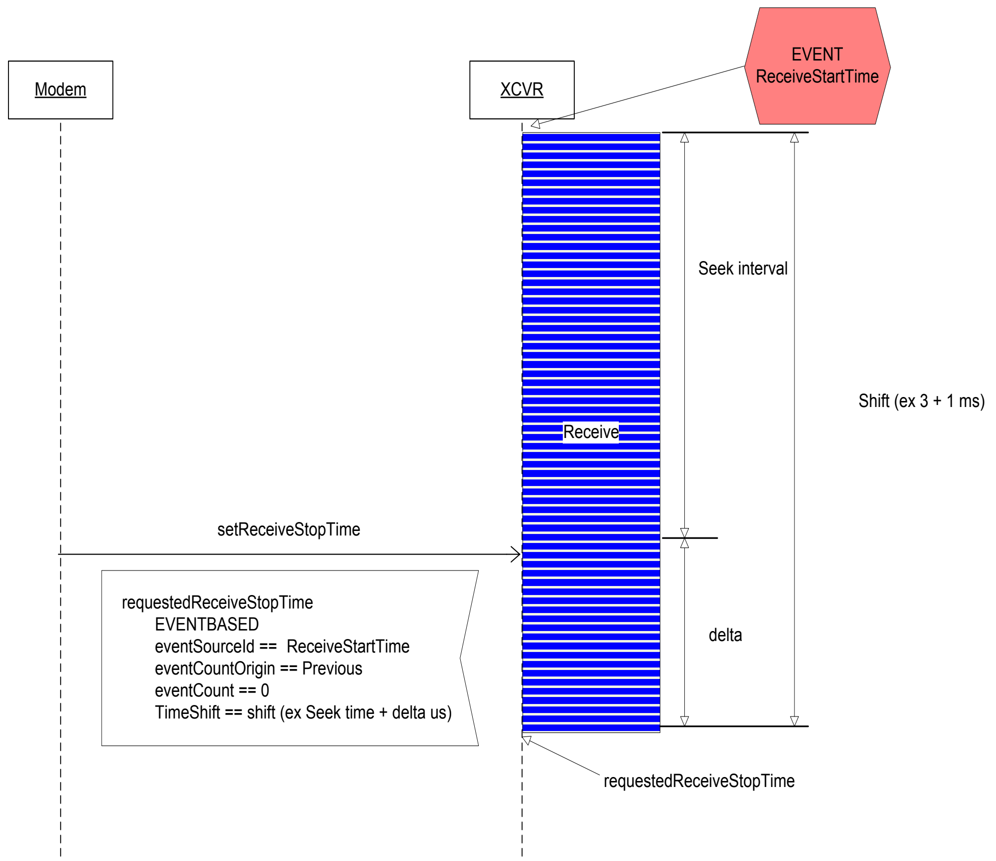
In this diagram the previous Transmit Burst is used in order to get the reference events enabling accurate timing. Therefore both the **`requestedReceiveStartTime`** and **`requestedReceiveStopTime`** parameters are set in *Event Based Time* mode, with an **`eventSourceId`** equal to the TransmitStartTime. Previous TransmitStartTime event is well known by the Transceiver since it controlled the starting time of the previous Burst. **`eventCountOrigin`** is set to **`previous`** to reference the previous event occurrence. **`TimeShift`** indicates the shifting in time regarding the event occurrence, hence the **`TimeShift`** for the StartTime is set to the previous Transmit Burst duration plus the guard time and the StopTime **`TimeShift`** to the same amount plus the Receive Burst duration itself.

Undefined duration Receive Burst



This use case could be a typical use case for a synchronization acquisition procedure. The **`requestedReceiveStartTime`** is set to *Immediate Time* mode to start the searching procedure at a given time. The waveform does not know "a priori" the amount of time getting synchronization could take so the **`requestedReceiveStopTime`** is set to *Undefined Time* mode. The waveform will decide later on (next diagram) when to stop the Receive Burst.

Stopping the synchronization procedure



In the diagram it is assumed that for some reason the synchronization procedure has to stop at a well-defined time (for example to keeping track of frame slotted structure). Under this assumption the waveform has to take advantage of the only event known by the Transceiver sub-system i.e. the ReceiveStartTime. The waveform will use the `requestedReceiveStopTime` and complete the parameter data with a `TimeShift` encompassing the searching time plus some margin to avoid requesting for an StopTime that has already been reached (late request).

5.2.2 IMEC interfaces

Two implementations of the imec sensing engine are developed. The base platform for both prototypes is the SPIDER board: a PCB containing a USB interface to connect to the host, an FPGA to connect the different components on the board to each other and a DIFFS chip. Via an SAMTEC connector and second board can be plugged on containing the RF section, including ADC/DAC operation. Two implementations will be available:

- A board containing an RF section using the imec SCALDIO IC. This IC contains a full RF transceiver including analog to digital conversion. A picture of this setup is shown in figure Figure 21.
- A combination of two boards: the WARP board, containing and RF IC and ADC/DAC blocks and a second board which serves mainly as interconnect between connectors on the SPIDER and WARP board.

Since both RF ICs have different capabilities the programming options will differ between the two solutions but the same API is used to control the settings.

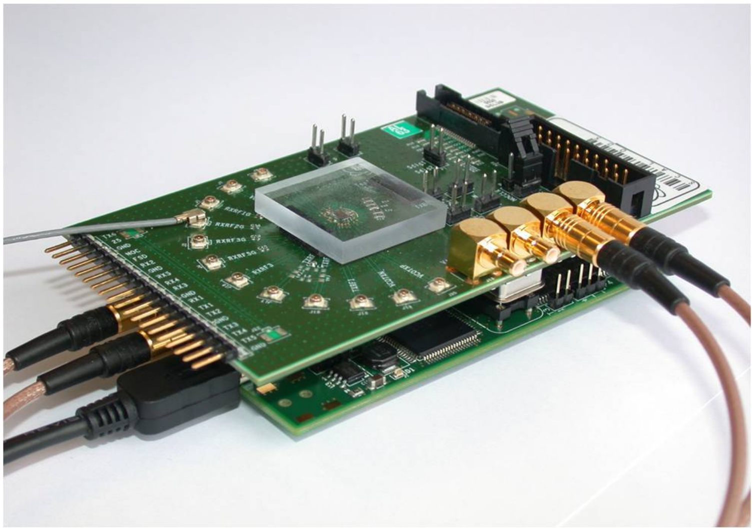


Figure 21: SPIDER and SCALDIO sensing prototype

The hardware abstraction layer is implemented in ANSI C and runs on the host PC. This HAL hides the USB driver from the user and provides functions that are used in the API to implement high level function calls. How it connects to an IBBT w-iLab.t node is shown in Figure 22,t he block labelled HW is the actual sensing engine.

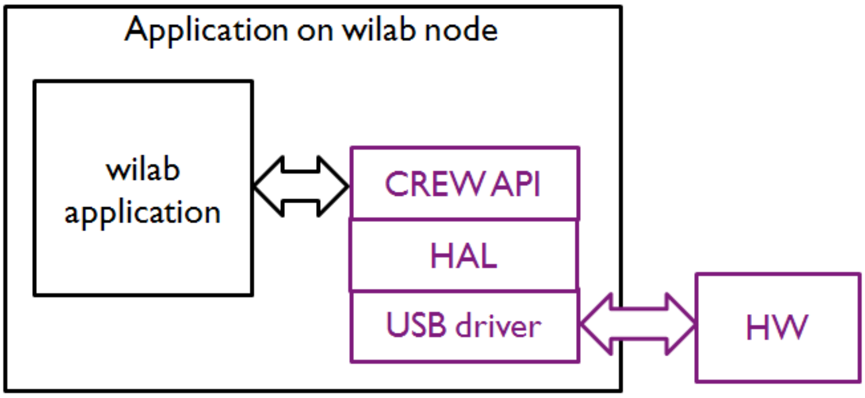


Figure 22: sensing engine connected to wilab node

An overview of the HAL functions is provided below:

Table 4: Sensing engine HAL functions

Function prototype	Description
id = se_open	Function to open the sensing engine hardware. A handler to identify the sensing engine is returned.
se_config(id, RF struct, DIFFS struct)	Function to configure the sensing engine. This function requires an identifier for the hardware, a struct containing all RF settings and a struct containing all DIFFS settings.
se_start_measurement(id,pointer_to_result)	This function will start the actual with the configuration as set by se_config. The function requires an identifier to the device to start and a pointer to where the measurement result needs to be stored.
se_stop_measurement(id)	This function will stop an acitve measurement on

	a device with identifier id.
se_close(id)	Function to close the connection to the device identified by id

Two structs are used to configure the complete sensing engine. The parameters for the RF struct are listed in Table 5.

Table 5: RF struct parameters

RF struct	
BW	Baseband filter settings
GAIN	Gain value for the complete RX chain
RF	RF frequency setting

The blocks where these parameters have impact can be clearly seen on the block diagrams of the SCALDIO chip below: BW (left), Gain (middle) and RF frequency (right)

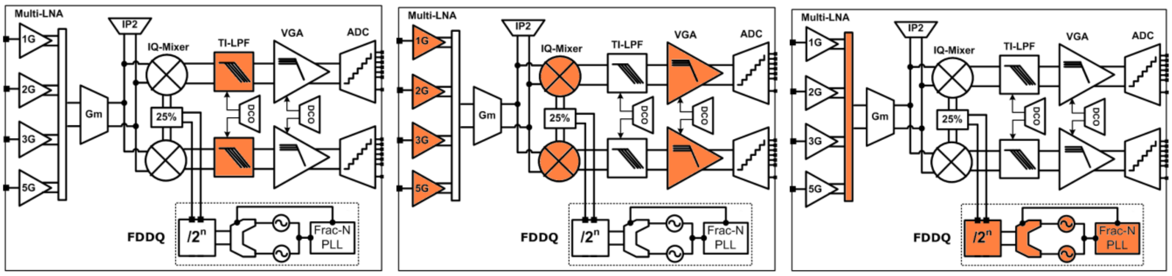


Figure 23: SCALDIO chip block diagrams

The parameters on the DIFFS struct are listed in Table 6.

Table 6: DIFFS struct parameters

DIFFS struct	
AGRAC_FW	Pointer to the location of the firmware for the Automatic Gain and Resource Allocation Controller.
SENSEPRO_FW	Pointer to the location of the firmware for the SENSPRO processor on the DIFFS chip
DATAPATH	Struct containing all parameters to set up the datapath leading to the SENSEPRO processor correctly.

The blocks to which the different parameters apply can be identified in the block diagrams of the DIFFS chip below: AGRAC_FW (left), SENSEPRO_FW (middle) and DATAPATH (right)

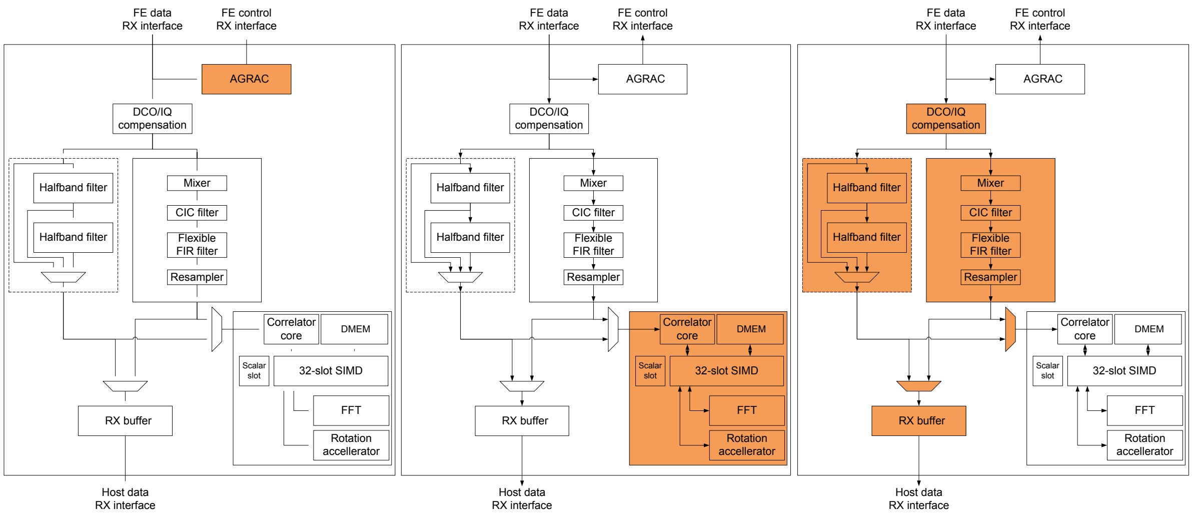


Figure 24: DIFFs chip block diagram

To enable the user to correctly program the sensing engine without knowing the specifics of the hardware components a high level user API is available. An overview of the high level API function calls is provided in Table 7.

Table 7: user API description

Function prototype	**Description**
id = crew_open	This functions enable the connection to the sensing engine and returns an identifier which is to be used in the following function calls in order to address the correct sensing engine
crew_channel(id,mode,channel,detector,pointer_to_results)	This function handles the complete configuration of the sensing engine. 5 Parameters are used to describe the configuration • id: identifier to the address the correct sensing engine • mode: selection of the correct wireless standard • channel: select the desired channel • detector: selection of the wanted detection algorithm to run on the DIFFS • pointer_to_result: pointer to where the result of the sensing operation should be stored
crew_ism_sweep(id,detector,pointer_to_results)	This function implements a sweep of the complete ISM band. The user can select the detection algorithm via the detector parameter
crew_dvb_sweep(id,detector,pointer_to_results)	This function is only available on when the sensing engine is equipped with a

	SCALDIO RF section. A sweep across all channels in the DVB band is carried out and the result is stored in the location specified in pointer_to_results. The user can select the detection algorithm via the detector parameter.
crew_close(id)	This function closes the connection to the sensing engine and cleans up the environment.

Depending on the configuration of the sensing engine, either with a SCALDIO RF section or with a WARP RF section, different settings for the mode, channel and detector variables are possible. An overview is provided below.

Available parameters via the user API in case a WARP RF section is used:

- Detector
  - Algorithm
    - Power measurement (vs. threshold)
    - FFT (vs threshold) - [32/64/128] bins
  - Gain configuration
    - Automatic Gain Control
    - Fixed gain
- Mode/channels
  - Bluetooth / 1 to 80
  - Zigbee / 1 to 15
  - WLANg / 1 to 13
  - WLANn / 1 to X

Since the SCALDIO RF section supports a wider range of frequencies more modes are available in case this RF section is used. Additionally some detection algorithms related to the wireless standards operational in these frequencies are also available. An overview is provided below:

- Detector
  - Algorithm
    - Power measurement (vs. threshold)
    - FFT (vs. threshold) - [32/64/128] bins
    - LTE multiband energy detection
    - Cyclostationary [2k-8k] / Guard interval [1/4-1/8-1/16-1/32]
  - Gain configuration
    - Automatic Gain Control
    - Fixed gain
- Mode/channels
  - Bluetooth / 1 to 80
  - Zigbee / 1 to 15

- WLANg / 1 to 13
- WLANa / 1 to X
- WLANn / 1 to X
- LTE / channels to be determined
- DVB-T / 1 to 69

This selection of algorithms, modes and channels should provide enough flexibility for the common user, if not the user can resort to the HAL to gain more in depth access to the hardware.

5.3 Combined virtual components description

5.3.1 LTE detector simulation environment

5.3.1.1 Simulation environment

In order to validate the multi-antenna LTE detection algorithm described in section 0, a spatial LTE simulation environment was developed as described in Figure 25.

Firstly, an LTE signal database was built using TUD "LTE testbed reference signal generator". Signals with transmission characteristics (bandwidth, physical layer cell identity, CP length, scrambling, etc.) were generated and stored into files. These files will allow testing the algorithms in various LTE modes. These signals are perfect (no noise, no propagation channel, no interference, no frequency offset) and their goal is to check the good understanding of the LTE standard and to validate the algorithms in perfect environment.

Secondly, the LTE signal files generated during the first step are used as an input to the multi-antenna propagation channel simulator. This simulator, described in section 5.3.1.2 can simulate the signal received by a UE from several LTE BTS (or from various RATs emitters), taking into account the spatial locations of the BTS, the multipath of the signals from each BTS, the reception noise, the relative received power from each BTS, etc. The goal is to validate the detection algorithm in a more realistic but known environment. The advantages of the multi-antenna approach will be proved.

It is important to underline that the first signal (the useful signal to be detected) is always an LTE signal, but depending on the simulated scenario, the other signals can be either LTE signals or a non-LTE signal (a secondary user in CR-oriented use case).

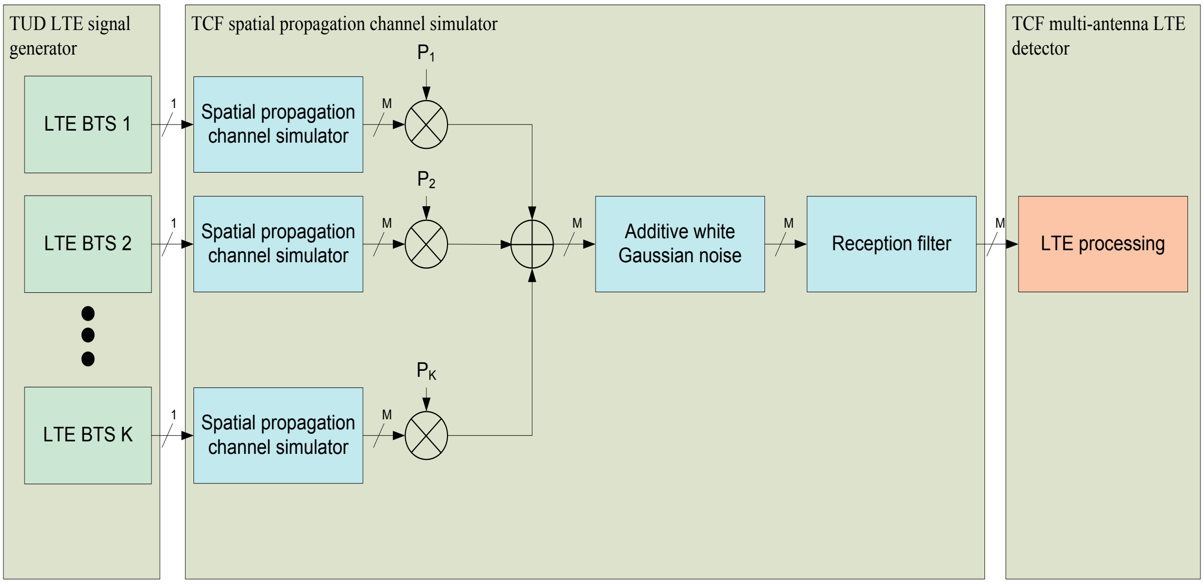


Figure 25: Spatial LTE simulation environment

5.3.1.2 Spatial propagation channel model

In the context of mobile communications, it is well established that the distortions induced by the propagation channel between the transmitter and the receiver deeply influence the performance of the demodulation algorithms. Moreover as the algorithms proposed in section 0 are using an antenna array and antenna processing, it is thus crucial to be able to reliably simulate the propagation channels between the active emitters and the M receivers. The multi-channel propagation model used for the algorithmic study is an extension of the classical Clarke mono-channel model [1].

The Clarke model allows taking into account the situation of a moving device at speed v. The antenna array receives different plane waves due to multiple reflections on various obstacles (near or far obstacles):

- With random amplitudes,
- With random phases,
- With random directions of arrival,

At the propagation channel output, the received signal in base-band is given by the following expression:

$$\underline{\mathrm{x}}(t) = \sum_{l=1}^{L} a_l d(t-\tau_l) \left[\sum_{n=1}^{N_l} c_{n,l} e^{j(2\pi v/\lambda \cos(\theta_{n,l}-\gamma)t+\varphi_{n,l})} \underline{\mathrm{s}}_{n,l} \right] \quad \text{Eq. 5-27}$$

where:

- $d(t)$ is the useful modulated signal,
- $\underline{\mathrm{x}}(t)$ is the received signal vector,
- L is the number of paths,
- a_l is the attenuation of the l^{th} path (a path is due to a reflection on a far obstacle),
- τ_l is the delay of the l^{th} path,
- N_l is the number of elementary sub-paths associated to the l^{th} path. The sub-paths are due to the multiple reflections typically all around the device. All sub-paths associated to a path are considered to have the same delay (the delay differences are negligible compared to the inverse of the signal bandwidth),
- $c_{n,l}$ is the attenuation of the n^{th} sub-path associated to the l^{th} path,
- v is the device speed,
- γ is the angle between the device speed and the North (randomly uniformly distributed $[0, 2\pi]$),
- λ is the wavelength,
- $\theta_{n,p}$ is the azimuth of the n^{th} sub-path associated to the l^{th} path, (see Figure 26 for the orientation conventions),
- $\varphi_{n,1}$ is the phase of the n^{th} sub-path associated to the l^{th} path (randomly uniformly distributed $[0, 2\pi]$),

- $\underline{s}_{n,l}$ is the steering vector of the n[th] sub-path associated to the l^{th} path. The steering vector depends on the array geometry and on the direction of arrival of the sub-path:
- $\underline{s}_{n,l} = \left[e^{j\Delta\phi_{n,l,1}} \cdots e^{j\Delta\phi_{n,l,M}}\right]^{\mathrm{T}}$,

  where:

  - M is the number of antennas
  - $\Delta\phi_{n,l,k}$ is the geometric phase shift of the n sub-path associated to the l^{th} path between the k^{th} antenna and the array centre:
  - $\Delta\phi_{n,l,k} = 2\pi(R_k/\lambda)\cos(\theta_{n,l} - \alpha_k)$,

    where:

    - (R_k, α_k) are the polar co-ordinates of antenna k.

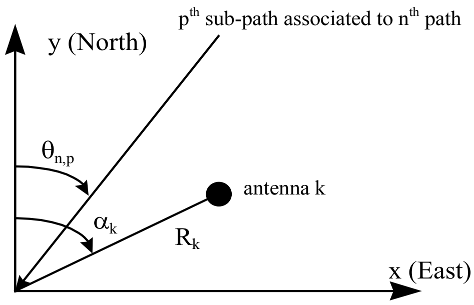


Figure 26: Orientation convention for the measure of angles

In this model, all the parameters are stationary. The only variations are due to the device speed, which generates a fading variation of the received signal on each antenna. When the device is not moving, there is spatial diversity (each antenna has its own fading level), but this spatial diversity is stationary.

In short, our propagation model can be used to simulate various scenarios, including:

- Stationary propagation conditions: N_l = 1. The user chooses all the parameters for the simulation. This simple scenario is essentially used in order to validate the algorithms.
- Rayleigh fading: all the paths have the same number of sub-paths (N_l = 10 for all the simulations) with the same amplitude: $c_{n,l}$ = 1/sqrt(N_l).
- Rice fading: the first sub-path has a higher amplitude than the other paths ($c_{0,l}$ >$c_{n,l}$ = 1/sqrt(N_l). The relative amplitude of the different sub-paths is an input parameter.

Finally, the various angles of arrival $\theta_{n,l}$ are chosen randomly and uniformly distributed in an angular reception cone.

5.3.2 Combining the imec spectrum sensing agent and the IBBT w-iLab.t

5.3.2.1 Motivation

Wireless testbeds come in different sizes and flavours. While some testbeds focus on physical layer phenomena, other testbeds focus on higher layers of the OSI stack and are used to design and evaluate protocols for wireless networks, ranging from MAC to application layer.

The testbeds of the latter category, typically offer the possibility to collect basic packet-level statistics, such as the number of packets sent or received over the wireless interface, the number of packets that are received but cannot be decoded due to a CRC error, or the number of packets that failed to be sent. While in these testbeds, the physical layer might not be the topic of primary interest, the behaviour of the physical layer and state of the wireless medium is a crucial factor influencing the outcome of the experiments. For example, in the event that the wireless medium in the environment of the testbed is heavily loaded, high packet loss rates will usually result in protocols performing worse than in an environment without any RF interference in the frequency range of interest.

When developing cognitive networking protocols, accurate characterization of the RF spectrum becomes more important. Advanced spectrum sensing solutions are then important, both (1) to provide input to the cognitive protocols – indeed, cognitive protocols cannot decide on how to adapt to the current wireless environment if they are not capable of sampling the RF spectrum; and (2) to accurately monitor the RF spectrum range of interest before, during, and after the experiment, thus enabling the testbed to report on the “quality” of newly developed protocols and provide information on the spectral efficiency.

By integrating the imec spectrum sensing agent in the IBBT w-iLab.t, the two above goals were achieved during the first year of the CREW project.

5.3.2.2 Implementation and possibilities

Figure 27 gives an overview of how the integration between the imec spectrum sensing agent and the w-iLab.t is realized.

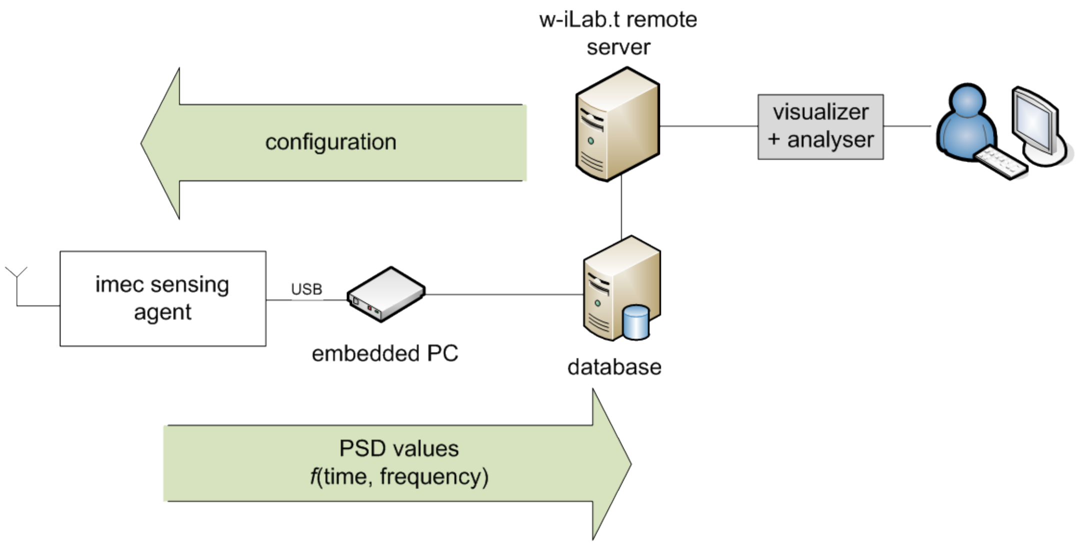


Figure 27: schematic overview of the integration

The sensing agent is currently preconfigured to continuously scan the 2.4 *GHz* ISM spectrum (an implementation for the 5 *GHz* would be possible as well). The sensing agent produces power spectral density (PSD) values at a high rate (up-to-date figures are available the portal). Per 20MHz bandwidth, the sensing agent performs an FFT with 128 bins. Thus, the sensing agent provides a single PSD value for every bin of 156250 *Hz*. In the current set-up, a 120 *MHz* band is scanned (2.39 *GHz* – 2.51 *GHz*), resulting in 768 bins.

In order to limit the amount of information sent to the embedded PC, the sensing agent pre-processes measurements in real-time: per frequency bin, the maximal PSD out of a window of n consecutive PSD measurements per frequency bin is determined, with n=3 in the current implementation The choice to use a the maximum value rather than an average value, was made because we are currently primarily interested in detecting whether or not a specific part of the frequency spectrum was in use during a certain timeslot.

The result is that the embedded PC grabs the 768 bins ca. 60 times per second. To reduce the amount of raw data by a factor 20, a max hold filter was implemented in software on the embedded PC. The resulting bins are transmitted over a UDP socket about 3 times per second to the database (mysql-server) and are inserted by a bash script into circular buffer table which can store the samples for one day.

These measurements are then stored in the database of the testbed using following format:

Table 8: database format

created [mySql timestamp, added when received in the database]	***createdus*** [actual timestamp, added on the embedded PC]	***seID*** [unique identification of the spectrum engine reporting the values]	***sensingEngineData*** [1 PSD value per frequency bin]

As can be seen from the Table 8 above, a unique identifier is attributed to each of the sensing engines. This makes it possible to support multiple spectrum sensing engines in the network, and to link the PSD values to a specific device and thus location in the network. Moreover, a timestamp is added to the PSD values as soon as they are measured *(createdus).* As this timestamp is synchronized with the other logging mechanisms operational in the testbed, each PSD value can be linked to a specific moment in time, e.g. the transmission of a packet. As such, a spatio-temporal view of the frequency occupation in the testbed becomes possible, thus complementing the packet-level information that was already available before this integration was completed.

Once these values are stored in the database, the w-iLab.t uses its generic analyser and visualizer tools to present the spectrum information directly to the user in real-time. For performance reasons, the visualizer updates at a slower rate than the values are stored in the database. However, note that the more fine-grained information is still available in the database, and may be used in this form by the experimenter.

5.3.2.3 Additional possibilities and future work

The integration approach, as followed above, can also be used on top of other testbeds, with minimal adjustments. Moreover, the tools that were and will be developed to visualize and process the spectral measurements may be used in other test facilities as well.

In the case of the w-iLab.t testbed, due to regulatory limitations, wireless nodes are only allowed to transmit in the 2.4 *GHz (*and 5 *GHz)* ISM band. Therefore, the configuration of the sensing engine in our integration is set fixed, scanning only the 2.4 *GHz* ISM band. Moreover, in the current implementation, configuration parameters such as the number of reports per second are also set to a fixed value. These settings should not be considered as limitations: they are purely a matter of configuration. Additional configuration settings will be available in the future.

Furthermore, a set of metrics is to be automatically determined during experiments, based on the spectral measurements obtained from the sensing engines.

6 Conclusion

This document gave a detailed description of the CREW federated cognitive radio testbed in its first basic operational stage.

It began with a short overview of the available equipment in the different wireless communication testbeds. The IRIS SDR testbed of TCD is a highly reconfigurable software radio platform with software based on C++ and XML, and hardware based on GPPs. The IRIS software radio platform is complemented by a physical layer testbed consisting of USRPs acting as RF frontends, a vector signal generator and a spectrum analyzer. The whole testbed can be controlled remotely via the internet using an SSH connection. The TWIST testbed at TUB is a multi-platform hierarchical sensor network testbed comprising 102 eyesIFX and 102 Tmode Sky wireless sensor nodes. Additionally, multiple low-cost USB spectrum analyzers and multiple shimmer2r sensor nodes for mobile BAN test cases are provided to experimenters. Also the TWIST testbed can be controlled remotely via the internet. The imec sensing platform enables users to perform spectrum sensing experiments by either reprogramming the hardware or accessing captured I/Q samples. The imec spectrum sensing platform can be integrated into other systems or testbeds using its USB interface, allowing access to all measurement data as well as to configure and to control the device. An API and a HAL are provided to integrate the sensing device with a host computer. Either already existing sensing algorithms can be uploaded and used on the device, or customized algorithms developed by the experimenters can be used. The IBBT w-iLab.t testbed consists of 200 wireless sensor nodes installed in an office environment, as well as 60 nodes located in a shielded environment without external interference. All nodes operate in the 2.4 GHz and the 5 GHz ISM bands. 10 imec sensing nodes and 8 USRPs complement this equipment for cognitive radio experiments. The hardware installation comes along with a wide range of tools and software for experimentation and a benchmarking framework to create reproducible wireless environments and to assess the experimentation results. All devices are accessible via the internet. Users can modify the testbed with custom firmware, software drivers and protocols. The multi-antenna LTE detection algorithm offered to experimenters is an advanced method to detect even weak base stations in the LTE spectrum based on the detection of the primary and secondary synchronization signals.

The available testbed equipment can be used in different ways. Mixing and matching different hard- and software components creates “virtual components” with new functionalities. A prerequisite is the precise definition and description of the component interfaces. An example for such an interface is the Transceiver Facility API that specifies the interface between the radio transceiver and the signal processing unit of an SDR system. It is based on the concept of bursts of baseband samples that are transmitted between signal processing unit and transceiver. Also for the imec sensing agent a HAL interface has been specified and described that makes the devices accessible via USB. The LTE multi-antenna sensing approach can be accessed and tested using a defined spatial LTE simulation environment, consisting of an LTE signal database, a special propagation channel simulator and the signal processing of the detector.

With this set of information, this document can be seen as a reference for the usage of the CREW testbed by internal and external experimenters regarding the capabilities and the usability of the available equipment.

7 References

[1] R.H. Clarke, “A statistical theory of radiomobile reception”, Bell Syst. Tech. J. 47, pp. 957-1000 (1968)

[2] L.E. Brennan, I.S. Reed, “An adaptive array signal processing algorithm for communications”, IEEE Trans. Aerospace and Electronics Systems., vol. AES 18, N°1, January 1982.

[3] Federal Communications Commission, “Second memorandum opinion and order,” FCC 10-174, Sept. 2010

[4] "IEEE Standard for Spectrum Sensing Interfaces and Data Structures for Dynamic Spectrum Access and other Advanced Radio Communication Systems.," IEEE Std 1900.6-2011 , vol., no., pp.1-168, April 22 2011 doi: 10.1109/IEEESTD.2011.5756728

[5] E. Nicollet “Transceiver Facility Specification” SDRF-08-S-0008-V1.0.0, January 28, 2009.

[6] http://www.cpri.info/spec.html

[7] http://www.obsai.org/

[8] http://www.mipi.org/specifications/digrfsm-specifications

8 Appendix A: CREW Portal

Please note that the Portal is optimized for viewing in a browser. In the snapshot below, some of the pictures are resized to better fit the A4 page format of the deliverable. For an up-to-date and interactive version of the portal, please consult the public portal at www.crew-project.eu/portal.

Figure 28 - screenshot of the CREW portal welcome page

Home
[Edit] [Export] [Clone]

Filter on frequency range: <Any>
Filter on OSI layer: <Any>
Filter on radio type: <Any>
Apply

Testbed or cognitive component	Short description
IRIS	Iris is a software radio architecture that has been developed by the CTVR , built in C++, it is used for constructing complex radio structures and highly reconfigurable radio networks. Its primary research application is to conduct a wide range of dynamic spectrum access and cognitive radio experiments. It is a GPP-based radio architecture and uses XML documents to describe the radio structure. This test bed will be highly beneficial when conducting cognitive radio experiments as it allows us to simulate a wide range of networks as well as giving us a highly flexible architecture to manipulate based on intelligent observations made about its surroundings. Each radio is constructed from fundamental building blocks called components. Each component makes up a single process or calculation that is to be carried out by the radio. For instance, a component might perform the modulation on the signal or scale it by a certain amount. Each component supports one or more data type and passes dataset to other components along with some metadata such as a time stamp and sample rate. There is a data buffer between each component to ensure the data is safe, even if one component is processing data much faster than another **Available Frequency bands:** ISM 2.4GHz, ISM 5GHz, LTE bands, TV bands **OSI layers to experiment with:** PHY **Radio interface(s) available:** flexible radio
LTE/LTE advanced testbed	Dresden's LTE/LTE+ like testbed was set up in 2008 as part of the Easy-C project (www.easy-c.com). The signal processing hardware includes Sorbas602 eNodeB Simulators, Sorbas202 Test UEs and Sorbas472 Radio Units, all supplied by Signalion (www.signalion.com). The eNBs and UEs are connected through IF interconnects with the radio unit frontend, which supports up to two Tx and two Rx channels for MIMO capability. The testbed operates in EUTRAN band VII (DL @ 2670-2690 MHz / UL @ 2550-2570 MHz) with fixed bandwidth of 20MHz and in FDD mode. For the CREW project, two experimentation setups are available. The indoor lab features 5 eNBs and 4 UEs. While the hardware is stationary itself, the Tx and Rx antennas can be positioned anywhere in the lab room. Further, 4 additional UEs are mounted on studio racks / carts and can be moved within the building. The approximate transmit power is 15 dBm. The outdoor lab consists of two base station sectors that are fixed on two opposing corners of the faculty building (Figure 4), approx. 150 m apart. In addition to the mobile indoor UEs from setup 1, 3 rickshaw UEs are available for outdoor experiments in the vicinity of the building. The transmit power is approximately 30 dBm. **Available Frequency bands:** LTE band VII **OSI layers to experiment with:** PHY **Radio interface(s) available:** LTE
Sensing Engine	The imec sensing enigne is a prototype implementation of a sensing engine for mobile communication devices. The prototype consists of two main blocks: an analog RF front-end including analog to digital conversion and a DIgital Front-end For Sensing (DIFFS). For permanent deployment in the w-iLab.t testbed a WARP board is selected as analog RF front-end, covering the 2.4 and 5 GHz ISM bands. An in house developed flexibele SCAlable raDIO (SCALDIO) is also tested in a lab environment, covering an RF input range from 0.1 up to 6 GHz and a channel bandwidth up to 40 MHz. The digital front-end is an ASIP specifically designed for sensing operations, signal conditioning and synchronization. The chip contains a flexible filter block, including re-sampling and a SIMD core, extended with multiple accelerator cores. Multiple sensing algorithms are implemented on the system, ranging from straight-forward energy detection schemes, over more complex feature detection (for e.g. DVB-T) to multiband energy detection after FFT leakage removal for OFDMA LTE signals. **Available Frequency bands:** ISM 2.4GHz, ISM 5GHz, LTE bands, TV bands **OSI layers to experiment with:** PHY **Radio interface(s) available:** flexible radio

Figure 29 - Screenshot of the sortable "short overview" tables (1/2)

TWIST	The TKN Wireless Indoor Sensor Network Testbed (TWIST) is a multi-platform, hierarchical sensornetwork testbed architecture developed at the Technische Universität Berlin. One instance is currently deployed at TUB campus: a total of 204 sensor nodes (102 eyesIFX and 102 Tmote Sky nodes) are distributed in a 3D grid spanning 3 floors of an office building, resulting in more than 1500 m² of instrumented office space. Two nodes of each platform are deployed, while the larger ones (~28 m²) have four nodes. This setup results in a fairly regular grid deployment pattern with intra node distance of 3m. Within the rooms the sensor nodes are attached to the ceiling. The TWIST architecture introduces a layer of "super-nodes" (previous figure, right) between the sensor nodes and the testbed server, which manages sensor node reprogramming, configuration or accessing debug information over the serial connection. TWIST relies on COTS hardware and fully leverages the features of the USB 2.0 standard. The sensor nodes are connected to the super-nodes via USB hubs, which act as concentrators and also provide a power supply management capability. This enables active topology control and node fault injection modelling through selective powering on and off of nodes. TWIST is currently being extended by mobile robots which can be used for experiments that involve controlled mobility. At the end of CREW Year 1 (at the time of the first open call) one mobile robot can be used for local experiments. **Available Frequency bands:** ISM 2.4GHz **OSI layers to experiment with:** Application, Routing / transport, MAC **Radio interface(s) available:** IEEE802.11 a, IEEE802.11 bg, Zigbee
w-iLab.t	The w-iLab.t allows flexible testing of the functionality and performance of wireless networking protocols and systems in a time-effective way, by providing hardware and the means to install and configure firmware and software on (a selection of) nodes, schedule automated experiments, and collect, visualize and process results. Thanks to an in-house designed hardware control device, unique features of the testbed include the triggering of repeatable digital or analog I/O events at the sensor nodes, real-time monitoring of the power consumption, and battery capacity emulation. At a first location, the "w-iLab.t Office" consists of a wireless Wi-Fi (IEEE 802.11a/b/g) and sensor network (IEEE 802.15.4) testbed infrastructure, deployed across three 90 m x 18 m floors of the IBBT office building in Ghent, Belgium. At 200 places throughout the office spaces, meeting rooms and corridors, wireless hardware is mounted to the ceiling. In Zwijnaarde, Belgium, located approximately 5 km away from the "w-iLab.t Office", a second location is equipped with another 60 wireless nodes nodes, with IEEE 802.11a/b/g/n, IEEE802.15.4 and IEEE802.15.1 (Bluetooth) interfaces. The location also hosts software defined radio platforms (USRP) and spectrum scanning engines developed by imec. **Available Frequency bands:** ISM 2.4GHz, ISM 5GHz **OSI layers to experiment with:** Application, Routing / transport, MAC, PHY **Radio interface(s) available:** Bluetooth, flexible radio, IEEE802.11 a, IEEE802.11 bg, IEEE802.11 n, Zigbee

Figure 30 - Screenshot of the sortable "short overview" tables (2/2)

Note to the reader of this deliverable: the following pages are generated from the printer-friendly version of the "advanced information section" on the CREW portal. Please note that in some cases, this web document contains links that refer to external websites and documents for more details. They have not been included in this deliverable but are accessible online.

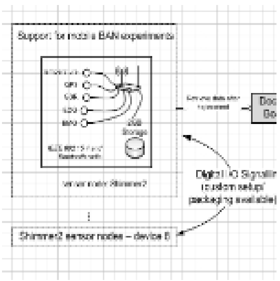
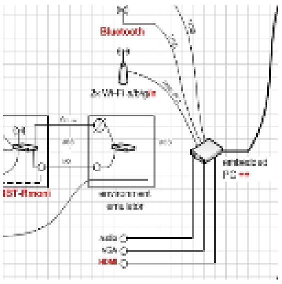
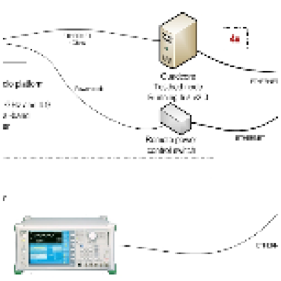
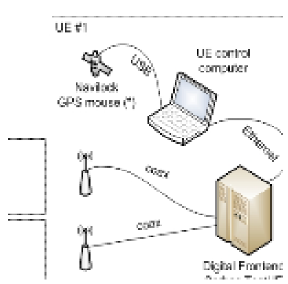
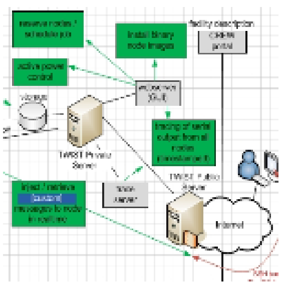
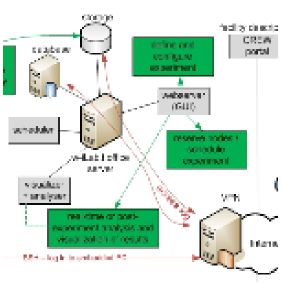
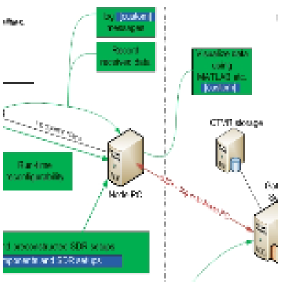
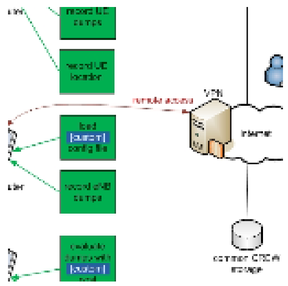
Portal: advanced documentation

The sections below contain **advanced information** on the different CREW testbeds. For **information on the benchmarking platform**, please consult the [section of the w-iLab.t testbed on benchmarking](). You can use the menu on the left of this website to navigate through the portal. You can use the menu on the left of this website to navigate through the portal.

The information on the portal will be regularly as additional information and cognitive components become available.

Schematic overview

Please click the thumbnail extracts below to get a full screen view of the different infrastructures. After clicking the thumbnails, click to zoom in. The images may also be downloaded on the bottom of this page.

TWIST - Berlin	w-iLab.t - Gent	Iris - Dublin	LTE-Advanced - Dresden
 Hardware overview	 Hardware overview	 Hardware overview	 Hardware overview
 Usage overview	 Usage overview	 Usage overview	 Usage overview
Access documentation	Access documentation	Access documentation	Access documentation

Attachment	Size
wilab-UsageOverview.png	158.04 KB
wilab-HardwareOverview.png	183.3 KB
TWIST-UsageOverview.png	119.91 KB
TWIST-HardwareOverview.png	114.63 KB
IrisTestbed-HardwareOverview.png	183.23 KB
IrisTestbed-UsageOverview.png	208.95 KB

Attachment	Size
LTE-UsageOverview.png	88.02 KB
LTE-HardwareOverview.png	185.32 KB

IRIS documentation

The reconfigurable radio consists of a general-purpose processor software radio engine, known as IRIS (Implementing Radio in Software) and a minimal hardware frontend. IRIS can be used to create software radios that are reconfigurable in real-time.

Please use the links below to learn more about how Iris can be used.

Testbed Description

Iris is a software radio architecture that has been developed by CTVR, The Telecommunications Research Centre at TCD, Written in C++, Iris is used for constructing complex radio structures and highly reconfigurable radio networks. Its primary research application is to enable a wide range of dynamic spectrum access and cognitive radio experiments. It is a GPP-based radio architecture and uses XML documents to describe the radio structure. This testbed provides a highly flexible architecture for real-time radio reconfigurability based on intelligent observations the radio makes about its surroundings.

Each radio is constructed from fundamental building blocks called components. Each component makes up a single process or calculation that is to be carried out by the radio. For instance, a component might perform the modulation on the signal or scale the signal by a certain amount. Each component supports one or more data types and passes datasets to other components, along with some metadata such as a time stamp and sample rate. There is a data buffer between each component to ensure the data is safe, even if one component is processing data much faster than another.
All components within the radio exist inside an engine. An engine is the environment in which one or more component operates. Each engine defines its own data-flow and reconfiguration mechanisms and runs one or more of its own threads. As with components, each engine is linked by a data buffer. Iris currently features two data types, the PN Engine and the Stack Engine. The PN engine is typically used for PHY layer implementations and is designed for maximum flexibility. It has a unidirectional data flow and runs one thread per engine. The Stack Engine is designed for the implementation of the network stack architecture, where each component is a layer within the stack and runs its own thread of execution. It also has a bidirectional data flow.

Iris's capability to reconfigure the radio on the fly lies in the controllers. A controller exists independently of any engine and runs in its own thread of execution. A controller subscribes to events within components and reconfigures parameters in other components based on the observation of these events. For instance, a controller could be set up to observe the number of packets passing through a certain component and, upon reaching a certain number of packets, change the operating frequency of the radio.

The Iris 2.0 architecture is illustrated in Figure 2. A radio is constructed and configured using XML documents. Each component is named and has its inputs, outputs and exposed parameters explicitly specified. Engines are declared and components are placed in their relevant engines. Controllers are then declared at the top of the XML document and the links between each component are declared at the end of the document.

The hardware components of the testbed at TCD consists of four Quad core machines, each of which has either a USRP 1 or a USRP 2 and RFX2400 daughterboard connected to it. The USRP (Universal Software Radio Peripheral) is a family of hardware used as an RF frontend for software radios. The USRP 1's have an 8MHz bandwidth and the USRP 2's have a 24MHz bandwidth (using Gigabit Ethernet to communicate between the USRP and the computer). The daughterboards are capable of transmitting between 2 and 2.9 GHz.

Apply for an account

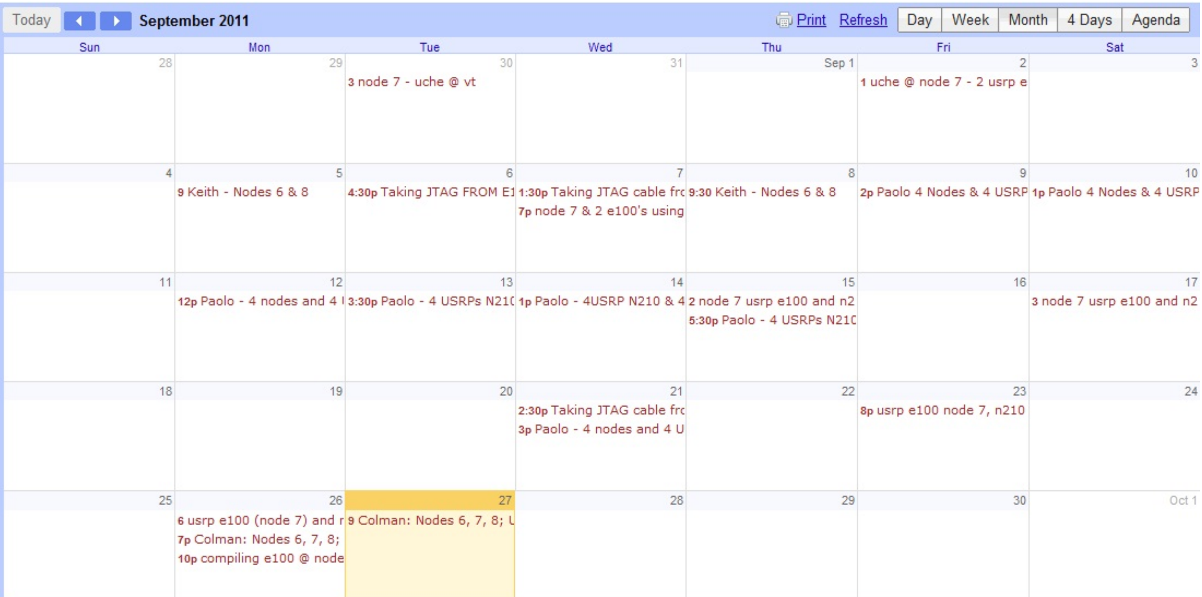
Remote access

The testbed is designed to permit fully remote access for carrying out experiments. This page provides information required to use the testbed from a remote location.

To use the Iris testbed you first require a user account to log onto the ctvr-gateway server. These can be applied for by emailing either tallonj@tcd.ie or finnda@tcd.ie explaining the nature of the experimentation desired to be carried out. Due to limitations in the number of nodes available applications must be handled on a case by case basis.

Scheduling an experiment

On receiving login details, the experimenter will also be issued with access to the Google calendar used for scheduling experiments. It is essential to schedule experiments, specifying which nodes are to be used, prior to use of the testbed. An example shot of the calendar is shown below.



VNC access

Once login details for the ctvr-gateway server are received, use them to login to ctvr-gateway.cs.tcd.ie via SSH. Once you have a terminal for this server open, SSH again onto the node you wish to access as follows:

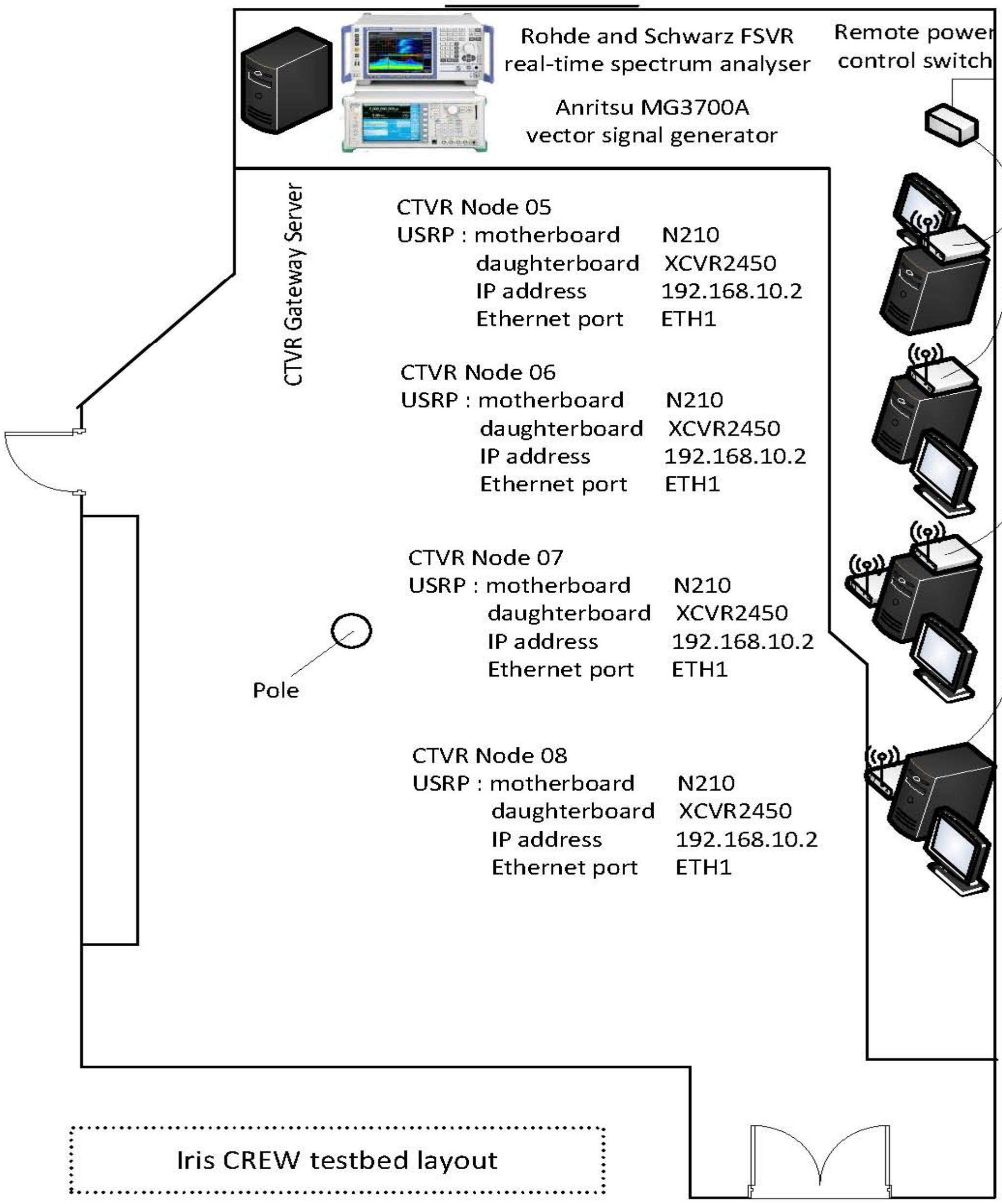
```
ssh nodeuser@ctvr-node07.cs.tcd.ie

vncserver :1 -geometry 1280x900
```

This will create a vncserver on display 1 of node 07 and with a 1280x900 screen resolution. Once the server is running, use a VNC client to connect. In this case, we would connect to ctvr-node07.cs.tcd.ie:1. When you are finished, kill the VNC server on the testbed node as follows:

```
vncserver -kill :1
```

Powering the USRPs

We have installed a remote power switch which allows us to remotely power each of the USRPs on and off. This switch can be controlled through web interface. Access the switch by navigating to http://ctvr-switch.cs.tcd.ie in your web browser. The login details are identical to those used to access the nodes themselves. Here, you can power the USRPs for each node on and off. **Please remember to power USRPs off when you have finished using them.** The following diagram shows the positioning of the different testbed nodes as well as the spectrum analyser and signal generator.



Powering the USRPs via command line/scripts

The remote switch can also be accessed via HTTP Post commands, using a tool such as `curl`, or equivalent calls in a script or program. Using a UNIX based system with `curl` installed

```
curl --data 'P<port>=<command>' http://nodeuser:ctvrnodepass@ctvr-switch.cs.tcd.ie/cmd.html
```

will alter the state of socket <port> according to `<command>`. `<port>` choices are as follows:

* 1 - Node 5 USRP N210 (ETH1)
* 2 - Node 6 USRP N210 (ETH1)
* 3 - Node 7 USRP N210 (ETH1),E100 (ETH2)
* 4 - Node 8 USRP N210 (ETH1)

`<command>` choices are:

* 0 - Switch Off
* 1 - Switch On
* t - Toggle state
* r - Restart

Commands to multiple ports can be strung together using ampersands, as per the following example:

```
curl --data 'P0=r&P1=r&P2=r' http://nodeuser:ctvrnodepass@ctvr-switch.cs.tcd.ie/cmd.html
```

Spectrum Analyser Remote Access

The main spectrum analyser in the testbed room is a Rohde & Schwarz FSVR real-time analyser.

* Host name: ctvr-analyser.cs.tcd.ie
* IP address: 134.226.55.156
* Frequency range: 10Hz - 7GHz
* Real-time analysis with persistence
* Support for IQ analysis (inc. OFDM)
* Maximum sampling rate for IQ acquisition: 128MS/sec

Remote access via VNC

* Verify that the analyser is switched on and connected to the network by pinging it using

```
ping ctvr-analyser.cs.tcd.ie
```

* Use a VNC client to connect to **ctvr-analyser.cs.tcd.ie**

Remote control and IQ acquisition using Matlab

In order to connect to the analyser and arbitrary waveform generator using Matlab, you must first install the National Instruments VISA runtime engine.

* Download the runtime engine for your operating system and install:

* **Windows**: : Runtime engine for Windows (34MB .exe)
* **Linux/SUSE/RedHat**: Runtime engine for Linux/SUSE/RedHat (6MB .iso)
* **Mac OS X**: Runtime engine for Mac OS X (6MB .dmg)

Use of licensed bands

For use of wireless spectrum outside of unlicensed bands the experimenter is directed here.

Attachment	Size
Iris_Testbed_Lab_Diagram.jpg	150 KB
ctvr_testbed_google_calendar.jpg	101.79 KB

First example experiment

The full installation instructions for iris can be found at: https://ntrg020.cs.tcd.ie/irisv2/

The wiki contains information on how to install iris on both Windows and Linux OS.

As well as information on how to run a radio and on the test bed in general.

In this sample experiment we will run a simple radio and then adapt a component and add a controller, with a view to exploring the basic functionality of both. The steps a researcher should follow to complete the experiment are outlined below.

1. Follow the instructions outlined in the wiki to run radio, OFDMFileReadWrite.XML

2. If this radio is functioning correctly, "radio running" will appear on the command line.

3. To add a controller to the radio, we must first create an event in one of the components to which the controller can subscribe. To do this, open the shaped OFDM modulator and register an event in the constructor function.

4. Once the event is registered we must create a condition that must be satisfied for the event to be activated. To do this, open the "process" function (as this is where all the calculations are carried out) and specify a condition that activates the controller whenever, for example, 100 packets have passed through.

5. Once this has been done the controller can be made. Open the "example" controller; this gives us a template to work with.

6. Within the controller we must do two things, subscribe to the event that has been set up in the component and specify the parameter that we wish to change as well as the value we wish to change it to.

7. To change the parameter, we specify the name of the parameter as well as the component and engine that it is in. These are assigned in the "ProcessEvent" function.

8. The logic that dictates what the parameter is changed to also goes in this function.

9. Recompile all the relevant code, include the controller in the XML file and run the radio as before.
If the radio is running properly, you should see the event being triggered on the command line and the new value of the parameter in question.

Test and Trial Ireland

In order to enable research into innovative new technologies, which would require transmission and reception testing within licensed bands, Test and Trial Ireland have the ability to make certain bands in the Irish wireless spectrum available for use. Test and Trial Ireland is a licensing programme which was launched by the Commission for Communications Regulation in Ireland (ComReg).

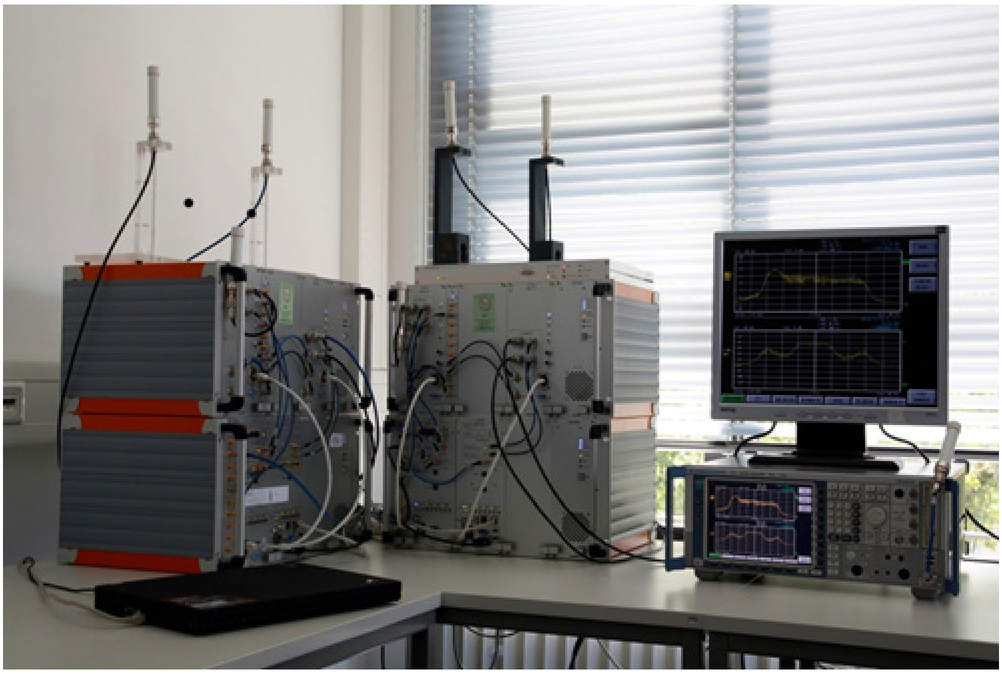
If the experimenter requires use of licensed bands further details on the programme, as well as information about how to apply for spectrum, are available at http://www.testandtrial.ie/.

LTE advanced documentation

Hardware

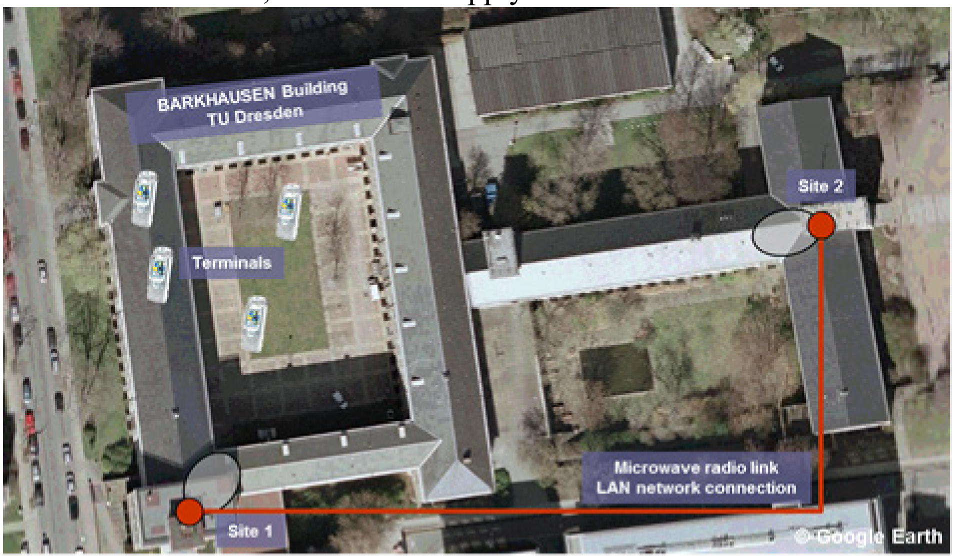
Signal processing hardware by Signalion (www.signalion.com)
• Sorbas602 eNodeB Simulator with ZF Interface to a Sorbas Radio Unit ("ZF Interconnect")
• Sorbas202 Test UE with ZF Interface to a Sorbas Radio Unit ("ZF Interconnect")
• Sorbas472 Radio Unit by Signalion: EUTRAN band VII(2.6 GHz), 20MHz bandwidth, Tx power approx. 15dBm (indoor) and approx. 30dBm (outdoor), supports up to two Tx and two Rx channels

Available hardware
Indoor lab:
• 5 stationary eNBs (on desks)
• 4 stationary UEs (on desks)
• 2 mobile UEs (mounted on studio racks)
• 2 mobile UEs (mounted on carts)



Outdoor lab:

- 2 eNBs with 1 sector each (mounted on the roof of the building)
- 3 mobile UEs (mounted on rickshaw/bus)
- 6 batteries, can supply an UE for around 2-4 hours



Measurement equipment
• Rhode&Schwarz FSH4
• Rhode&Schwarz FSQ8

Tools

SimpleProxy 1.3.1
This tool is installed on all eNB control computers and is used to connect to an eNB, load a configuration and dump IQ data at eNB.

TestUE Config
This tool is installed on all UE control computers and is used to configure an UE.

Test UE Trace
This tool is installed on all UE control computers and is used to monitor UE activity in real-time

UE_dump_tool
This tool is installed on all UE control computers and is used to record the UE's IQ data dumps.

Getting started: A basic tutorial

This tutorial explains how to set up a basic transmission. Download this Zip archive with all necessary default configuration files.

Setup the eNB

- Power the hardware
    - Sorbas eNB Simulator
    - Radio Unit
    - eNB control computer
- Configure the eNB

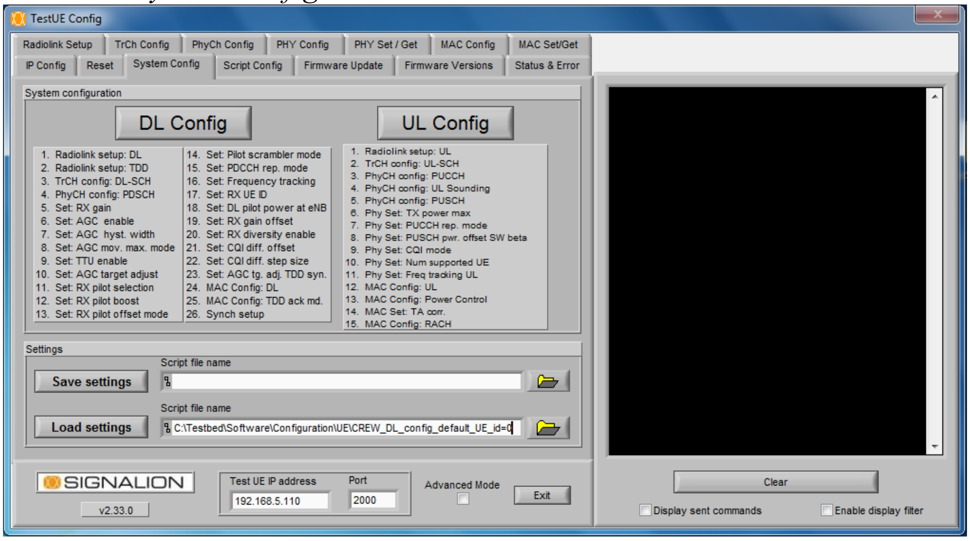
- Open the *SimpleProxy 1.3.1* tool

- Click *Load Settings* and select *CREW_DL_config_default_1eNB_2UEs.xml* or *CREW_UL_config_default_1eNB_2UEs.xml*
- Click *Reset* to reset the eNB
- Wait for eNB broadcast message to appear in the logging box below
- Click *Config* to send the configuration to the eNB
- Check logging box for errors

Setup the UE

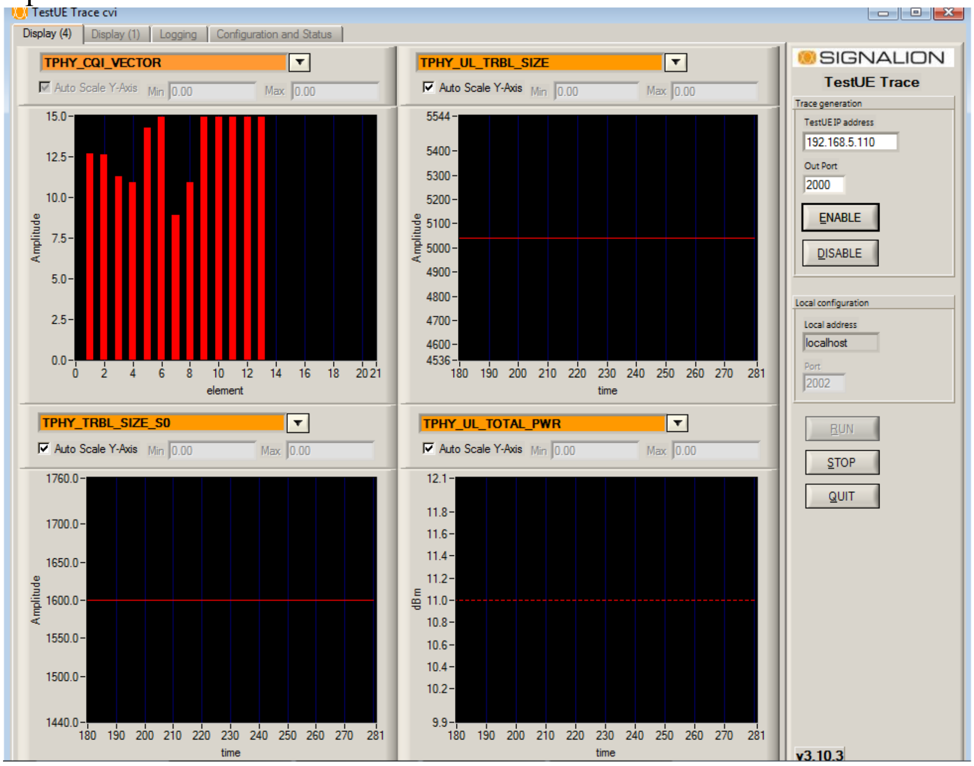
- Power the hardware
  - Sorbas Test UE
  - Radio Unit
  - UE control computer
- Configure the UE
  - Open the *TestUE Config* tool

- Select the *System Config* tab



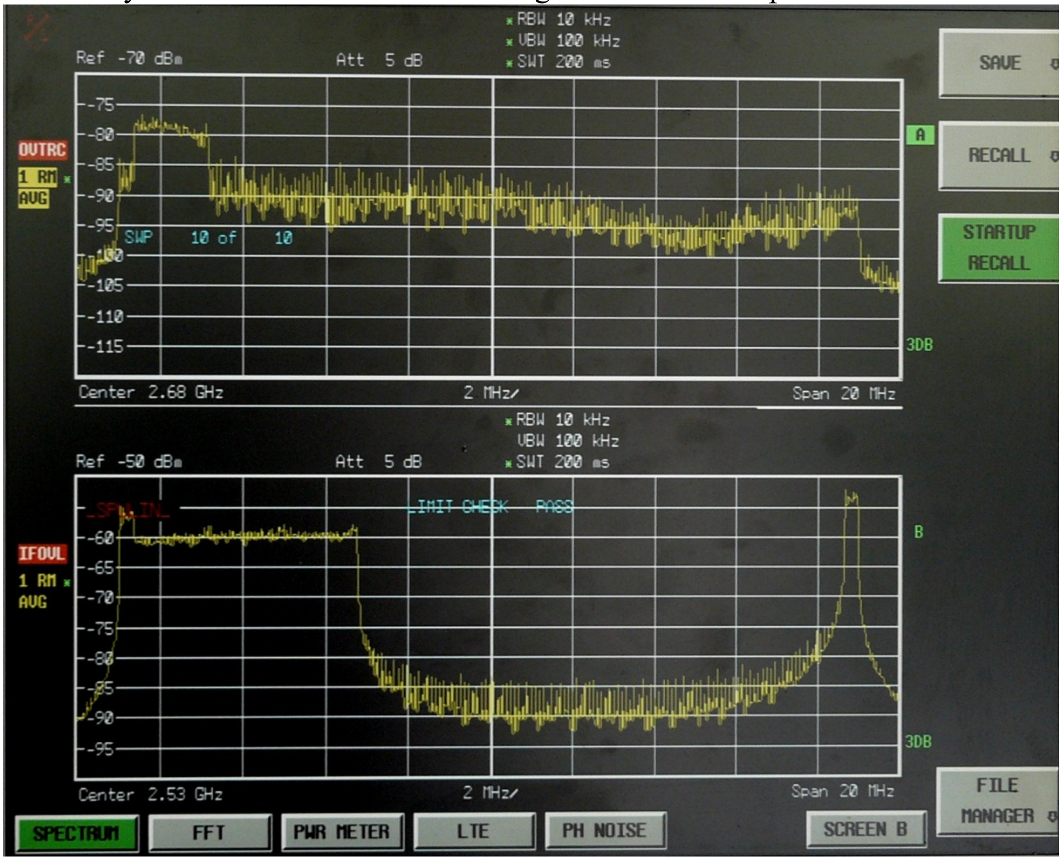
- Click *Load settings* and select *CREW_DL_config_default_UE_id=0* or *CREW_UL_config_default_UE_id=0*
- Click *DL config* and wait for UE response
- Click *UL config* and wait for UE response

- Trace
  - Open the *TestUE Trace* tool



  - Select the *Display (4)* tab
  - Click *Enable*
  - Click *Run*

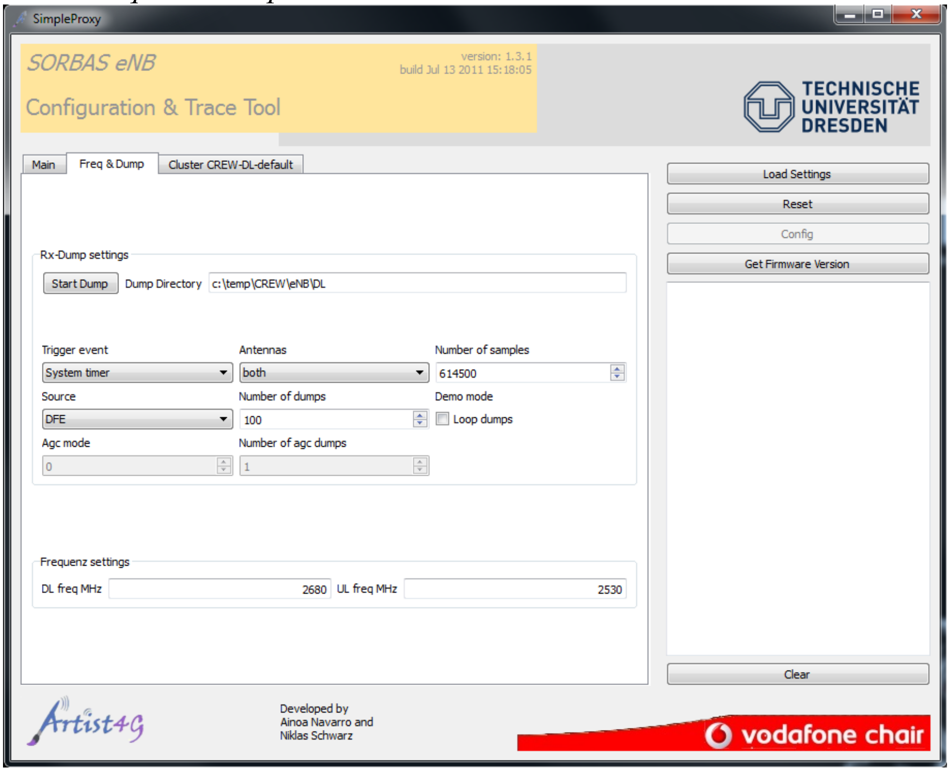
The system is now running. Check spectrum on R&S FSQ.



Record IQ data dumps

- Dump at the eNB
    - Open the *SimpleProxy 1.3.1* tool

- Click *Freq && Dump* tab



- Click *Start Dump*

- Dump at the UE
  - Browse into the directory of the *UE_dump_tool*
  - Click *start.bat*
- Process IQ dumps
  - Extract IQ samples and AGC values with *dumpDemux.m* script
  - Perform further processing in Matlab

Deviations from LTE Rel. 8

Please note that the TU Dresden testbed supports LTE Rel. 8 functionality for the most parts, however there are several deviations:

Downlink
The frame structure and control channels slightly different:
• PDCCH is always on the 2. OFDM-symbol (variable position according to Rel. 8)
• PHICH is not in the first OFDM symbol and has a different structure/content
• PCFICH is not supported
• PBCH is not supported

Uplink
The uplink operates with OFDM modulation.

Please contact us at nicola.michailow@ifn.et.tu-dresden.deif you want to know if a particular feature is supported.

TWIST documentation

Browse the sections below for information about the TWIST testbed.

Introduction and overview of capabilities

TKN Wireless Indoor Sensor network Testbed (TWIST)

The TKN Wireless Indoor Sensor network Testbed (TWIST), developed by the Telecommunication Networks Group (TKN) at the Technische Universität Berlin, is a scalable and flexible testbed architecture for experimenting with wireless sensor network applications in an indoor setting. The TWIST instance deployed at the TKN group includes 204 sensor nodes and spans three floors of the FT building on the TU Berlin campus, resulting in more than 1500 square meters of instrumented office space. TWIST can be used locally or remotely via a webinterface.

Mobile components

In addition to TWIST, which is a fixed testbed infrastructure, CREW experiments involving mobility can be carried out in the TKN premises using additional mobile equipment. The use of this equipment requires that experimenters are present at the TKN premises. The mobile components are:

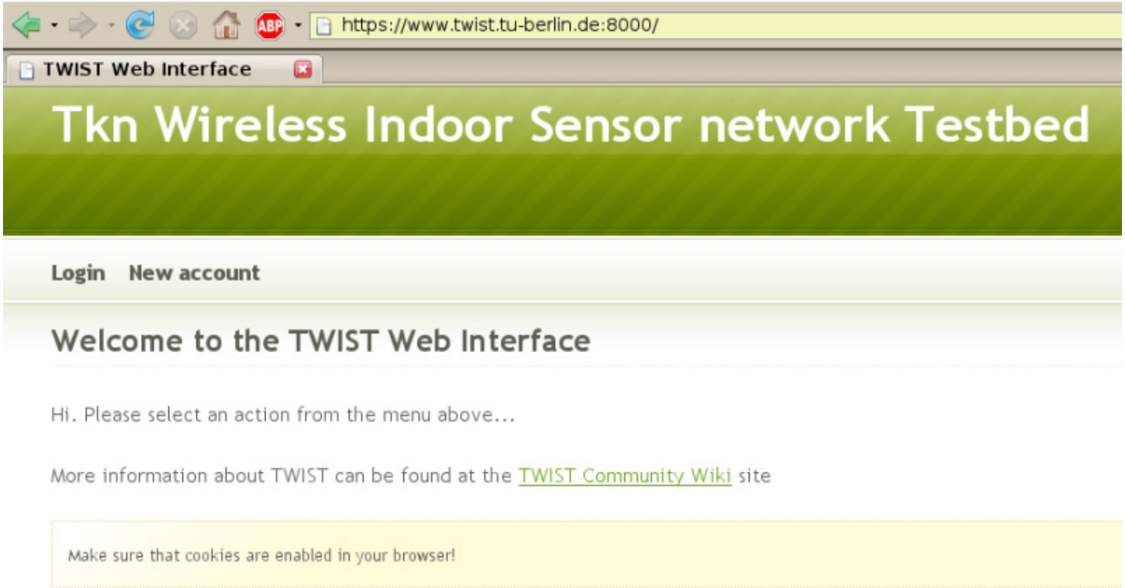
- 1 mobile robot: iRobot Roomba together with a Microsoft Kinect sensor. The robot runs ROS (an open-source, meta-operating system) and it can be programmed to follow certain trajectories in the TWIST building. Shimmer2 sensor nodes or WiSpy devices (see below) can be mounted on the robot, e.g. to record RF environmental maps, or perform experiments emulating body area networks (BANs) as well as experiments involving interaction between a mobile network and the fixed TWIST infrastructure.
- 8 Shimmer2 nodes, which are wearable sensor nodes similar to the popular TelsoB platform and can be attached to a person (or robot).
- 5 WiSpy 2.4x USB Spectrum Analyzers, which are low-cost devices to scan RF noise in the 2.4 GHz ISM band.

Getting started: tutorials

Below you find information on how to get started using the TWIST testbed. Most steps involve remote access via the TWIST web interface, but there is also a more advanced tutorial on how to control TWIST via the cURL command line tool.

Requesting a user account

To access the TKN instance of the TWIST web interface you need to have registered an account. If you are not yet registered, go to the TWIST web interface where you should see the following welcome page:



Make sure that your browser has cookies enabled and click on "New account". In the form fill in your name, email address and choose a username (at least 6 characters) and a password. Make sure you confirm the password and answer the spam control question. Then press the "Request" button; if you filled in the form correctly you will see a new page saying "Successful account request".

Now go to the TWIST terms of use page. Copy and paste the content of this page into an email, add the requested information (the nature of the intended experiments, etc.) and send this email to the TWIST administrator (email address is given on the same webpage). Please also make sure that you explain your relationship to the CREW project.

The last step in obtaining an account is in the responsibility of the administrator, and you will be notified by email when your account has been activated. If there are any problems, please contact Jan Hauer.

Attachment	Size
TWIST_login_screen.png	96.54 KB

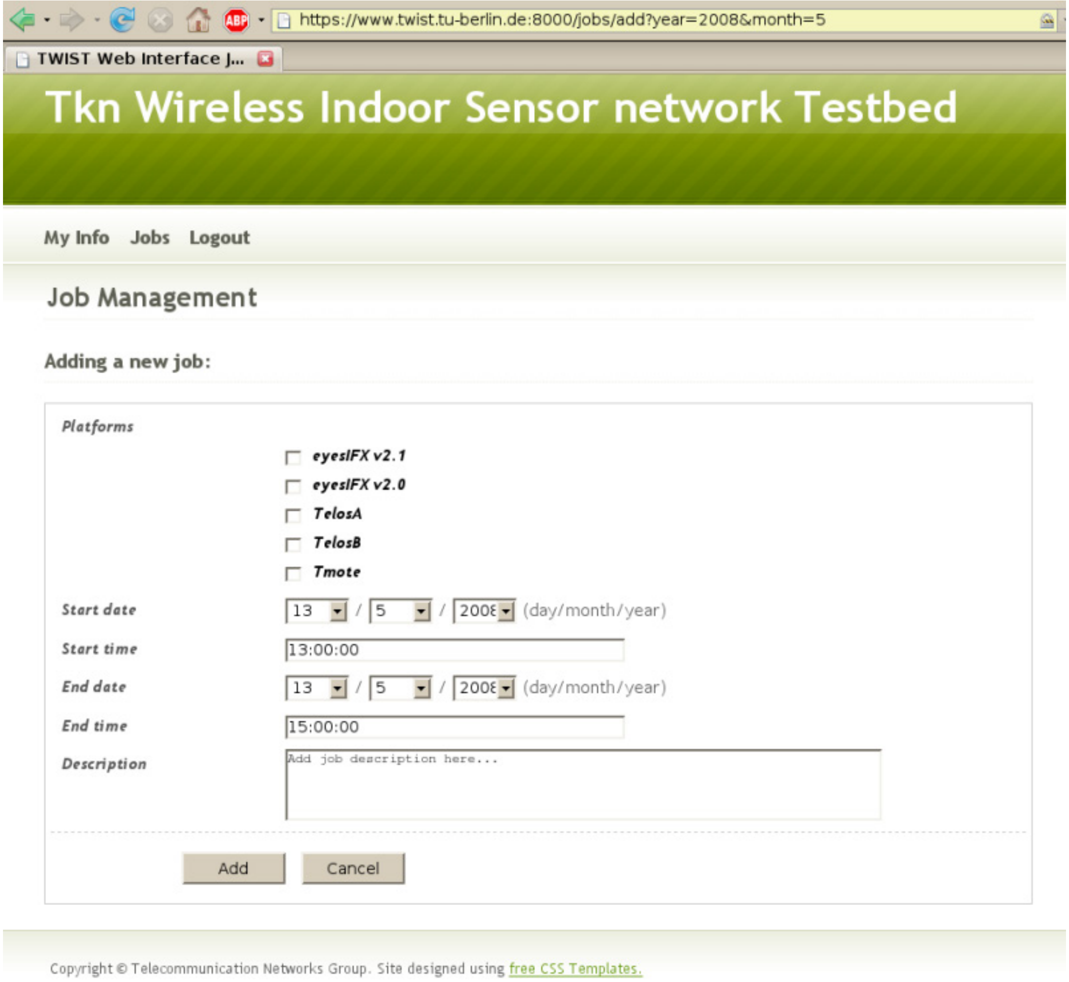
Running a simple experiment

Installing a node image

In this section we install the TinyOS 2 *Oscilloscope* application on a set of Tmote Sky nodes in the TKN TWIST testbed. The *Oscilloscope* application is described in the TinyOS 2 tutorial 5. After you have compiled the application with `make telosb` open a web browser and access the TWIST web interface. Press the "Login" button, enter your username and password and then click on "Sign in". If you have not yet registered a TWIST user account take a look at this tutorial page.

You will see a welcome page where you have three options: manage and update your account settings ("My Info"), schedule and control jobs in the testbed ("Jobs") or logout ("Logout").

Click on "Jobs" and you will see a list of scheduled jobs, i.e. the currently active jobs as well as pending future jobs. Take a close look at the list and find a time period for which Tmote/TelosB nodes are not reserved by someone else. Then click on "Add" and you will see the Job Management page as follows:



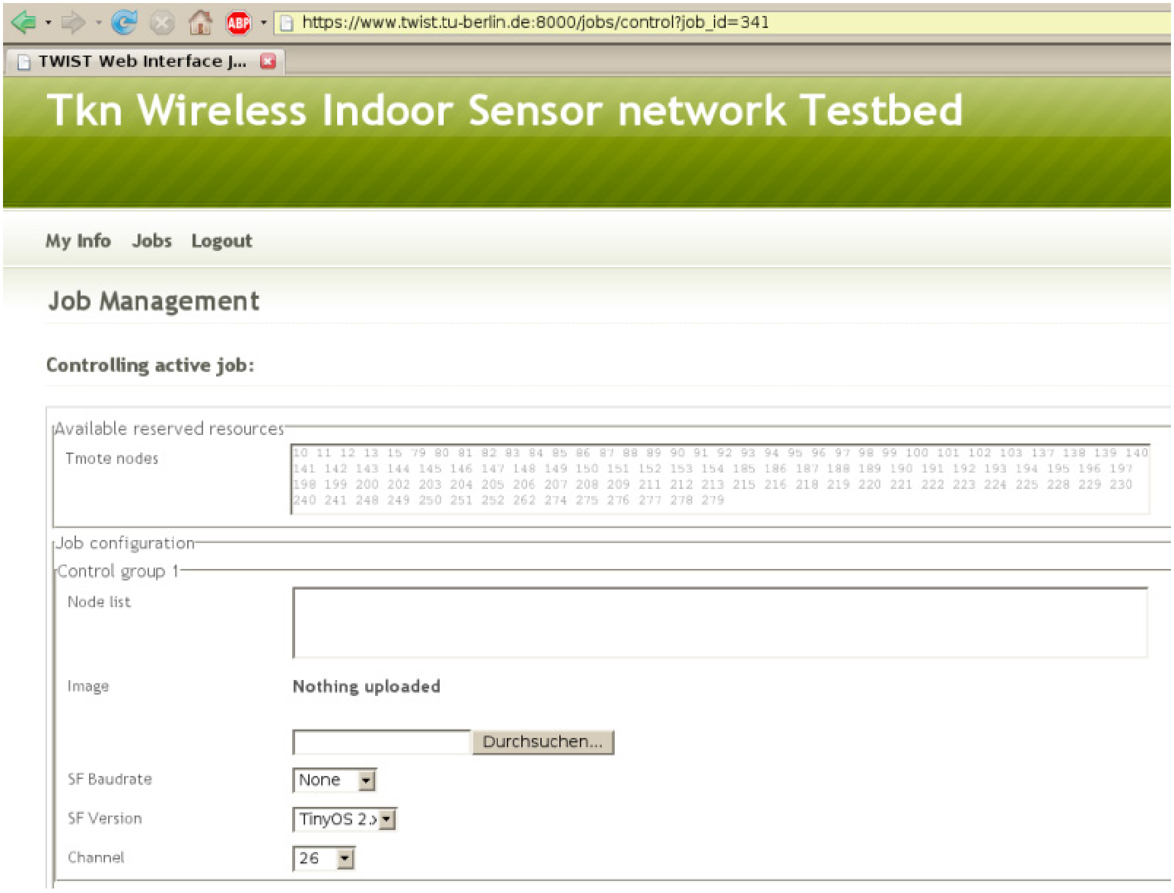
Under "Platforms" select Tmote; then choose a "Start/End date" and "Start/End time" such that the time interval is not overlapping with other jobs, which you checked in the previous step. You cannot make a real mistake here, because the system will automatically check for and not permit jobs that are overlapping in time if they use the same mote platform. However, different platforms (eyesIFX vs. Tmote) *may* be used concurrently. In the field "Description" enter a short note on what you plan to do in your job, such as "Testing the T2 Oscilloscope application", then click on "Add". If the time interval that you entered was accepted you will be taken back to the list of scheduled jobs, otherwise you get an error message and need to adapt the values.

The list of "Scheduled jobs" should now include your job. Your entry is likely to have gray background colour, meaning that it is registered but not yet active. The current system time is always shown in the upper right corner of the page and once your job becomes active -- its start time is shown in the column "Start" -- the background colour of your entry will turn yellow (you need to click the reload button of your browser).

When your job is active apply a tick mark at the left side of the entry and press the "Control" button at the bottom (the "Edit" button would be used to change the time of your job and with the "Delete" button you can remove your job).

Hint: When your job is active (during a experiment) you can still extend its "End time" by clicking on "Edit" on the "Jobs" page, provided that the new "End time" does not overlap with other registered jobs.

After you have clicked the "Control" button you will see the page for controlling your active job as shown in this figure:



This page is divided into the list of Tmote node IDs available in the testbed ("Available reserved resources"), a section for submitting up to three different program images to be programmed on a subset of the nodes ("Job configuration") and a set of buttons (on the bottom, not shown in Figure 3) to perform some actions, such as installing the image(s) on the nodes.

For the TinyOS 2 Oscilloscope application we want to install the Oscilloscope program image on some Tmote nodes, and one node will need to act as gateway and will be programmed with the TinyOS 2 BaseStation application (see TinyOS 2 tutorial 5). Because we will install two different application images, in the "Job configuration" field we will use two of the three "Control group" sections: the "Control group 1" section for the Oscilloscope application and the "Control group 2" section for the BaseStation application.

In the "Control group 1" section, enter in the "Node list" field a whitespace-separated list of the node IDs on which the the Oscilloscope is to be programmed, let's say `10 11 12`. For convenience you can copy & paste from the list of IDs shown on top in the "Available reserved resources" list.

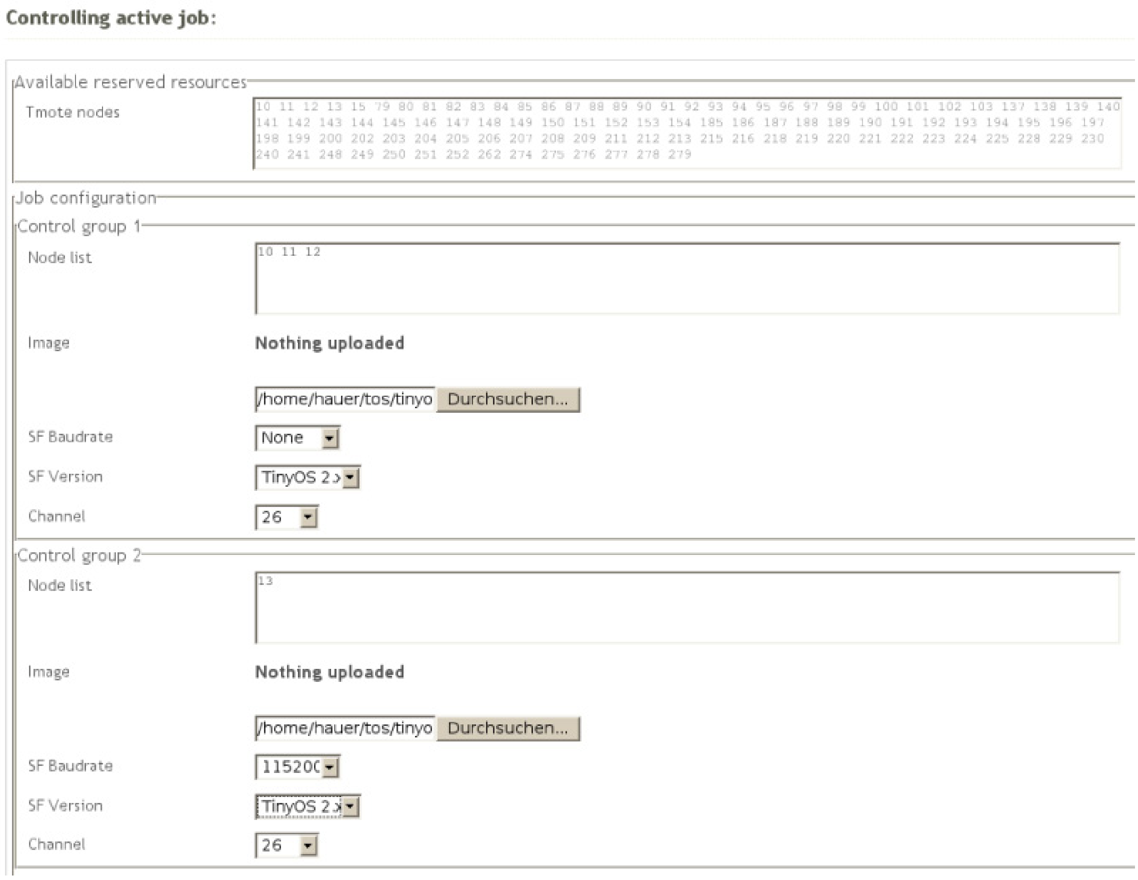
Then click on the "browse..." button next to the "Image" field just below the "Node list" field. Select the Oscilloscope image, which is the `main.exe` in your local `tinyos-2.x/apps/Oscilloscope/build/telosb` (you must have compiled the Oscillocope application with "make telosb" before). The "SF Baudrate" and "SF Version" fields control whether a SerialForwarder will be started for all nodes in the respective "Node list". Since we only need a SerialForwarder for the BaseStation application, we don't change the values (leaving it "None", "TinyOS 2.x"). Finally, "Channel" is the IEEE 802.15.4 channel to be used by the Tmote Sky radio CC2420 (if you change the channel for the Oscilloscope application, make sure that you do the same for the BaseStation application). In fact, the value of the `CC2420_DEF_CHANNEL` symbol inside your progam image will be replaced by the value of the "channel" field and thus, if your application includes the TinyOS 2 CC2420 radio stack, you can still modify the default radio channel after you have compiled the image.

Hint: The node ID is another symbol that is modified for each node individually before programming the image. It is accessible via TOS_NODE_ID in a TinyOS application.

We use the "Control group 2" section for installing the BaseStation program image on another node. In the "Node list" field enter `13` (or whichever node ID you want to use for the BaseStation application) and under "Image" click "browse..." and select the `main.exe` from your local `tinyos-2.x/apps/BaseStation/build/telosb` folder (you must have compiled the BaseStation application with "make telosb" before). Because we want to later establish a serial connection to the BaseStation node, select the pull-down menu under the "SF Baudrate" field and choose a serial baudrate. Whenever this field has a value other than *None* a SerialForwarder will be started for *all* nodes in the respective "Node list". The default baud rate for the "TelosA", ",TelosB" and "Tmote" platforms is 115200 baud.

Hint: You can change the baud rate for a telos node by modifying tinyos-2.x/tos/platforms/telosa/TelosSerialP.nc (this file is included by telosa, telosb and tmote platform). Make sure you recompile your application after changing the file.

The "SF Version" field defines the version of the Serial Forwarder protocol. Because we are using a TinyOS 2 applications select "2" (for a TinyOS 1 application you would select "1"). If the "SF Baudrate" field is *None* then the "SF Version" is ignored. Finally, make sure you select the same "Channel" as the one for the Oscilloscope application. Your configuration should now look like the one shown the next figure:



To actually program the images on the nodes scroll down, press the "Install" button and wait. After not much longer than 1 minute you should see a page with the "Execution log". Check for possible errors (any line "*Could not find symbol [...] ignoring symbol*" is only telling you that the respective symbol was not found/changed in the application image) and scroll down to the bottom where you can find a summary of the "Install" operation. Here you can also see that a SerialForwarder has been started for node 13:

```
To forward SF e.g. for node 13 use: ssh -nNxTL 9013:localhost:9013
twistextern@www.twist.tu-berlin.de
```

In the next section we will establish an ssh tunnel to the TWIST server and connect to the SerialForwarder of the BaseStation node. The remainder of this section summarizes the fields and options for controlling an active job over the web interface.

The following table describes the fields in the "Job configuration" section:

Field	Meaning
Node list	Whitespace separated list of node IDs on which the image will be programmed
Image	The image to be programmed on the nodes in the "Node list"
SF Baudrate	Whether a SerialForwarder is started for each of the nodes in "Node list" and what baudrate it will use
SF version	The version of the SerialForwarder: use 1 for TinyOS 1.x and 2 for TinyOS 2.x

Field	Meaning
Channel	The CC2420 radio channel

The following table describes the buttons on the bottom of the "Controlling active job" page:

Button	Meaning
Install	Installs the image(s) on the node(s) specified in the above "Job Configuration" section; SerialForwarders will be started (if selected) and nodes are powered on
Erase	Programs the TinyOS *Null* application on the selected set of nodes
Reset	Resets (powers off & on) the selected set of nodes
Power On	Cuts the USB power supply for the selected nodes
Power Off	Enables the USB power supply for the selected nodes
Start SF	Starts a SerialForwarder for the selected nodes
Stop SF	Stops the SerialForwarder for the selected nodes
Start Tracing	Stores the serial data output from the nodes in a trace file
Stop Tracing	Stops storing data in a trace file

By pressing the "Start Tracing" button the serial data output from all nodes are automatically stored to a trace file. This file can be accessed via the job control page by pressing the "Traces" button (with your job checked). If you want to use automatic tracing then it is recommended that during install you select the correct "SF Baudrate" and "SF Version". After the install process, you can then simply click on "Start Tracing" without having to manually start the serial forwards.

Exchanging Data via the Serial Connection

Through the previously described "Install" operation a SerialForwarder for the BaseStation node was started. In order for your `tinyos-2.x/apps/Oscilloscope/java/Oscilloscope.java` client to connect to this SerialForwarder, you first need to establish an SSH Tunnel to forward the port of the SerialForwarder to your machine. At the very end of the execution log you find the syntax for this SSH command (type it into a shell):

```
ssh -nNxTL 9013:localhost:9013 twistextern@www.twist.tu-berlin.de
```

Once you have forwarded the port you can access the remote SerialForwarder like a local one. However, when you start your client application make sure that it attaches to the correct port as specified in the SSH Tunnel (the above command forwards the remote port to your local port 9013). For example, to start the JAVA Oscilloscope client you would first need to set the MOTECOM environment variable as follows:

```
export MOTECOM=sf@localhost:9013
```

Now you can start the Oscilloscope GUI by typing:

```
.\run
```

in the `tinyos-2.x/apps/Oscilloscope/java` directory as described in TinyOS 2 tutorial 5.

You should now see an Oscilloscope GUI like the one described in the TinyOS tutorial.

Attachment	Size
TWIST_job_management.png	104.83 KB
TWIST_active_job_control.png	89.52 KB
TWIST_example_job_control.png	61.46 KB

Using cURL for automated control

cURL is a command line tool that can, among other things, transfer files and POST web forms via HTTPS. It can thus be used to automate sequences of operations on the testbed, such as installing an image or powering a node off. Before you can actually control your job you need to authenticate via *cURL* (Step 1) and find you job ID (Step 2). Afterwards you can control your job (Step 3) and download traces (Step 4) associated with your job ID. The following steps list the relevant *cURL* commands.

Step 1: Authenticate

Use the following format to authenticate and store the secure cookie for the future requests (replace YOUR_USER_NAME and YOUR_PASSWORD with your username and password, respectively):

```
curl -L -k --cookie /tmp/cookies.txt --cookie-jar /tmp/cookies.txt -d
'username=YOUR_USER_NAME' -d 'password=YOUR_PASSWORD' -d 'commit=Sign in'
https://www.twist.tu-berlin.de:8000/__login__
```

Note that all data fields have to be URL encoded either implicitly using --data-urlencode or explicitly (in case you have special characters in username/password)

Step 2: Find the job_id

You need to know the job_id before you can use curl to control it. This can also be done by fetching and parsing the jobs page with cURL, maybe passing the output through "tidy"

```
curl -L -k --cookie /tmp/cookies.txt --cookie-jar /tmp/cookies.txt
https://www.twist.tu-berlin.de:8000/jobs | tidy
```

Step 3: Control

The following is a list of examples on how to control a job. Make sure that you replace the *job_id* and *node IDs*.

- **Erase -** For job_id 346, erase nodes 12 and 13:

```
curl -k --cookie /tmp/cookies.txt --cookie-jar /tmp/cookies.txt -F
__nevow_form__=controlJob -F job_id=346 -F ctrl.grp1.nodes="12 13" -F
erase=Erase https://www.twist.tu-berlin.de:8000/jobs/control
```

- **Install -** For job_id 346, install TestSerialBandwidth on nodes 12 and 13 and start serial forwarders:

```
curl -k --cookie /tmp/cookies.txt --cookie-jar /tmp/cookies.txt -F
__nevow_form__=controlJob -F job_id=346 -F ctrl.grp1.nodes="12 13" -F
ctrl.grp1.image=@/home/hanjo/tos/tinyos-
2.x/apps/tests/TestSerialBandwidth/build/telosb/main.exe -F
ctrl.grp1.sfversion=2 -F ctrl.grp1.sfspeed=115200 -F install=Install
https://www.twist.tu-berlin.de:8000/jobs/control
```

- **Power Off -** For job_id 346, power off nodes 12 and 13:

```
curl -k --cookie /tmp/cookies.txt --cookie-jar /tmp/cookies.txt -F
__nevow_form__=controlJob -F job_id=346 -F ctrl.grp1.nodes="12 13" -F
'power_off=Power Off' https://www.twist.tu-
berlin.de:8000/jobs/control
```

- **Power On -** For job_id 346, power on nodes 12 and 13:

```
curl -k --cookie /tmp/cookies.txt --cookie-jar /tmp/cookies.txt -F
__nevow_form__=controlJob -F job_id=346 -F ctrl.grp1.nodes="12 13" -F
'power_on=Power On' https://www.twist.tu-berlin.de:8000/jobs/control
```

- **Start Tracing -** For job_id 346, start tracing on nodes 12 and 13:

```
curl -k --cookie /tmp/cookies.txt --cookie-jar /tmp/cookies.txt -F
__nevow_form__=controlJob -F job_id=346 -F ctrl.grp1.nodes="12 13" -F
'start_tracing=Start Tracing' https://www.twist.tu-
berlin.de:8000/jobs/control
```

- **Stop Tracing -** For job_id 346, stop tracing on nodes 12 and 13:

```
curl -k --cookie /tmp/cookies.txt --cookie-jar /tmp/cookies.txt -F
__nevow_form__=controlJob -F job_id=346 -F ctrl.grp1.nodes="12 13" -F
'stop_tracing=Stop Tracing' https://www.twist.tu-
berlin.de:8000/jobs/control
```

Step 4: Collect data

To collect the specific trace file from archived job 336

```
curl -g -k --cookie /tmp/cookies.txt --cookie-jar /tmp/cookies.txt -d
'job_id=339' -d 'trace_name=trace_20080507_114824.0.txt.gz' -o
trace_20080507_114824.0.txt.gz https://www.twist.tu-
berlin.de:8000/jobs/archive/traces/download
```

Hardware and testbed lay-out

The TKN Wireless Indoor Sensor network Testbed (TWIST), developed by the Telecommunication Networks Group (TKN) at the Technische Universität Berlin, is a scalable and flexible testbed architecture for experimenting with wireless sensor network applications in an indoor setting. It provides basic services like node configuration, network-wide programming, out-of-band extraction of debug data and gathering of application data, as well as several novel features:

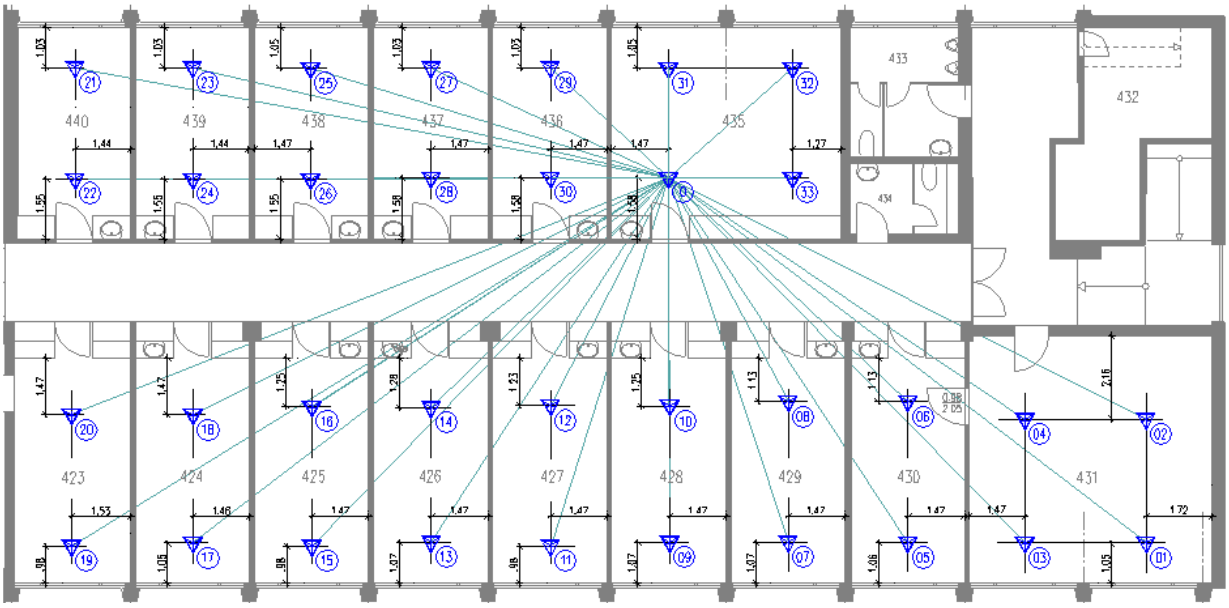
- experiments with heterogeneous node platforms
- support for flat and hierarchical setups
- active power supply control of the nodes

The self-configuration capability, the use of hardware with standardized interfaces and open-source software makes the TWIST architecture scalable, affordable, and easily replicable. The TWIST architecture was published in this paper.

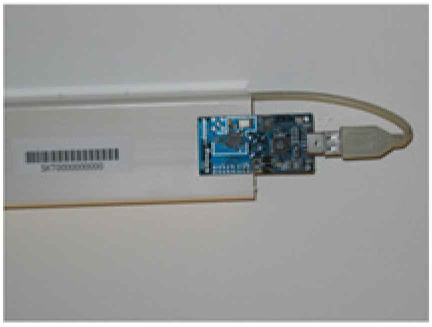
The TWIST instance deployed at the TKN group is one of the largest academic testbeds for indoor deployment scenarios. It spans the three floors of the FT building at the TU Berlin campus, resulting in more than 1500 square meters of instrumented office space. Currently the setup is populated with two sensor node platforms:

- 102 TmoteSky nodes, which are specified in detail here.
- 102 eyesIFXv2 nodes; this platform is an outcome of the EU IST EYES project. The platform is based on an MSP430 MCU and the TDA5250 transceiver, which operates in the 868 MHz ISM band using ASK/FSK modulation with data-rates up to 64 Kbps. A summary of the platform's hardware components is given, for example, in this paper.

In the small rooms, two nodes of each platform are deployed, while the larger ones have four nodes. The setup results in a fairly regular grid deployment pattern with intra node distance of 3m. The following shows the node placement on the 4th floor of the building (floors 3 and 2 have a very similar layout):

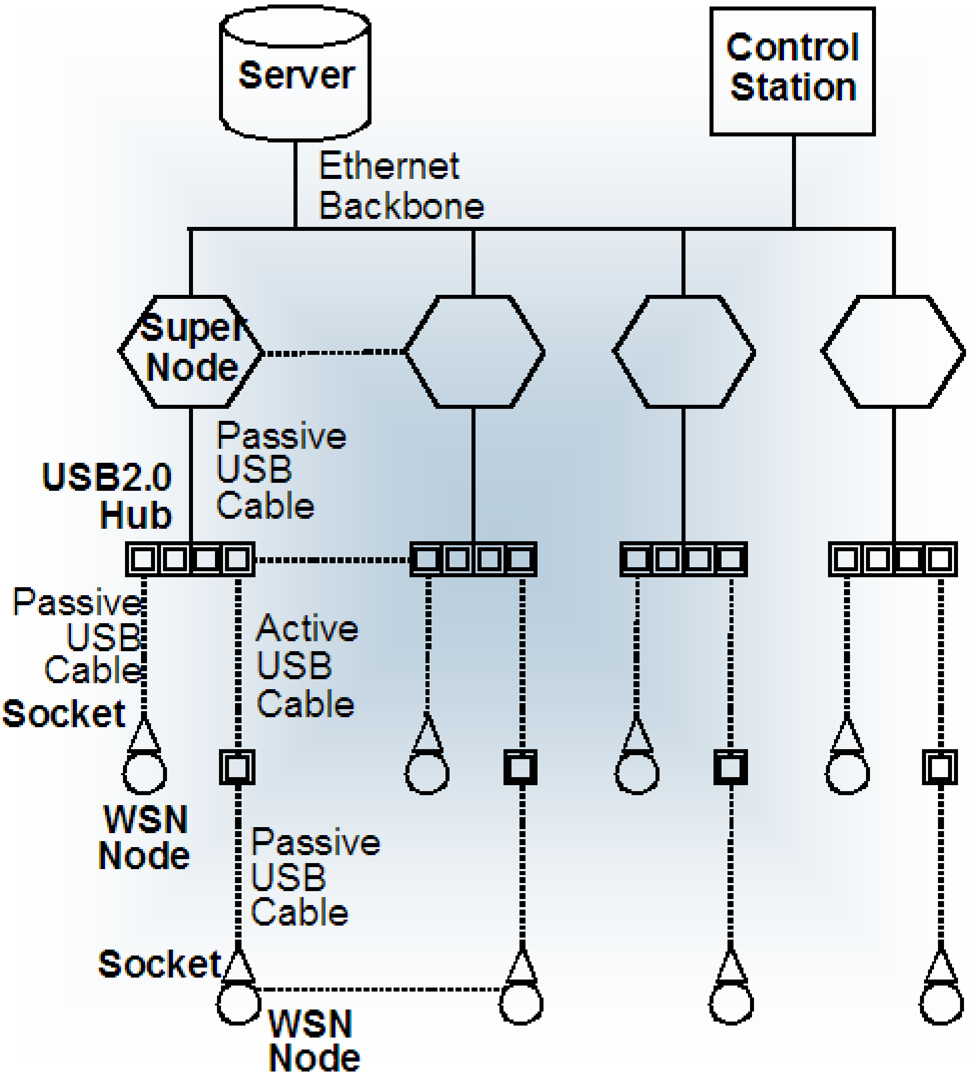


The testbed architecture can be divided into three tiers. The sensor nodes form the lowest tier, they are attached to the ceiling as visualized in the following figure, which shows a Tmote Sky and an eyesIFXv2 node in one of the office rooms:

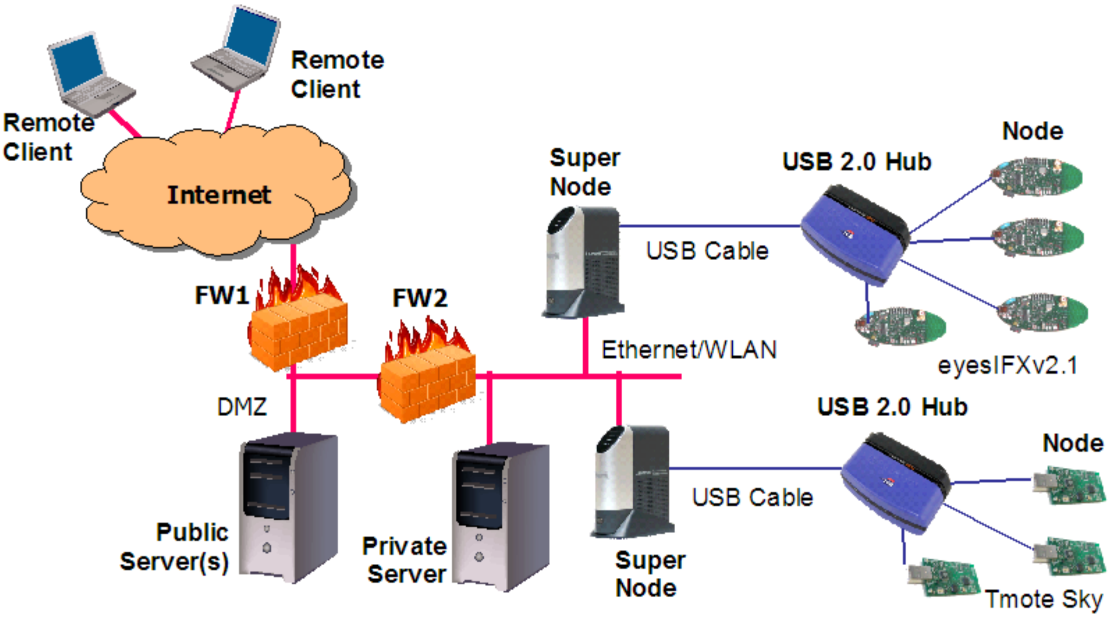


Sensor nodes are connected via USB cabling and USB hubs to the testbed infrastructure. If TWIST only relied on the USB infrastructure, it would have been limited to 127 USB devices (both hubs and sensor nodes) with a maximum distance of 30 m between the control station and the sensor nodes (achieved by daisy-chaining of up to 5 USB hubs). Therefore the TWIST architecture includes a second tier: so-called "super nodes" which are able to interface with the previously described USB infrastructure. We are using the Linksys Network Storage Link for USB2.0 (NSLU2) device as super nodes as depicted in the following picture:

The third and last tier of the architecture is the server and the control stations which interact with the super nodes using the testbed backbone. The server, among other things, implements a PostgreSQL database that stores a number of tables including configuration data like the registered nodes. It also provides remote access via a webinterface. The following figure provides a general overview of the TWIST hardware architecture:



The hardware instantiation of the TWIST hardware architecture at the TKN group is shown in this figure:



Attachment	Size
TWIST_components.png	115.05 KB
TWIST_architecture.png	70.2 KB
TWIST_slug.png	1.04 MB
TWIST_telos.png	1.01 MB
TWIST_floorplan.png	25.42 KB
TWIST_tmote_and_eyesIFXv2.png	66.14 KB

System health monitoring

The system health of the TKN TWIST instance is constantly monitored using the CACTI monitoring tools:

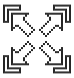
You can either use the CACTI System Health Summary, which displays information on the utilization of the testbed server and super node status. The information is updated every 30 min.

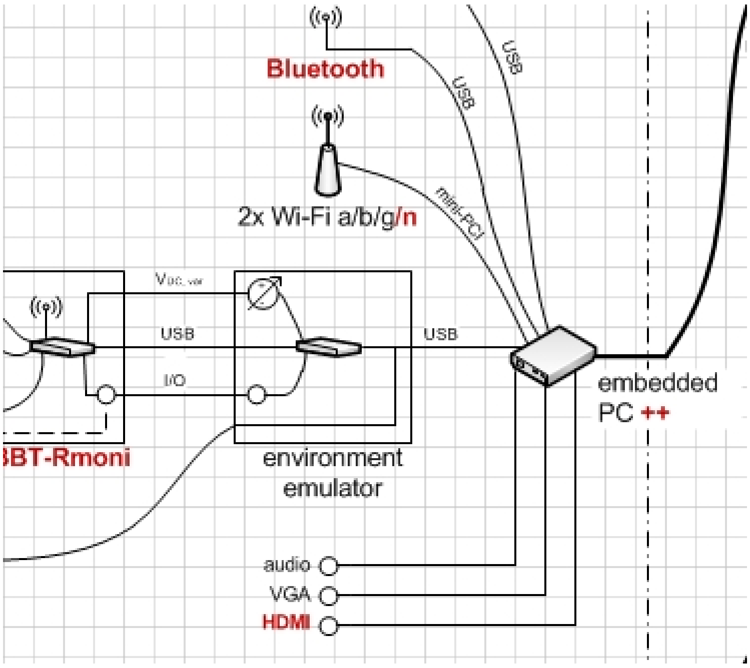
Or you can access the CACTI System Health Browser to see more fine-grained information on some particular systems components (please use account name "guest" and password "guest" to get access to the public data.)

w-iLab.t documentation

The sections contain an overview of all information needed to get you started using the w-iLab.t. If you are new to the testbed, the tutorials are a good place to start.

Introduction to w-iLab.t: overview of capabilities

The w-iLab.t (short name: wilab) is an **experimental, generic, heterogeneous wireless testbed** deployed in the IBBT building and at a second, remote location. w-iLab.t provides a permanent testbed for development and testing of wireless applications via an intuitive web-based interface. w-iLab.t hosts different types of wireless nodes: **sensor nodes, Wi-Fi based nodes, sensing platforms, and cognitive radio platforms** (that are limited to operating in the ISM bands due to license restrictions.) The wireless nodes are also connected over a wired interface for management purposes. Each of the devices can be fully configured by the experimenters. As the Ethernet interfaces that are put in place for management reasons can also be used during experiments as a wired interface, heterogeneous wireless/wired experiments are possible. As such, a very large number of (wireless) network experiments may be executed. Please click on the thumbnail below to get an overview picture of the hardware available in w-iLab.t. After clicking the thumbnail, click  to zoom in.



The two locations that are currently available in the w-iLab.t are:

1. The original "Office" deployment; Nodes (both sensor nodes and embedded PCs with Wi-Fi interfaces) are installed at 200 spots over three floors of an office environment.
2. A new deployment located in Zwijnaarde, nearby Gent, Belgium. All nodes at this location are more powerful in terms of processing power, memory and storage. Nodes are located at 60 spots throughout a utility room.

A short introduction to different possible experiments is presented below.

Sensor node experiments

Registered users can create their own executables, upload these executables, associate those executables with a selection of sensor nodes (this process is called "creating a job"), and schedule the job to be run on wilab. During the job, measurements and management data is logged to a database. This info is presented to the user upon job completion and may then be used for processing and visualization. In addition, real-time visualization tools are provided which make it possible to follow the state of the testbed and the experiment, while a job is still running. As such, w-iLab.t facilitates research in sensor network programming environments, communication protocols, system design, and applications.

Wi-Fi experiments + embedded PC's

Experimenters can also fully control the embedded PC's that are available. The embedded PCs run a Linux distribution and are equipped with two Atheros based Wi-Fi interfaces. These Wi-Fi interfaces may or may not be used during experiments. It is up to the user to define the behavior of the embedded PCs by installing software and/or scripts on the nodes. As such, the embedded PCs can be used for a very broad set of experiments. Just to give a few examples, it is possible to:

- enable a single wireless interface; configure it as an access point -> use of the embedded PC as an access point
- enable two wireless interfaces in ad-hoc mode -> use of the embedded PC als a two-interfaced Wi-Fi ad-hoc node
- install any type of software on the node: e.g. a webserver, a spectrum database, an aggregator node, ...

An experimenter can access the embedded PCs individually via SSH, or distribute software/drivers/kernels/scripts/... to multiple nodes at once, by using the web-based testbed interface. During the experiments, a directory on a w-iLab.t storage server is mounted automatically for logging purposes. Alternatively, experimenters may log information to their own storage servers, or store information to a database.

The default image of the embedded PCs in the office environment comes with the Madwifi wireless driver preinstalled. Experimenters may install their own drivers and protocols to the embedded PCs. As a general rule: everything you are able to do with an embedded PC with Atheros Wi-Fi cards on your own desktop, can also be implemented on a large scale in the testbed.

At the Zwijnaarde location, USB Bluetooth interfaces are also plugged in to the embedded PCs. Please check the hardware overview picture above to know what is connected to the embedded PCs at which location.

Cognitive networking platforms

At the Zwijnaarde location, a set of cognitive networking platforms are available. They can be remotely accessed over the internet. For information on which cognitive devices are available, please check the hardware overview image on top of this page. Again, it is up to the experimenters to decide how to use the hardware that is made available. Signals may only be transmitted in the 2.4 GHz and 5 GHz ISM band due to license restrictions.

Getting started: tutorials

To get familiar with the look and feel of the w-iLab.t testbed, we recommend going through the basic tutorial, in which you will run your first, pre-configured sensor network experiment.

If you want to get familiar with the more advanced functionality of the testbed, you can walk through the advanced tutorials.

Basic tutorial: your first experiment on w-iLab.t

Run your first experiment on w.iLab-t

In this basic tutorial you will learn how to run your first sensor experiment on Wilab.t. The sensor code we will use for this experiment is called RadioPerf. This application is able to send commands over the USB channel to the mote (e.g. start sending radio packets of x bytes to destination y). The mote also periodically sends reports back over the USB channel (e.g. how many packets it received, what the RSSI of the received packets was, ...).

In this tutorial you will learn how to tell a sensor node to start sending packets and afterwards analyze the the result with one of the Wilab.t tools.

1. **Request OpenVPN account**

   Send an e-mail to vwall-ops@atlantis.ugent.be to request an OpenVPN account for the w.iLab-t testbed. Be sure to also mention your affiliation and/or project for which you want access to the testbed.
   We recommend downloading the VPN software from the OpenVPN website.

   Once you installed the software and received the necessary certificates and credentials, you should be able to connect to the w.iLab-t testbed.

   Make sure you run the OpenVPN software as Administrator/root !

2. **Request w.iLab-t account**

   Now that you're able to connect to the w.iLab-t web interface, you can request an account on the testbed by completing the form on the signup page.

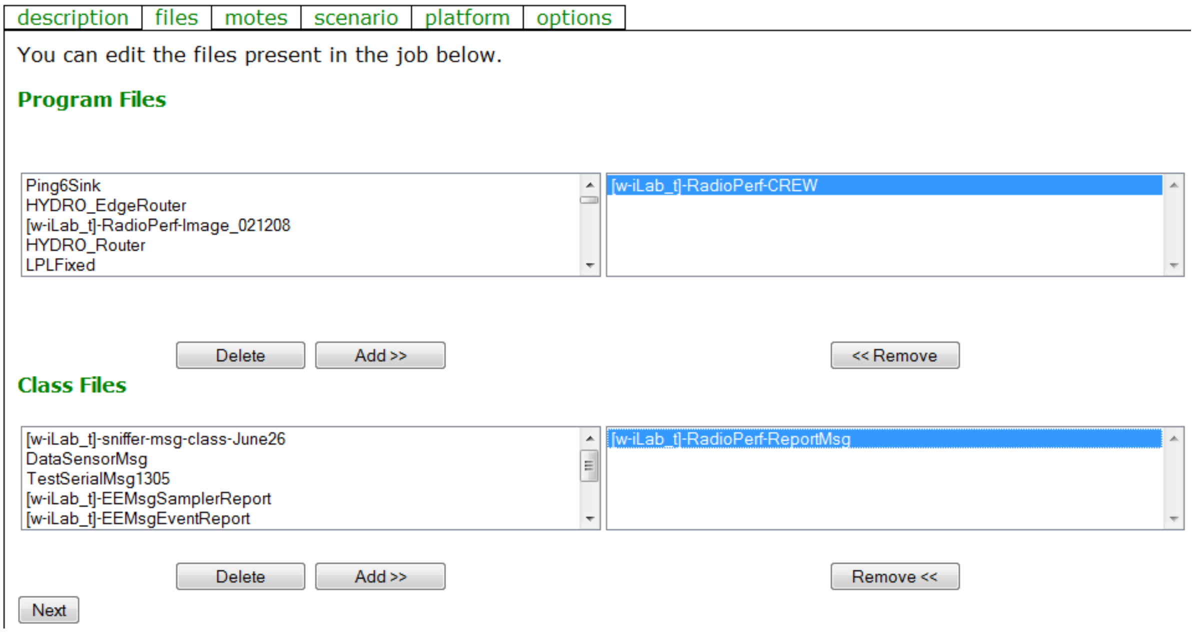
3. **Create your first job**

Once your account has been approved, you can log in to the w.iLab-t testbed.

Now go to the job page to create your first job. Click the **Create new job** button and fill in a name and description. Click **next** or go to the files tab.

In the files tab you must select at least one Program file and one Class File. The **Program File** contains the firmware that will be programmed on the sensor nodes. Sensor nodes can send messages to the w.iLab-t server which will be logged in the database. The **Class Files** define which messages, that are sent by the sensor node, will be logged in the database.

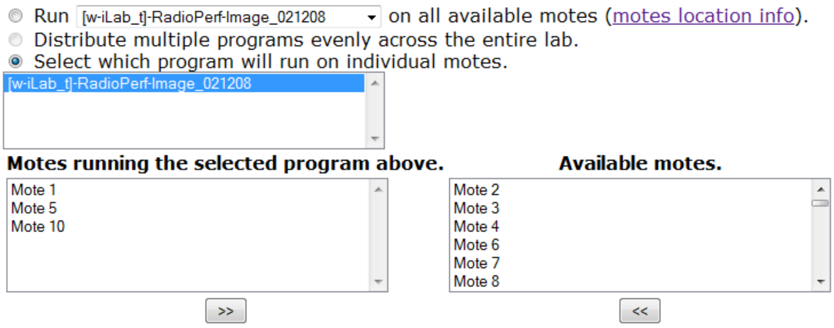
For our first job, we will use the RadioPerf-CREW image as Program File and the RadioPerf-ReportMsg as Class File. Select them in the list on the left and click the **Add>>** button.



At the bottom of the files tab, it is possible to upload your own images and class files.

Click **Next** or go to the motes tab.

You can choose to run the firmware on all available sensor nodes, or pick some specific nodes out of the list. For our first experiment, we can just run the experiment on all available sensor nodes.

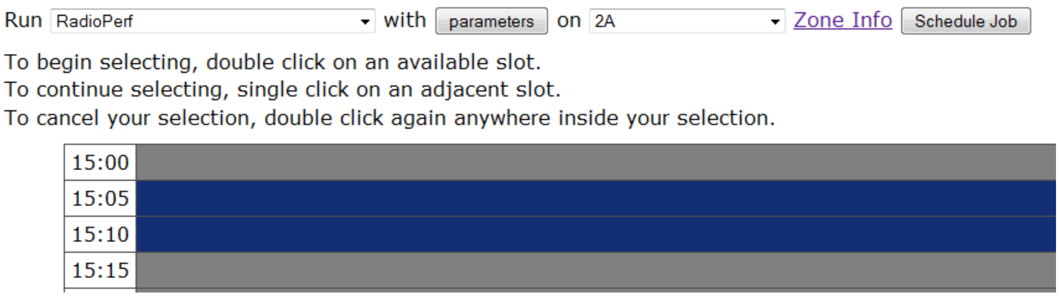


The scenario and platform tabs are not important for our first experiment, so just click the **Submit** button at the bottom of the page.

4. Schedule your first job

Now that we created our first job, we can schedule it to be executed on the testbed. On the schedule page select the job you want to execute and select a zone (part of the testbed) in which you want it to run. (Choose between 1A/1B/2A/2B/3A/3B).

Now double click the first time slot where you want the experiment to start and select some consecutive blocks to determine the duration of the experiment. For this first experiment, 10 minutes should suffice. Click **Schedule Job** to confirm the selection.



5. Analyze results

To analyze the results of your experiment (during or after), you can log in to your personal database via the user info page. You should take note of your wilab Database Name which is listing near the top of the page (this is NOT your email-adres).

After clicking the phpmyadmin link, you can fill in your username and password. On the left side of the phpmyadmin page you see some general databases and one database which is named after your own user name. Click this database to see what tables it contains.
For every job, there should be a table in the database (if it logged any info). Click the **browse** icon to see all the info your experiment has logged.

6. Visualize experiment

The toolbox page contains a list of analyzer and visualizer XML files. Select the **[w-iLab_t]_Visual_RadioPerf** XML file and click **Start Visual** to start the Java applet that will visualize your experiment. Now fill in your database user name (not email) and password.

You will need to install the sun-java6-plugin to get the applet working. The applet will NOT load with the alternative OpenJDK plugin (IcedTea). The applet has been tested in both Firefox and Internet Explorer.

Once the applet has finished loading, you should now see a blue circle with the sensor node id for every active node in your experiment. After some time, every node should log some info to the database and the circles should change color and now also show the estimated noise floor (ENF).

In the next step we will show how we can modify the experiment by changing some parameters.

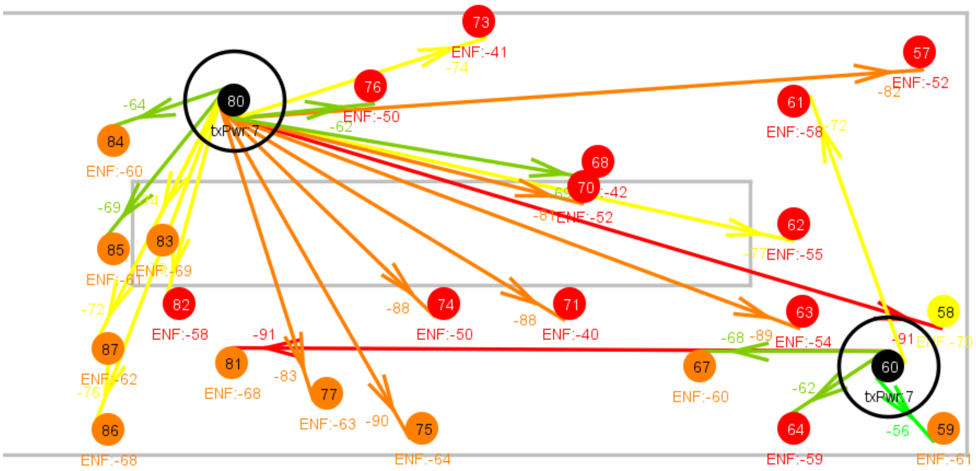
7. Schedule parametrized experiment

In this step we want to schedule the same job, but change some parameters so that one node will broadcast (single hop) some data packets to all nodes in its neighborhood. Therefore we go back to the schedule page , select the job we want to run and then click the **parameters** button. Now look for all parameters starting with RadioPerfP. The default values can be used except for the source parameter. If we want one node to transmit packets to all other nodes (destination value 65535 equals broadcast), we must change this to the id of the transmitting node (e.g. 80 if we run the experiment on zone 2A).

RadioPerfP.channel	20
RadioPerfP.destination	65535
RadioPerfP.interPacketDelay	1000
RadioPerfP.numberOfPacketsToSend	1000
RadioPerfP.packetSize	100
RadioPerfP.source	80
RadioPerfP.txPower	15

Now choose some time slots, select a zone and click the **Schedule Job** button.

Repeat **Step 6** to visualize the parametrized experiment. You should now see arrows from the sending node to all receiving nodes, with an RSSI indication next to the arrows.



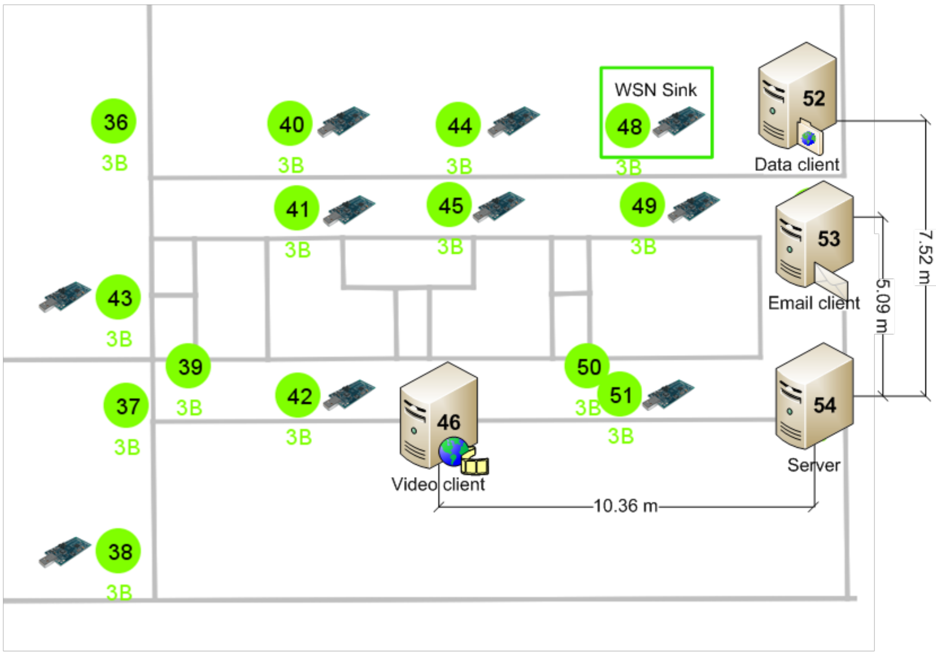
Advanced tutorial: combined wifi and sensor experiment

Advanced w.iLab-t experiment tutorial

In this tutorial we will use the more advanced features of the w.iLab-t testbed by benchmarking a channel switching protocol for wireless sensor networks (WSN) in a wireless environment we create by using linux scripts.

We will use 4 embedded PCs to generate the wireless interference for the wireless sensor network, together with a full zone of sensor nodes that use a simple reporting method to send periodical data to a sensor node sink.

The example topology used for this experiment is shown below, with the 4 embedded PCs being node 46,52,53 and 54. The server is the 802.11g access point, with the other 3 embedded PCs fulfilling the role of stations. All green nodes are the sensor nodes, reporting to node 48.



During this tutorial, we will upload scripts to W-iLab.t to manage the behavior of the PCs, and install a TinyOS image to control the WSN nodes. The goal of this tutorial is to familiarize yourself with the extended features of wilab, including WSN benchmarking, controlling the embedded PCs and visualizing experiment data on the w.iLab-t website.

1. Basic Tutorial

Please go through the basic tutorial before continuing with this advanced tutorial.

2. Create the sensor job

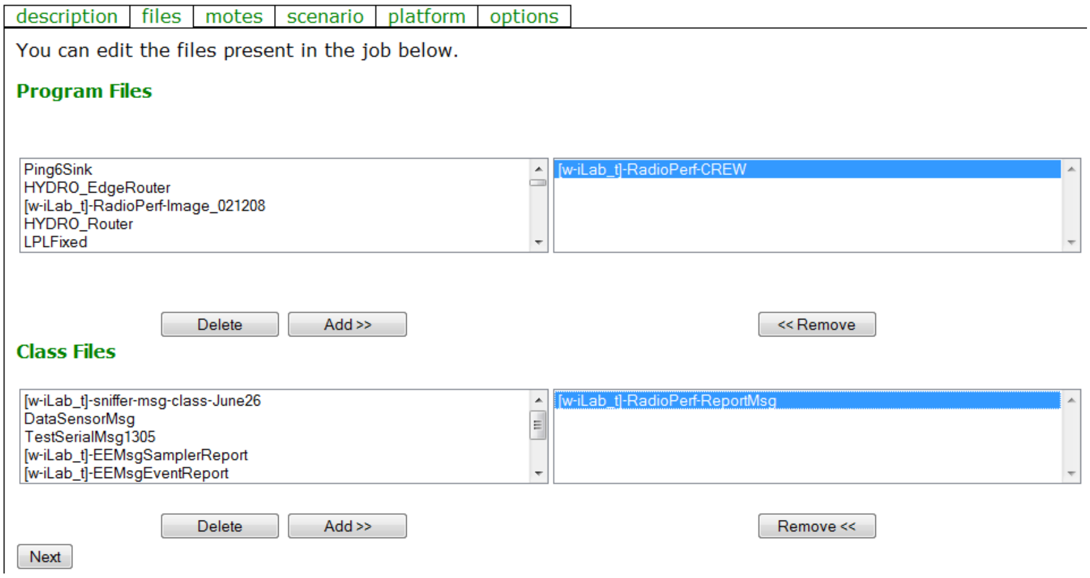
An experiment on Wilab.t can be run on both the sensor nodes and the embedded (Alix) PCs. The sensor part of the experiment is mandatory in the current version of Wilab.t. If you just want to perform an experiment on the embedded PCs, it is allowed to use dummy sensor code. For this tutorial however, we will make full use of the sensor and embedded PC capabilities. This part describes how you can upload your own sensor binaries and create your sensor job.

Go to the job page to create your sensor job. Click the **Create new job** button and fill in a name and description. Click **next** or go to the files tab.

In the files tab you must select at least one Program file and one Class File. The **Program File** contains the firmware that will be programmed on the sensor nodes. Sensor nodes can send messages to the w.iLab-t server which will be logged in the database. The **Class Files** define which messages, that are sent by the sensor node, will be logged in the database.

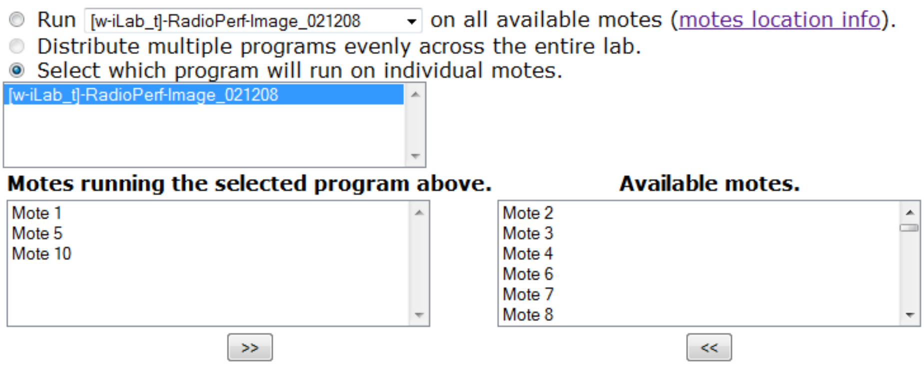
For this tutorial, we will upload our own sensor binaries and message classes. You can find all the necessary files here. After downloading these files, please scroll down to the bottom of the files tab and click **Browse...**. Now select the file you want to upload, provide a reference name (will be used to show the file in the list) and add a description (optional). Click the **Upload** button to start uploading. Repeat this step for every file you want to upload.

Now select the binary and message classes you just uploaded in the list on the left and click the **Add>>** button. Please note the ID (e.g. 6791) displayed to the right of the message class you just added, you will need this later.



Click **Next** or go to the motes tab.

You can choose to run the firmware on all available sensor nodes, or pick some specific nodes out of the list. The right plane shows the nodes excluded from the job whilst the left plane shows participating nodes. For this experiment, we can just run the experiment on all available sensor nodes.



The scenario tab will not be used in this tutorial. For more info on this feature, please read the howto.
We will get back to the platform tab at the end of the next section.

Just click the **Submit** button at the bottom of the page if you are done setting up your sensor job.

3. Define an iPlatform

The previous section described how you can run experiments on the sensor nodes. This part will go into detail on how to run linux scripts and/or binaries on the embedded PC's on Wilab.t.

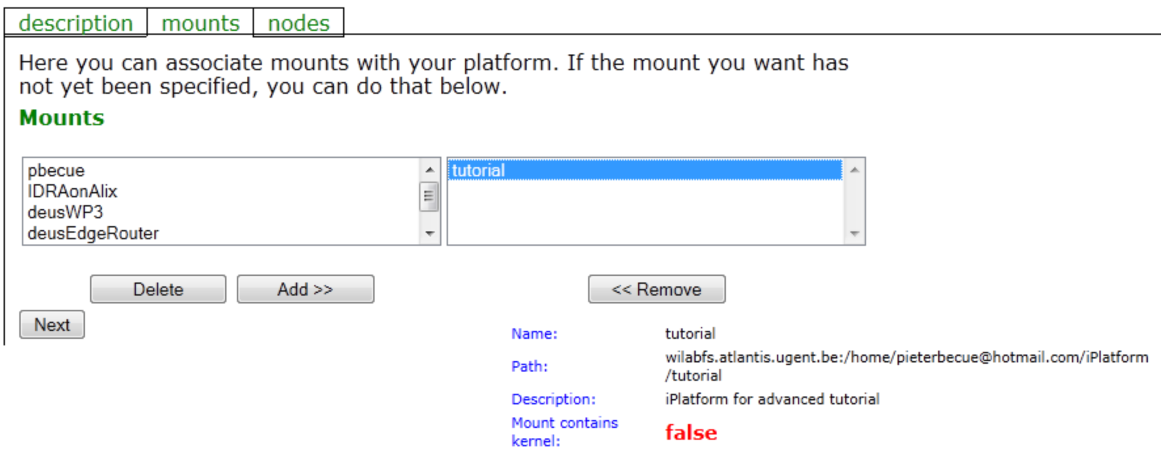
Go to the iPlatform page and click **New Platform**. On the description tab you can insert a name and a description(optional). Be sure NOT to insert any whitespace in the name field. Now click **Next** or go to the mounts tab.

Every Wilab.t user has his own user directory on the Wilab.t fileserver (Wilabfs). Please log in to wilabfs using your Wilab.t database name (see the user info page if you cannot remember it). An example command for the fictional "crewuser" account : *ssh crewuser@wilabfs.atlantis.ugent.be*

Step into the iPlatform directory and create a new folder for your new iPlatform (you might want to choose the same name you inserted on the web page). You can find the files to create your iPlatform here. Please copy all the files to your newly created directory and make the scripts executable using the *chmod +x scriptname* command.

Have a look at the *start_mount_code* file. This script will be executed automatically when your iPlatform is started. So everything you want to execute in your experiment should be called from this script. You can ofcourse create your own script and call it from the *start_mount_code* script as it is done in this tutorial. Please change the user credentials in the "variables" script so that it reflects your settings. Feel free to look at how the scripts interact with the database and log directories, but a complete decomposition is out of scope for this tutorial.

We can now go back to the web interface and select the nfs-mount to upload to Wilab.t. If you named your iPlatform e.g. "tutorial", your nfs-mount will probably look like this : *wilabfs.atlantis.ugent.be:/home/crewuser@crew-project.eu/iPlatform/tutorial*. The directory you define here **MUST** contain the *start_mount_code* script! Otherwise your iPlatform will not be executed.



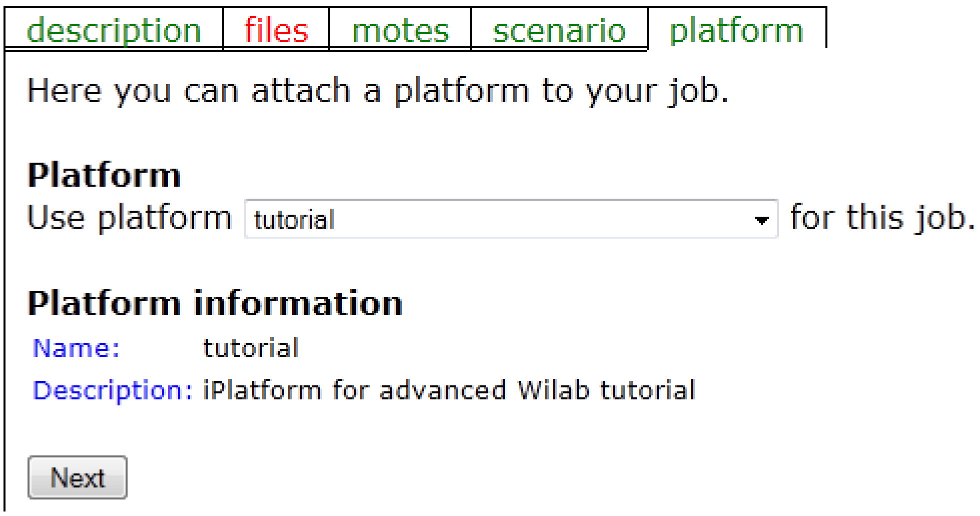
Just provide a reference name and a description(optional) and click **Upload**. The iPlatform can also contain a kernel. If you select this option, the kernel will be uploaded to the embedded PC's and will be executed when your iPlatform is started. We will not use this feature in this tutorial.

After uploading the nfs-mount, you can select it from the list on top of the mounts tab and click **Add>>**.

Proceed to the nodes tab to select the nodes on which you want to execute the iPlatform. This is the same as for the sensor nodes. Please note that in the current version of Wilab.t the sensor node must be included in the experiment in order to be able to run an iPlatform on the embedded PC connected to the sensor node.

Click **Submit** to save your iPlatform.

The last step in creating your job is to link your iPlatform to the sensor job. So go back to the job page, select the job you created in the previous section and click **Edit Job**.
Proceed to the **platform** tab and select your iPlatform from the dropdown list.



Don't forget to press the **Submit** button to save your job.

4. Scheduling your experiment

Now that we created our first job, we can schedule it to be executed on the testbed. On the schedule page select the job you want to execute and select a zone (part of the testbed) in which you want it to run. You should choose between 1A/1B/2A/2B/3A/3B, but the default configuration has been done for 3B. If you wish to run the experiment on a different level, you can change the 3 wifi nodes that will generate interference in the "variable" script, inside the iPlatform directory.

It is also possible to adjust the global variables of your sensor code. This can be a huge time saver, since the sensor code needs to be recompiled and uploaded after each change. The provided sensor code for this tutorial needs some parameters tweaked for the actual experiment, so please click on the **parameters** button.

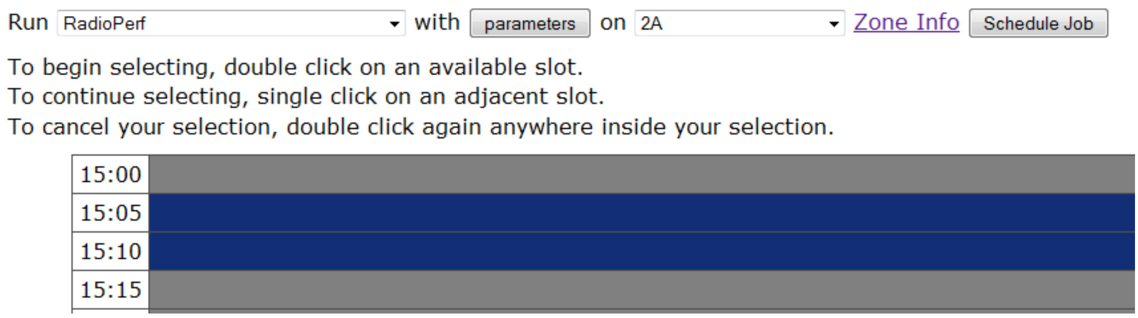
Now all the global parameters of sensor code are listed using a BNF syntax (please click the **bnf** button just above the parameters to learn more). For this tutorial, we are only interested in the prefix *Benchmarking_P*. The following parameters are important for this tutorial:

- **Benchmarking_P.target**: the nodeID of the WSN sink in the experiment (for 3B: *48*)
- **Benchmarking_P.node_purpose**: does a sensor node partake in the experiment? (for 3B: *50,48:0,1*)

For a full API of the benchmarking code, please read more here. After setting the parameters, it is time to schedule our experiment.

(IMPORTANT: the IBBT w.iLab-t does not allow wifi experiments to be performed during office hours. It is advisable that you schedule your experiment from 20u-6u Brussels time or during weekends.)

Now double click the first time slot where you want the experiment to start and select some consecutive blocks to determine the duration of the experiment. For this experiment, 20 minutes should suffice. Click **Schedule Job** to confirm the selection.



Now the experiment will start at the scheduled slot, you can choose to follow the experiment live or perform an analysis afterwards.

5. Analyse your experiment

There are multiple ways to analyse a running or previous experiment. For this tutorial we will be looking at three methods:

- w.iLab-t visualiser
- w.iLab-t analyser
- log files

Additionally, you can connect directly to your personal mysql database on the wilab server using a mysql library in your own scripts, or directly through the phpmyadmin web interface.

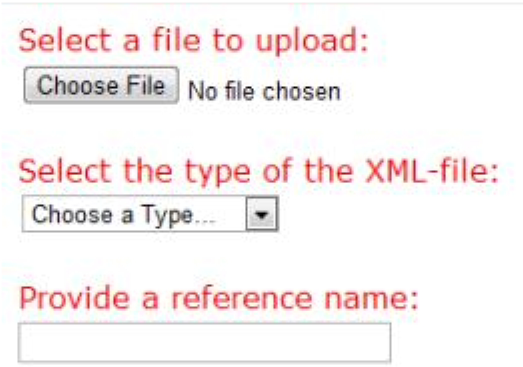
The main difference between the two tools on the w.iLab-t website is that the visualiser provides a spatial view of the testbed, with typically per node characteristics, while the analyser provides different graphs and is most often used for a temporal view of the experiment (line chart, scatterplots and barcharts are supported). Both tools can be used during or after an experiment.

We will now customize two configuration files for both w.iLab-t tools and upload them to our **toolbox** page. Download the zip file with both xml scripts here, and open them in your favorite xml/text editor. For both xml files 3 parameters have to be changed, preferably with a *replace all* command:

- **___USERNAME___**: your w-iLab.t database name (see the info page if needed)
- **___PASSWORD___**: your w-iLab.t password
- **___MSGID___**: the ID of the message class you added in the Job page (see the "Create the sensor job" section if needed)

If you would not like to edit the parameters now, you can upload the files as is, and the respective tools will ask you to substitute them at run time.

Go to your toolbox page and scroll to the upload box at the bottom:



Click **Choose file** to navigate to the xml configuration files and select the correct type (Analyser or Visualiser, corresponding with the first word of the file name) in the box below. Choose a name for your script, for this tutorial the file name will be used. Scroll down and press the **upload** button.

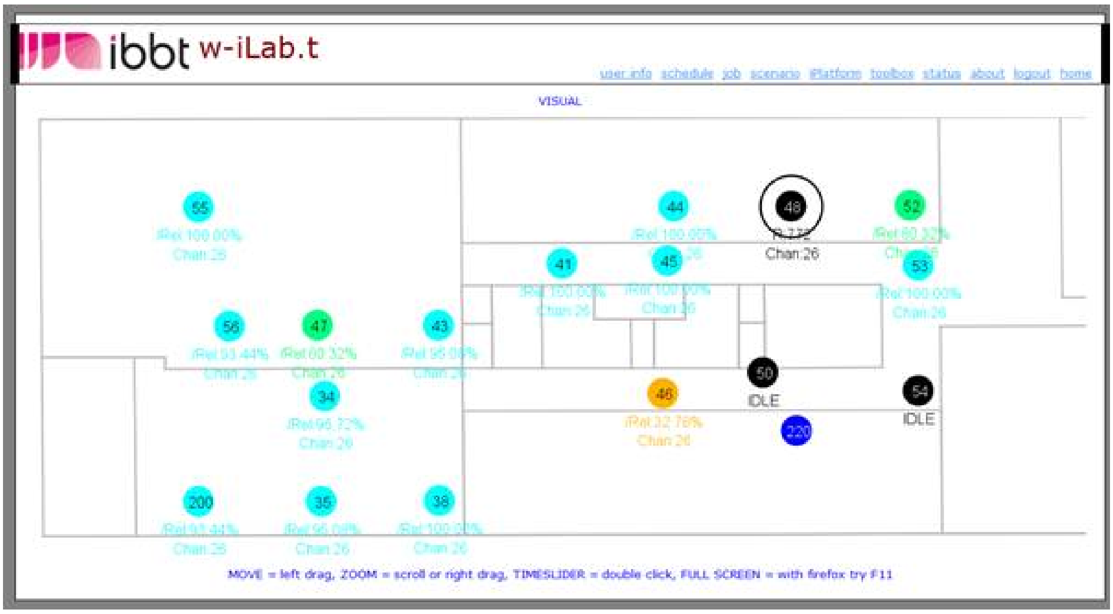
After uploading the configuration files it is time to familiarize ourselves with the two w-iLab.t analytic tools.

w.iLab-t visualiser

The toolbox page gives access to the visualiser and analyser, where you can select an XML configuration file for the respective tools. These files can be edited or created by yourself to suit your experiment.

Select the **Visualiser_BenchReliability** XML file and click Start Visual to start the Java applet that will visualise your experiment. Now fill in your database user name (not email) and password, together with the experiment ID you want to visualise.

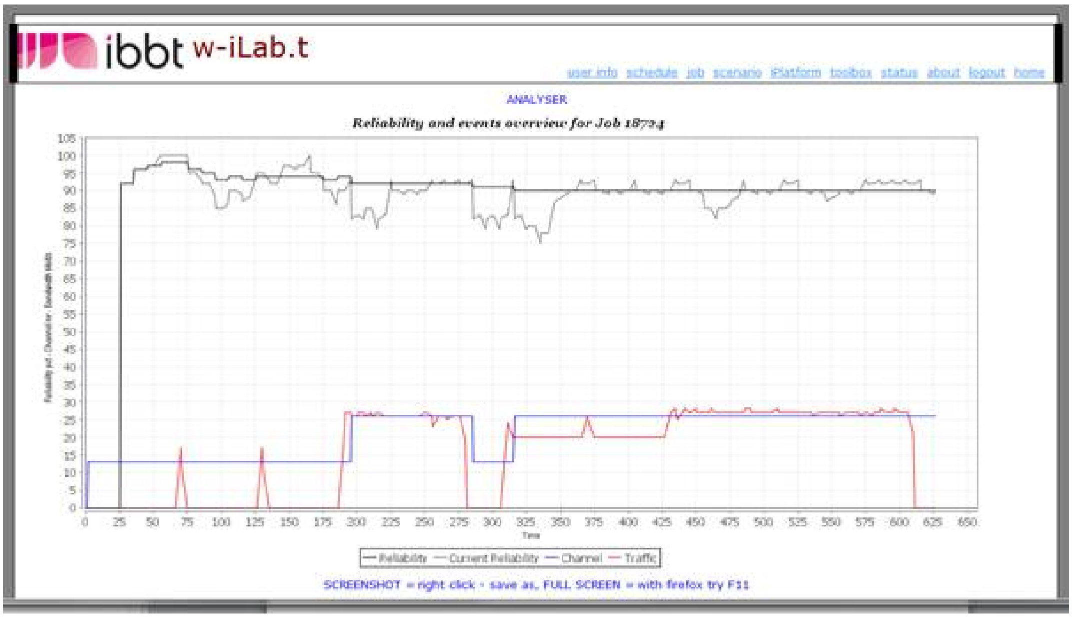
Once the applet has finished loading, you should now see a colored circle with the sensor node id for every active sensor node in your experiment. After some time, every node should log some info to the database and the circles should change color depending on their individual reliability. The reliability and chosen channel is also included in the text below each node. Nodes that receive messages are indicated with an additional circle, with the total count of received messages in text under the node.



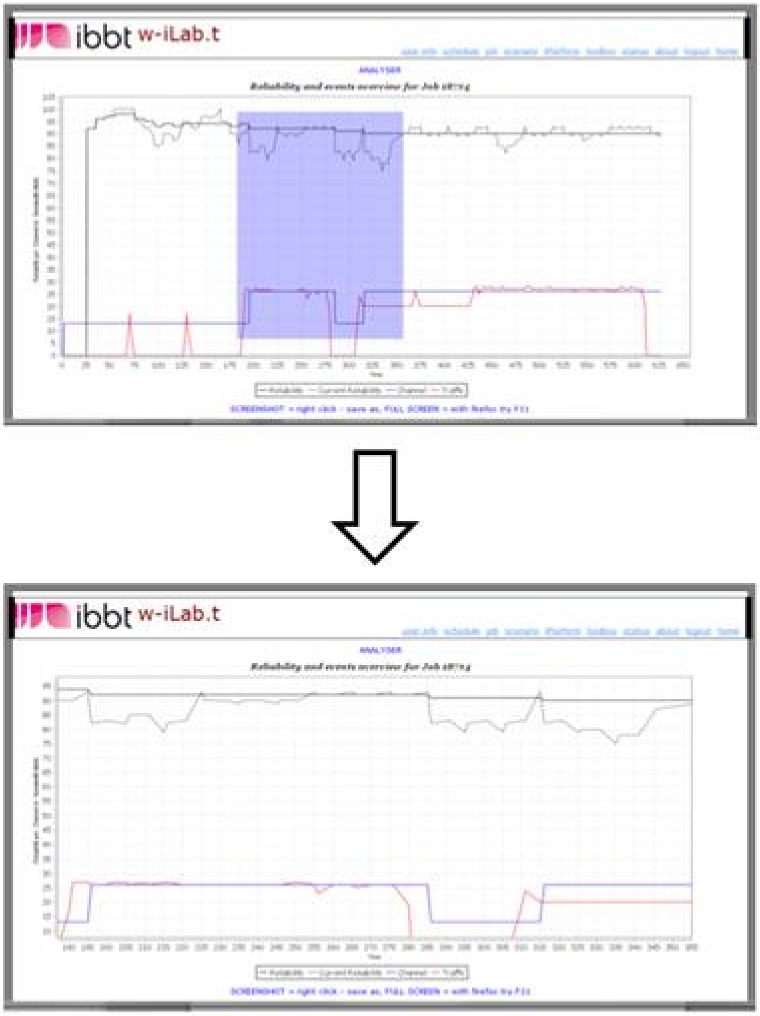
w.iLab-t analyser

The analyser is also located on the toolbox page, where you should select the **Analyser_BenchReliability** script. This script will present a line chart with different metrics recorded during the experiment. Fill in your database user name (not email) and password, together with the experiment ID you want to visualise when prompted. Please be patient for the graph to load, and you should get a visualisation of the following metrics:

- Cumulative sensor network reliability
- 30 second window average sensor network reliability
- Cumulative wifi traffic
- Zigbee channel of the sensor nodes



The resulting graph will allow you to more thoroughly analyse the performed experiment. The applet supports zooming, just by dragging a box over the area of interest. Be dragging the mouse in a northwest direction the entire x and y range is plotted again.



Additionally, the applet provides advanced customization and export functions using the right click context menu, which can be useful to import the post processed results in your favorite charting program.

log files

In case your experiment fails, has strange results or you want to look under the hood, you can access all the raw log data that your scripts produce in the log directory of the wilabfs server. These log files should be created by yourself if you write your own scripts, but some example convenience functions are provided (for a more extensive list, take a look at the source of the "variables" script included in this tutorial)

- Log directory for the specific node:
  `find ../ -maxdepth 1 -type l -exec ls -l {} \; | cut -f 2 -d '>'`
- Log directory for the total experiment:
  JOBDIR=`dirname $NODEDIR`

This leads to a hierarchical directory structure inside your iPlatform directory "log" that allows you to store all the desired logs per experiment and per node

Attachment	Size
sensor_code.zip	33.81 KB
iplatform_scripts.zip	5.02 KB
analyser_visualiser_config.zip	3.28 KB

Hardware and testbed lay-out

Office environment testbed

The office environment testbed is deployed in the IBBT office spaces, meeting rooms, student lab rooms, corridors, etc. It consists of 200 wireless node locations, each equipped with one or multiple (heterogeneous) wireless sensor nodes, as well as 200 x two IEEE 802.11a/b/g WLAN interfaces.

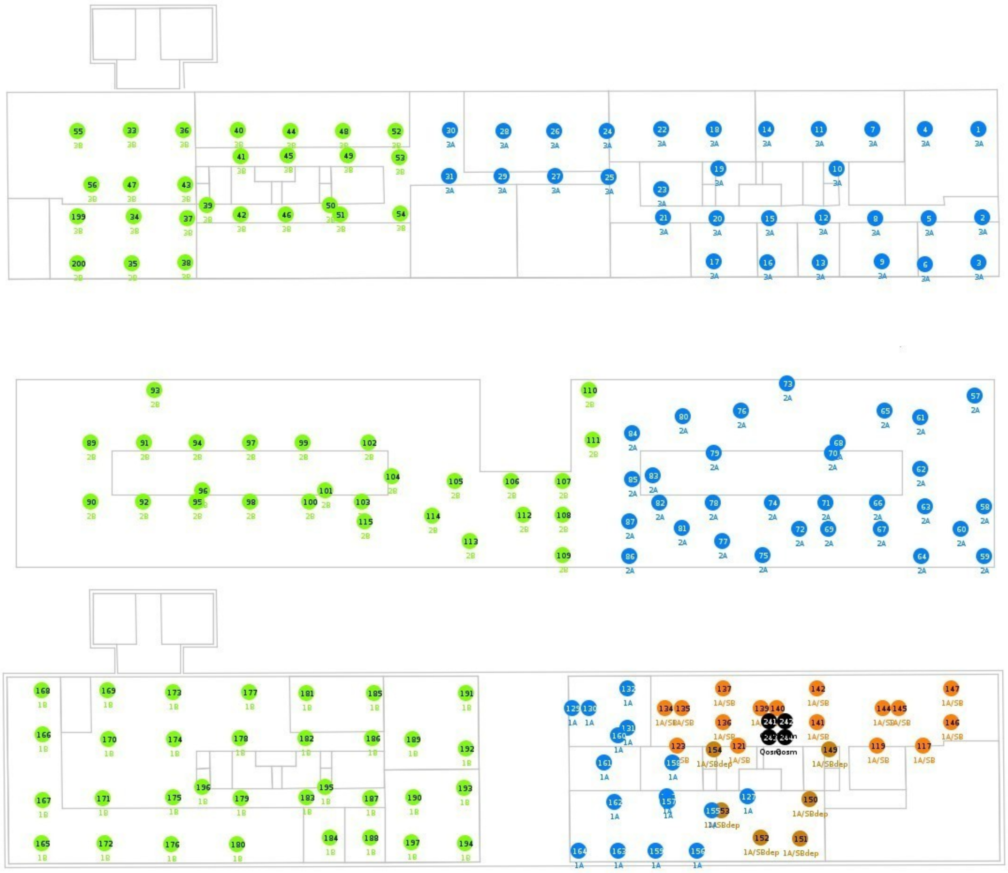
Important note: a Wi-Fi usage policy is in place at the w-iLab.t Office location. In the largest part of the building, use of Wi-Fi interfaces is not allowed on weekdays from 6AM to 8PM CET. Use of Wi-Fi in Sandbox zone at the first floor is always allowed.

Map & Zones

A live map of the nodes in the office environment is available here.

As every floor has been split into two zones A and B the nodes are colored blue and green. There are two special zones on the first floor; a sandbox zone (the time constraints related to the WiFi use between 8pm and 6am are not applicable here) in orange and black zone of four nodes that are installed in shielded boxes.

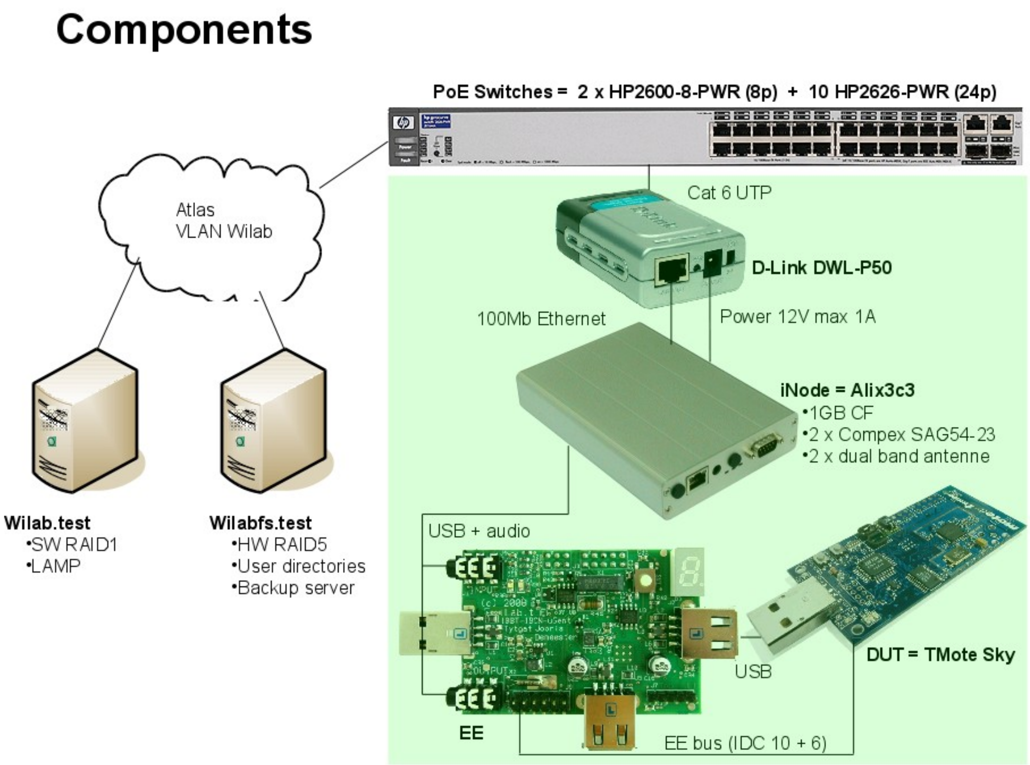
The screen shot below shows what to expect:



Attachment	Size
testbed-office-map.jpg	137.41 KB

Topology

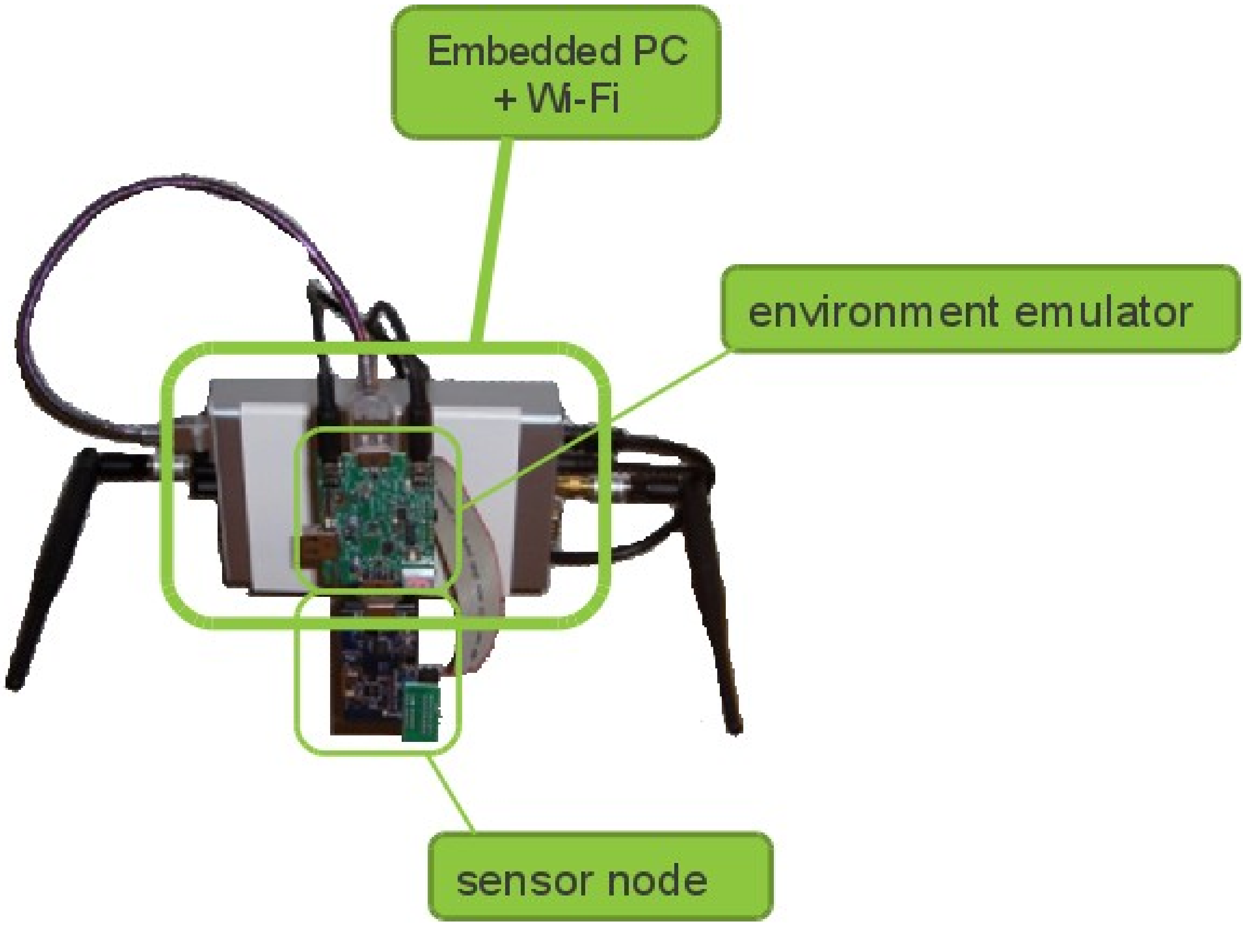
Around 200 identical configurations are deployed at the Zuiderpoort building. Every configuration exists out of an embedded PC with WiFi based on the Alix, a power over Ethernet splitter to power the PC, an environment emulator and TMote Sky sensor node. The power over Ethernet splitters are connected to the in total 12 power over Ethernet switches. The switches are then connected to the servers of the testbed. There is one fileserver to store the contents of the experiments and one LAMP server where the Testbed logic is implemented on. This picture shows global view on the components and the interconnections.



Attachment	Size
topology.jpg	119.66 KB

Configuration

On every of the 200 locations you can find a configuration like this:

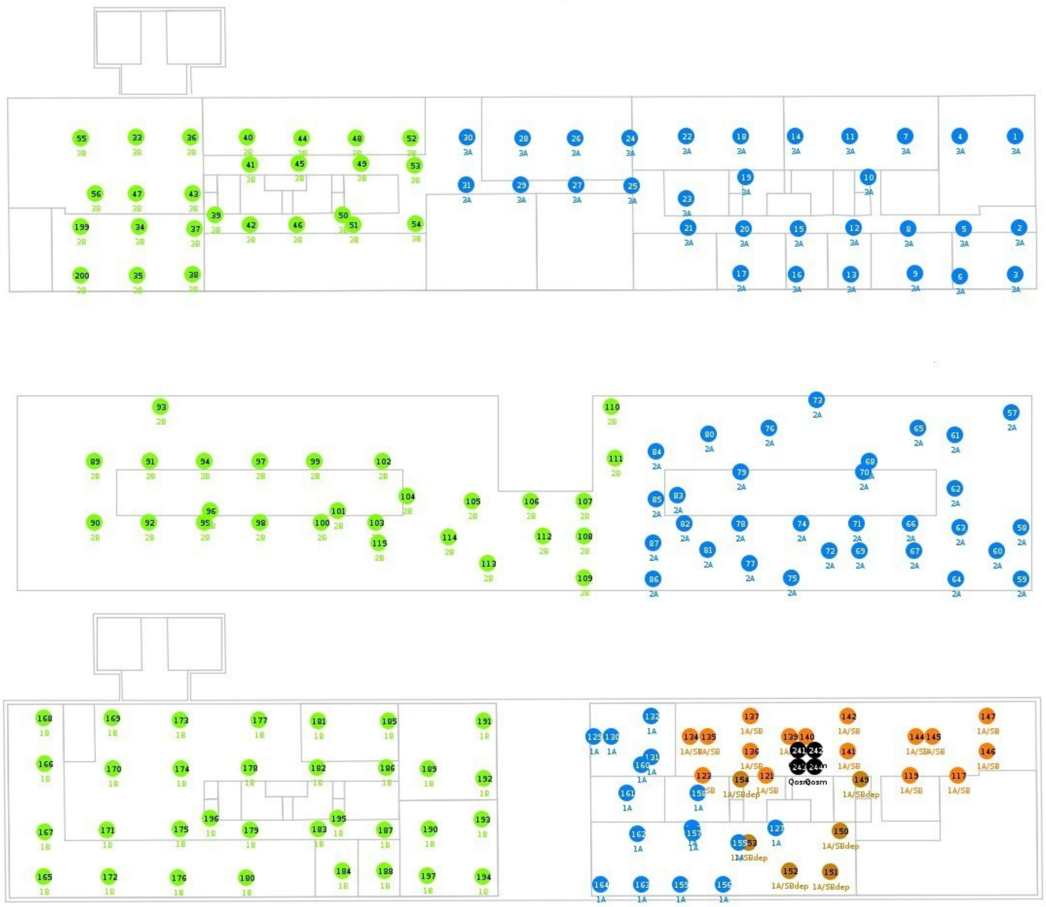


It consists out of an alix3c3 motherboard with schematics. This motherboard is based on a 500 MHz AMD Geode LX800 CPU with 256 MB DDR DRAM and the CS5536 chipset. The alix is equipped with:

- 1 x 1 GB CompactFlash card, SMI 2232 controller, SLC flash. Supports UDMA.
- 2 x Compex WLM54SAG 200mW AR5006XS 802.11a/b/g 54/108 Mbps miniPCI wireless card (only main connector is used)
- 2 x Pigtail cable, I-PEX to SMA female reverse connector, 15 cm cable
- 2 x Dual band antenna’s with specs
- On the first USB connector the Environment Emulator and the TMoteSky Sensor node are connected in cascade. The soundblaster input and output of the Alix are connected to the Environment Emulator.

User Quota

As shown in the picture below, the testbed is divided into several zones. The user quota is directly related to these zones.



Lets say for example that user *John Doe* has a quota of 120 minutes. There are several ways for *John* to consume his quota :

- Run an experiment of 120 minutes on 1 zone (1A,1B,2A,2B,3A or 3B)
- Run an experiment of 60 minutes on 2 zones combined (floor 1, floor 2 or floor 3)
- Run an experiment of 20 minutes on the entire testbed.

John can offcourse decide to run two different 60 minute jobs on e.g. zone 1A and zone 2B).

After the experiments are finished, *John*"s user quota will be reset to 120.

Sensornode: TMote Sky

TMote Sky sensor nodes consist of an TI MSP430F1611 processor running at maximum 8MHz, 10KB of RAM, 1Mbit of Flash memory and a Chipcon CC2420 radio operating at 2.4GHz with an indoor range of approximately 100 meters. Each node includes sensors for light, temperature, and humidity.

- TmoteSky data sheet
- Msp430f1611 data sheet
- Msp430f1611 Family guide
- CC2420 datasheet

Attachment	Size
tmote-sky-datasheet.pdf	1.17 MB
msp430f1611.pdf	1.34 MB
msp430f1611FamilyGuide.pdf	1.51 MB
CC2420_Data_Sheet_1_4.pdf	995.94 KB

Environment Emulator

The EnvEmu opens up a lot of real-life test cases, of which the major ones are described below.

- The EE can disconnect the USB power from the DUT, and power it with its own regulating voltage source. This enables the EE to emulate the real behavior of a battery depleting, energy harvesting power sources, failing of the mote, ...
- The current used by the DUT can be measured with a sample frequency of 10kHz. Using this approach, it is very easy to determine the exact power consumption when it is running the current application, protocol, ...
- The EE has some General Purpose digital Input / Output pins connected to the DUT. This allows for real-life, real- time digital sensor / actuator emulation. These pins can also be used to tag specific states generated by the DUT to ease the analysis and classification of the power consumption.
- Some analogue input/output pins are also connected to the DUT, making the emulation of real-life, real-time analogue sensors/actuators possible.
- Audio input/output signals can be injected and extracted from the embedded PC to the DUT making all sorts of audio over sensor network experiments very easy to conduct.
- The EE can be used as an energy harvester.The EE can disconnect the USB power of the device under test (DUT) and can alternatively power the DUT via its battery interface with a variable voltage supply. By measuring the current at a high sample rate and use this information to control the voltage supply we can build a controlled loop feedback mechanism. To emulate the real behavior of an energy harvester we implemented the law of Coulomb as a feedback mechanism. Furthermore as we are at all times aware of the consumed current by the DUT (sample rate of 4 kHz) and the tuned voltage we can determine the exact power consumption (more details below).

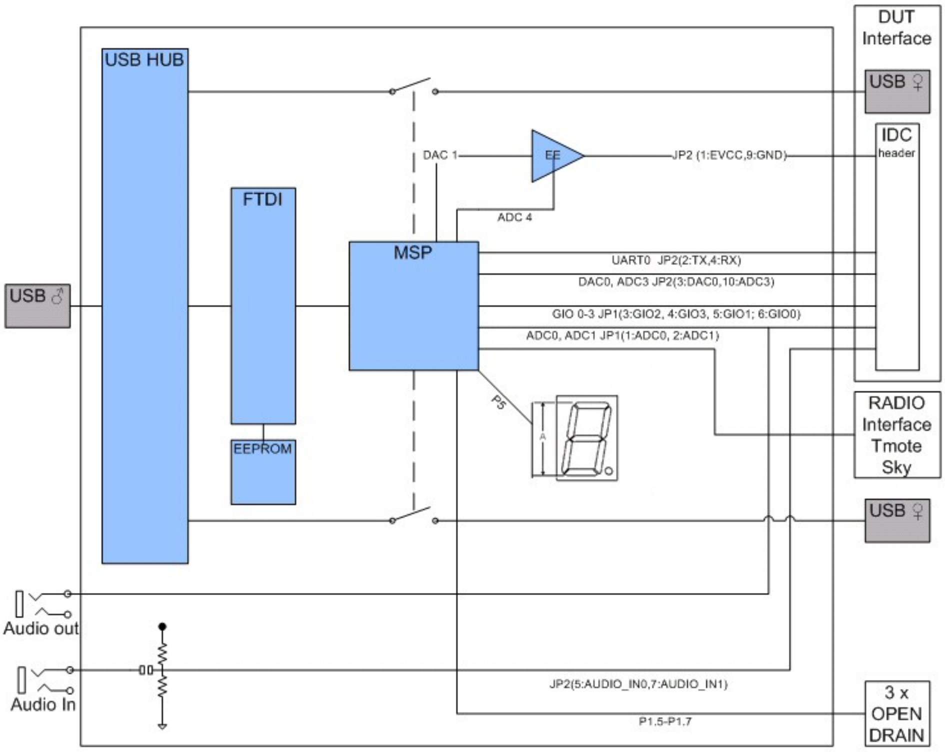
All of this can be prepared before running an experiment or can be adjusted real time during the experiment by using the scenarios tab.

EE related publications

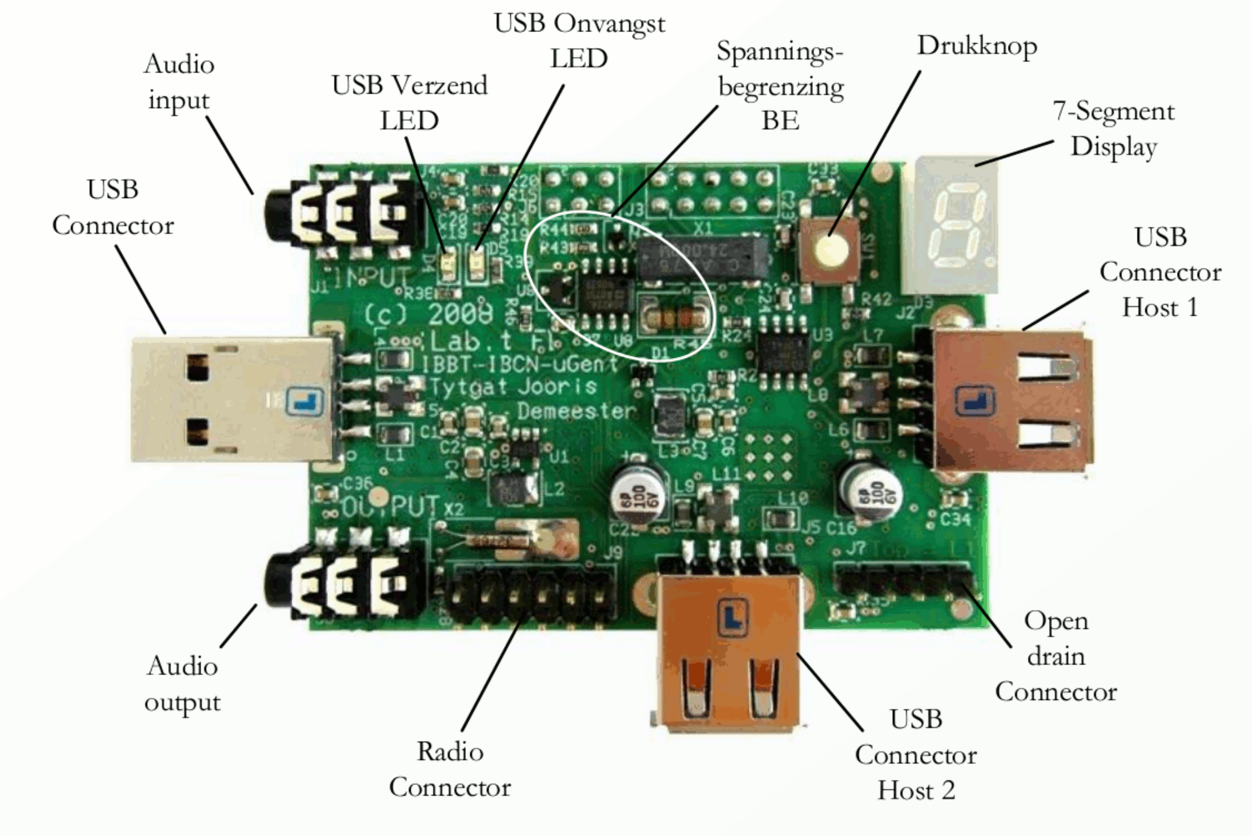
- I. Moerman B. Jooris P. De Mil T. Allemeersch L. Tytgat P. Demeester, "WiLab: a large-scale real-life wireless test environment at IBBT", published in Proceedings of DSP Valley seminar "Sensor driven state-of-the-art Mechatronics", Anderlecht, Belgium, 20 February 2008
- L. Tytgat B. Jooris P. De Mil B. Latre I. Moerman P. Demeester, "Demo abstract: WiLab, a real-life wireless sensor testbed with environment emulation", published in European conference on Wireless Sensor Networks, EWSN adjunct poster proceedings (EWSN), Cork, Ireland, 11-13 February 2009

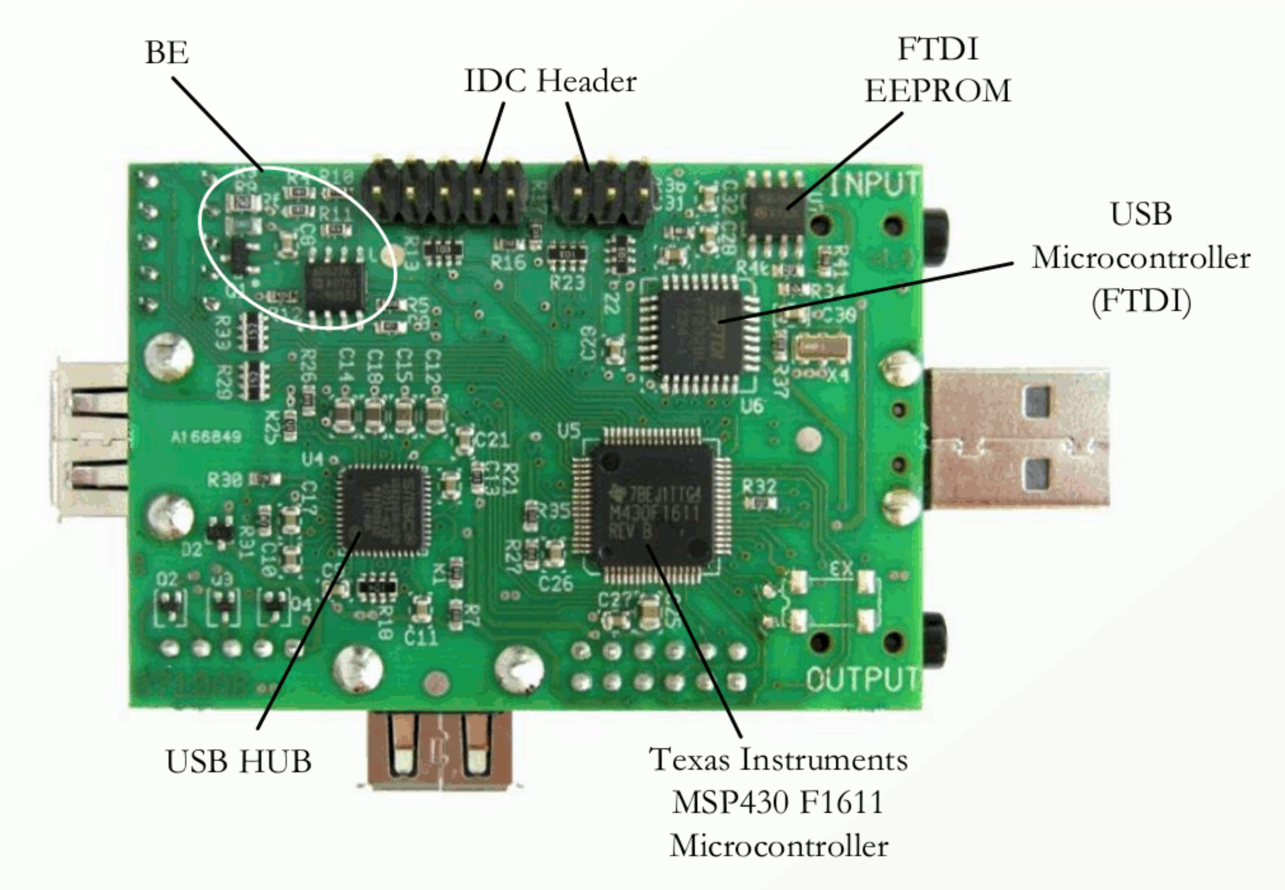
- Pieter De Mil, Bart Jooris, Lieven Tytgat, et al., “Design and Implementation of a Generic Energy-Harvesting Framework Applied to the Evaluation of a Large-Scale Electronic Shelf-Labeling Wireless Sensor Network,” EURASIP Journal on Wireless Communications and Networking, vol. 2010, Article ID 343690, 12 pages, 2010. doi:10.1155/2010/343690

If you take a closer look to the block diagram of the EnvEmu you will notice that is based on TMote Sky. We stripped down the TMote Sky by removing the sensors, leds, buttons, radio, I2C msp flash lock circuit and also flash. So basically we stripped it down to ftdi, msp430, IDC header and radio interface. We added a 3ports USB hub, 2 USB switches that are connected to slave female USB connectors, 7 segment display, battery emulator and audio jacks. See the block diagram below.



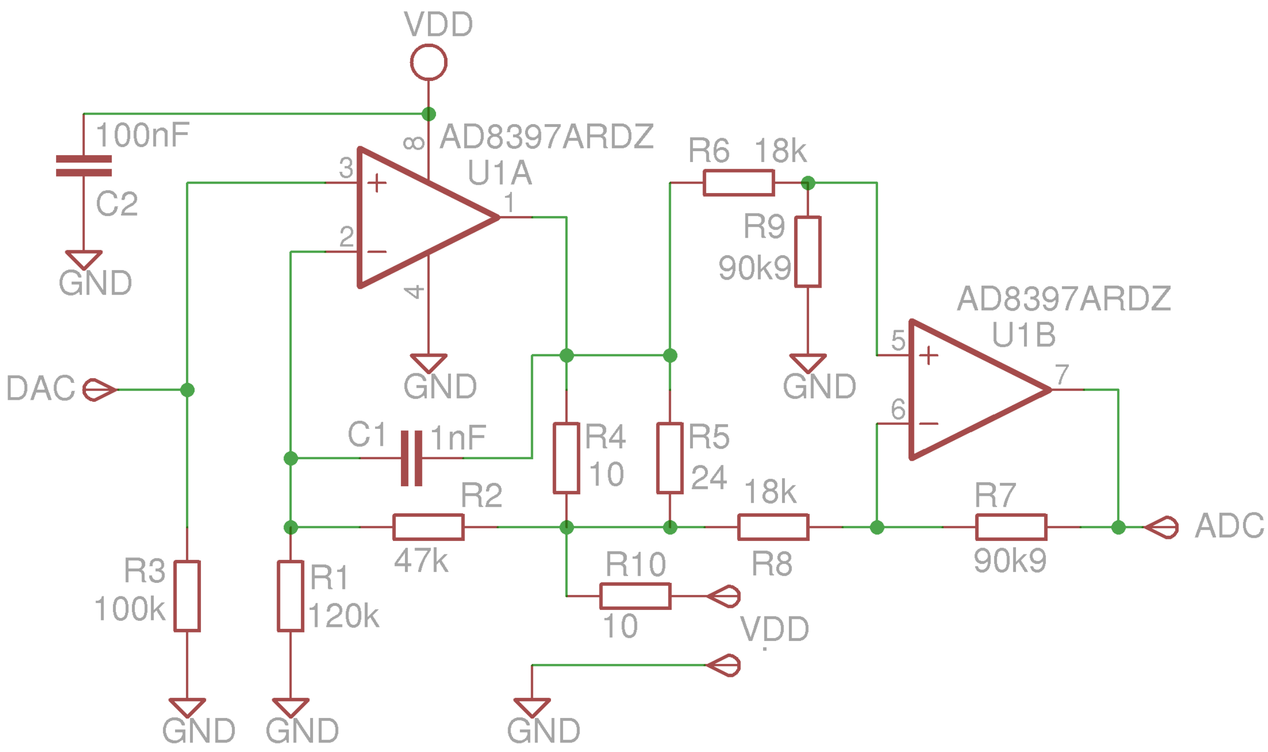
Both the front an the back view of the board can be find here:





The electronic diagram can be found [here]

Here you can find the detailed electronic circuit of the battery emulator.

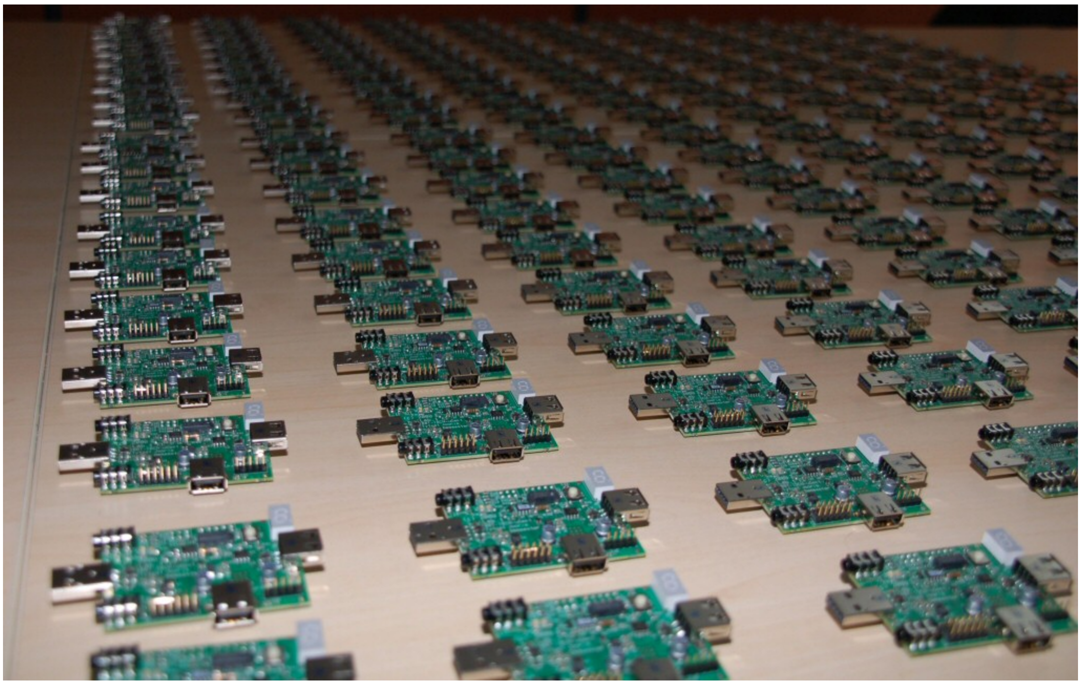


A new sensor board was created, which is basically a stripped version of the TMote Sky and we called it the Environment Emulator (EE). On the EE we connected VDD (schematic) to the USB power of the board. The ADC and DAC lines (schematic) are connected to DAC1 and ADC4 of an MSP430. The DUT lines are then interfaced to the battery interface of the device under test. Implementing just the schematics as it is and connect it to an existing tmote or telosb gives the same functionality. The main component in the schematic is U1 which is a rail-to-rail, high output current amplifier. U1a is used to implement a voltage follower and maps the 2.5 V coming from the DAC (maximum output of DAC1 of the MSP430) to 3.5V (the maximum supply voltage of the DUT). Standard op amp schematics are not able to drive high capacitive loads . C1 and R10 were added in the second version of the EE and are used as inner and outer loop compensations for a better response when driving high capacitive loads. 10uF is a typical input capacitor of an sensor node and is much higher than what an opamp (typical 200pF) can drive without compensations. U1b is used to implement a differential amplifier and maps a current of 70mA trough R4 and R5 to 2.5V on the input of the ADC (the maximum input voltage of ADC4 of the MSP430). To implement the law of Coulomb a tinyos application was developed where we implemented an user event 'stream' with these parameters; start value which is the DAC value at t0, virtual capacitor and the harvester itself. When executed a continuous sampler will start on ADC4 with a sample rate of 250us. On every sampler buffer done event the next Dac value will be calculated as follows (sampler buffer size is 50):

DacValue(t+1) = DacValue(t) + SUM (harvester - samplerBuffer(t)[i]) / virtualCapacitor.

The unit of the virtualCapacitor is 5uF and is derived from the interSampleDelay/numberOfSamplesPerBuffer. For example; for a battery emulation of 2.4V/3000mAh, we could define a full battery with initial voltage of 2824 (2.400 V) and a harvester which is equal to zero and a capacitor of 4500 F or the virtualCapacitor equal to

1.25 x 60 (min) x 60 (s) x 200k. For solar cell emulation, we could define a capacitor with initial voltage of 0 V and a harvester which is equal to 1mA (59) and a capacitor of 1F. On the DUT the first thing to do is to check if there is enough energy and if there is the radio can be enabled. The other way around would put the sensornode in an endless reboot sequence. For now with a interSampleDelay of 250uS will get a reaction time of 50 samples x 250 us which makes 12,5 ms. We will speed this up in the near future.



Zwijnaarde testbed (Pseudo shielded, open environment)

Important information on the Zwijnaarde testbed: please note that we are currently in the process of completing the Zwijnaarde rollout [current status, mid September 2011: we are now installing Ethernet and power cables; our hardware is in stock, the software is developed, tested, and ready to be deployed). The Zwijnaarde testbed will soon be accessible in a similar way as the office testbed.

The testbed will at latest be available to external users when new partners join the CREW project as part of the open call: this is January 2011.

Information on the components and functionality that will be available can already be obtained from the sections below.

Configuration

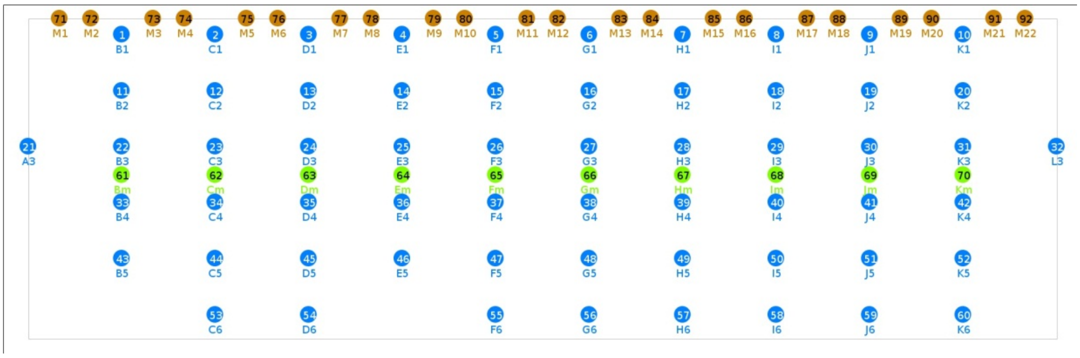
The fixed nodes are equiped with:

- ZOTAC NM10-A-E
  - specs

- KINGSTON ValueRam 4GB DDR2 800MHz PC2-6400 CL6
- SEAGATE Momentus 7200.4 160GB (2.5", SATA, 7200RPM, 16MB)
- IN-WIN BQ656 (mini-ITX, 160W power supply)
- 2 Wifi
  - Sparklan WPEA-110N/E/11n mini PCIe 2T2R chipset: AR9280
- 2 x 2 Attenuator 20 dB
  - Telegärtner J01156R0041; R-SMA (m-f) 0-6GHz 50Ohm 2W
- Environment Emulator
  - Eev2 (IBBT design)
- Sensornode
  - RM090 (see page on the RM090)
    - Msp430f5437 (18MHz-256k Flash- 16k Ram)
    - CC2520

Map

The nodes in the testbed are mounted in 66m by 20.5m open room in a grid configuration with dx = 6 meter and dy=3.6 meter. The 60 installed nodes are represented by the blue locations on the picture. The green and orange location will be used to connect the mobile nodes, the SDRs (USRP, WARP,...) and the cameras.

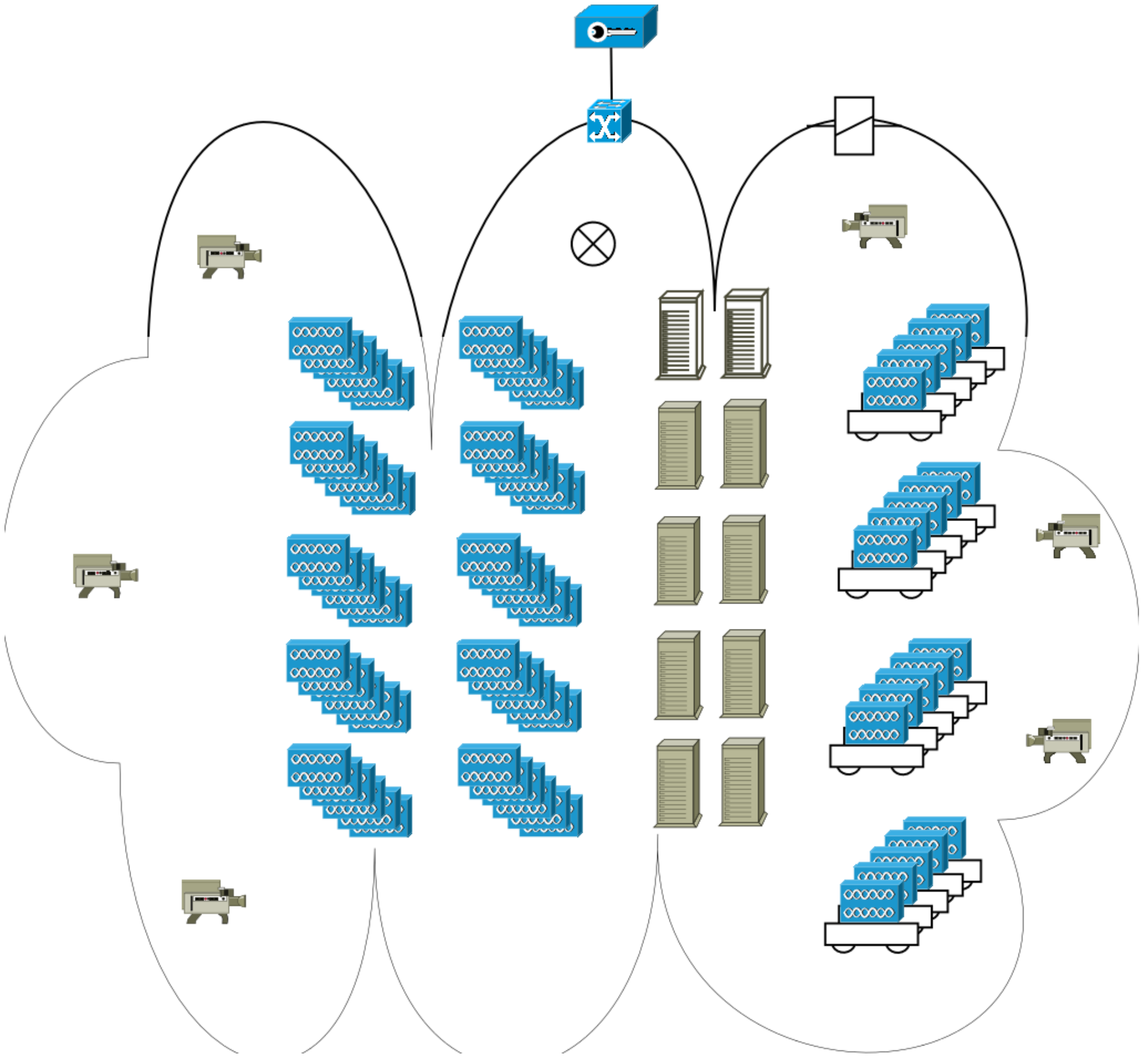


A picture of the room can be found here. In the front you see node on location J3, in the same line after the pipe you can see the locations J4,... As one can notice we have installed some copper tape on the non metalic pipes to minimize the interference with the Clean Room (which is located under the testbed) itself.

Attachment	Size
Wilab2_CR_shielding4.JPG	305.96 KB

Topology

The hardware components and there interconnections



The testbed is accessible via openvpn gateway (the key symbol on top) which gives the experimenter access to the OMF based lab. The two components on the edge of the cloud imply that all the components inside the cloud are connected to it. The switch symbol represent a stack of switches to interconnect all the devices on a Gigabit LAN where possible. The relay symbol represents the PDUs (power distribution units) to remotely get an idea on the power consumption and to switch off and on the devices. The lights can also be switched on and off remotely. The two white servers are ESXI servers. On top of them 5 VMs are running with OMF related managment software and backup services. The functionality of the VMs are AM (aggregate manager), EC (experiment controller) and a XMPP (Extensible Messaging and Presence Protocol) server. Each ESXI server frequently takes snapshots of all the running non backup VMs and copies them to the VM backup server on the other ESXI server. In case of a hardware failure we should be able to fast recover. Further there are 8 servers with identical hardware of the ESXI servers (except for the hard disks 160G instead of 2 times 1T) available for the experiments. The 60 dual mode access points symbols at the left side of the servers represent the described configuration. Some of them will be extended with OR IMEC sensing engine, WiSpy or SDR (Warp or USRP UN210 or E100) The 20 dual mode access point symbols on top of a robot platform on the right side of the servers represent the mobile nodes which can be expected end of 2011. Last but not least we have 6 axis cameras to verify the mobile nodes locations or to debug the leds on the sensornodes.

Some detailed information on the hardware can be found here:

- The 10 servers with **dual Intel®Xeon®Processor 5600 Series** and 12GB RAM are implemented as 5 1U twin machines with specs
- The 13 PDUs which can drive and measure 104 230VAC outlets are desribed in detail here.
- Two types of axis cameras will be installed;
  - 6 AXIS 212 PTZ Webcam
  - 2 AXIS 214 PTZ
- The installed switches are of the type
  - HP procurve 2510G
  - HP procurve 2610
  - HP procurve 2626-PWR (to power alix boards (mobility mgt) and the AXIS 212 PTZ.
- Siemens LOGO! 230RCE will be used to control the lights.

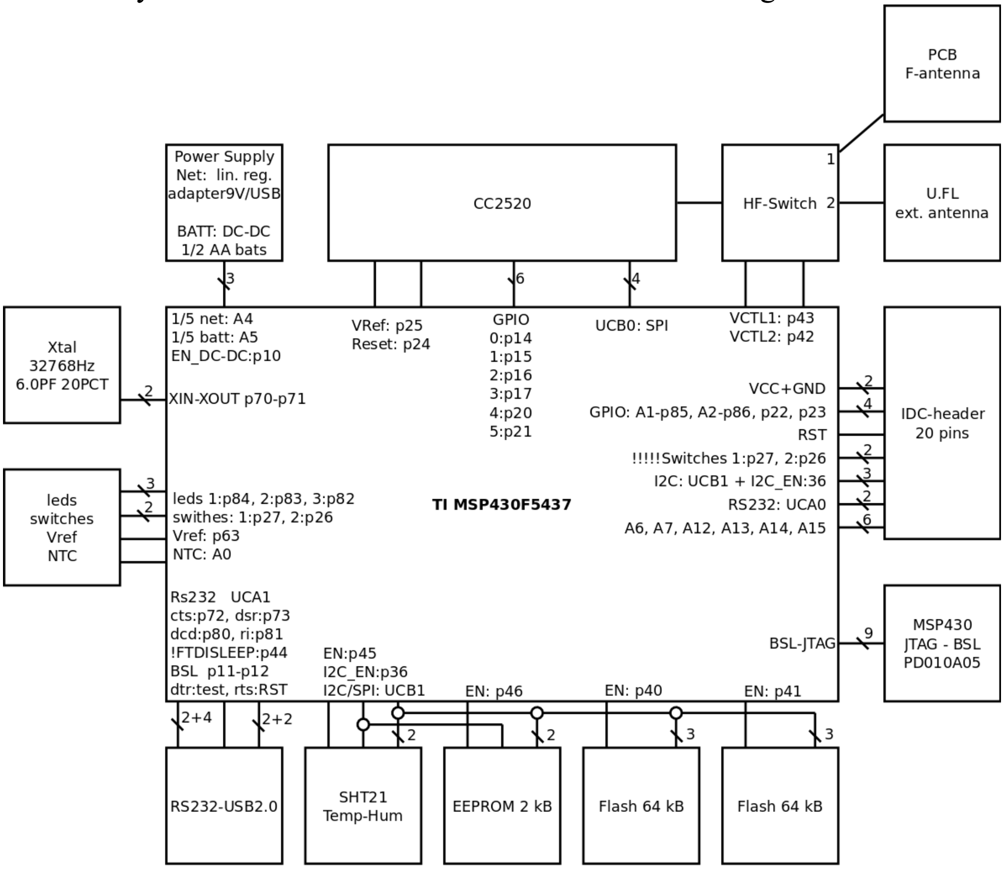
Sensornode: RM090

The RM090 sensor node is a joint design by Rmoni and IBBT. The major reason for IBBT to create a new sensor node was of the lack of sufficient resources on the TMote Sky (see table). Especially the 48k of flash on the TMote was the enabler to upgrade the testbed.

Feature	**Imote (2003)**	**Mica2 (2003)**	**MicaZ (2004)**	**Telos (2005)**	**IBBT-Rmoni RM090 (2010)**
CPU type @[MHz]	32bit ARM @12	8bit Atmel @8	8bit Atmel @8	16bit TI @8	16bit TI @18MHz
SRAM [kB]	64	4	4	10	16
FLASH [kB]	512	128 + 512	128 + 512	48 KB / 1024 KB	256 KB/128 KB + 16KB eeprom
Radio	BT	300-900MHz	802.15.4	802.15.4	802.15.4 2.4GHz band
Bandwidth [kb/s]	720	15	250	250	250
Carrier Sense/ Rx/Tx Current [mA]	15 / 24 / 24	08/10/27	08/20/18	01/20/18	18.5 /18.5 / 33.6@4dBm 25.8@0dBm
sleep current [uA]	01/01/50	19	27	6	2
OS support	TinyOS	TinyOS	TinyOS	TinyOS	TinyOS or IDRA

We tried to be as compliant as possible with the TMote Sky with the second generation chips of TI the msp430f5437 and the CC2520. The IDC headers, the 3 leds, the USB interface are as much compliant as possible.

Here you can find a block diagram of the RM090:



Attachment **Size**
ilabnode.png 113.9 KB

Cognitive components

At the pseudo-shielded Zwijnaarde location, a set of cognitive networking platforms are available. They can be remotely accessed over the internet.

Again, it is up to the experimenters to decide how to use the hardware that is made available. Signals may only be transmitted in the 2.4 GHz and 5 GHz ISM band due to license restrictions.

We are in the process of making the cognitive components available over the Internet. The integration will at latest be completed when new partners join the CREW consortium as part of the open call. Please check back later updated information.

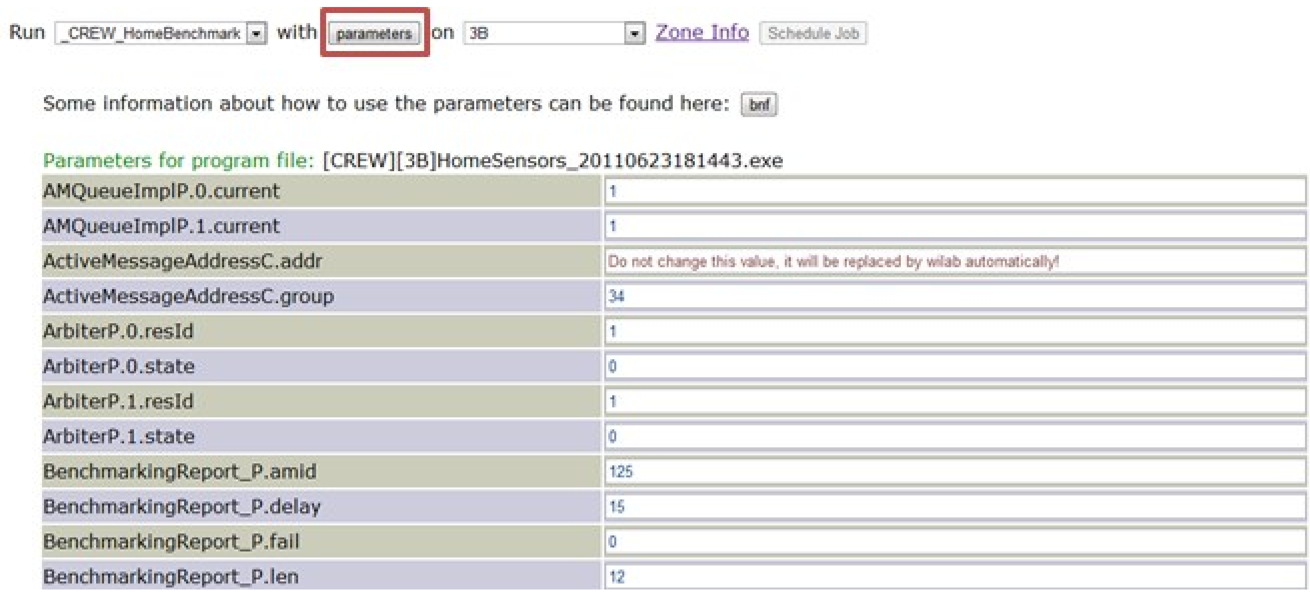
Using the hardware: tools, interfaces, services

Change sensor code variables at schedule time with BNF syntax

When you want to schedule a job on the w-iLab.t testbed with sensor code included, you can use our dynamic parameter utility.

This tool allows you to redefine global variables in your nesc code (TinyOS or Contiki) at schedule time using our BNF syntax. So you are not limited to just substituting one value, but have a powerful toolset to scan parameter ranges and schedule multiple experiments at once. More information on the Backus-Naur Form available on Wikipedia

To access this utility, click the **parameters** button on the w-iLab.t scheduling page.



In the table that appears, all global variables in the WSN images associated with the selected job are displayed. You can simply change the compiled value for each displayed variable, or you could use the more elaborate BNF syntax to create more powerful expressions that grant the following possibilities:

- schedule multiple experiments by using the pipe "|" character
- change variable for a specific node or a list of nodes by using a comma ","
- set a variable for a range of nodes using ".."

The available BNF syntax to utilize this functionality is given below, together with some examples:

<experiment>::=<paramlist>|<paramlist>"|"<experiment>

<paramlist>::=<paramsublist>|<paramsublist>";"<paramlist>

<paramsublist>::=<nodelist>":"<paramvalue>|<paramvalue>

<nodelist>::=<nodeid>|<nodeid>".."<nodeid><nodelist>","<nodelist>

<paramvalue>::=<number>

Example:
For the parameter Q we define 4 experiments where we assign the values A, B, C and D for all the nodes.
The expression for Q will be: A|B|C|D

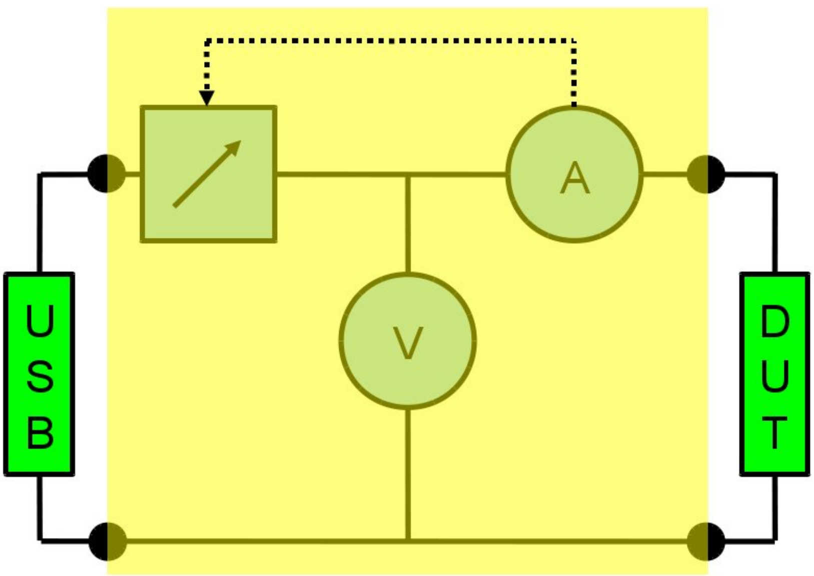
A more elaborate example:
For the parameter P we can define 3 experiments X|Y|Z. During experiment X we assign the value A for the nodes 1 to 10, the value B for the nodes 12,15 and 18 and the value C for all the other nodes. For experiment Y, the value A will be assigned to the parameter P. During the last experiment Z we will assign the value C for the nodes 10 to 20 and the value B for all the other nodes.
The expression for P will be: 1..10:A;12,15,18:B;C|A|10..20:C;B

When we now would schedule this job with parameters Q and P redefined, (3x4) 12 experiments will be generated!

NOTE: Be careful while writing expressions (NO SPACES,...) as there is no syntax check! Values (here A, B, C and D) are parsed as numbers, make sure that your data structure accepts your new input.

Power measurements on the sensornode

To measure the real time power consumption of any electrical device you will need a power source which is suited for the device under test, a voltage and current meter.



These three components are integrated in the EE.

Please have look at the internals of the EE at... The adjustable power source (aka battery emulator) is controlled by ADC1 of msp430 on the EE board. The maximum value of the ADC is amplified to the maximum of the msp430 family which is 3.60V. The second component, the voltage meter, comes for free as we can perfectly control the power source. The current meter is implemented by putting a small resistor in serie with the device under test. By amplifying and measuring of the voltage over this serie resistor we get a good idea of the current at ADC4. As we can sample the ADC4 at quite a high rate (10kHz) we get a real time power measurement which can show the power consumption at 802.4.15 packet level.

Every sensor node on the testbed has an accompaning EE so you can also get an idea of the power consumption at network level.

To execute a power measurment on the testbed you will need to execute the following events (see events):

- disable the USB power (gpioPinStatus ; disconnect the USB 5V line)
- enable the battery emulator (streamer event)
- start the current measurments (sampler event)

The order of execution is important! If you mix up the first two then an extra current of 5mA from an iniatialized USB chip will also be measured. If you don't execute the first event the node will partialy be powered by the USB and the battery emulator.

You can use the [w-iLab.t] visualiser to show the results.

Much more details about these components can be found on the scenario tab when enabling streamer (battery emulator) and sampler events.

RadioPerf tool

This contains more information on a **TinyOS** program called **RadioPerf**, which we developed at **IBBT**.

Introduction

RadioPerf functions both as a packet generator and a basic packet analyzer, making it very useful to test transmission performances. It consists of two different parts:

- The first part is a java GUI, running on the host computer, which sends control messages to the nodes and receives reports from the nodes.
- The second part is a TinyOS program running on the nodes itself. This tiny program receives the control messages from the host computer, and send back reports.

Download the source code

The source code is available here .

Instructions on how to compile all parts of the application is described below. We will release an easier-to-install version of this application in the near future.

Compiling the TinyOS sensor code

If you don't have TinyOS installed on your system, please follow the steps described on this page .

Attach a sensor node to your PC, step into the RadioPerf directory and execute :

make telosb install,1

This command will program the sensor node with the RadioPerf source code and give it ID 1.

More info on programming TinyOS compatible sensor nodes can be found on http://www.tinyos.net . The tutorials contain everything you need to know about programming TinyOS sensor nodes.

Compiling the Java GUI

Go to the java_Netbeans-5.5 directory and execute :

RadioPerf/java_Netbeans-5.5$ ***../commandline/genMakefile ../RadioPerfMessages.h src/***

You should now see a "Makefile" in the current directory.

If you encounter errors like:

- src/RadioPerfView.java:22: package org.jCharts.axisChart does not exist

  import org.jCharts.axisChart.ScatterPlotAxisChart;

- src/RadioPerfApp.java:140: package org.jdesktop.layout does not exist

  org.jdesktop.layout.GroupLayout layout = new org.jdesktop.layout.GroupLayout(getContentPane());

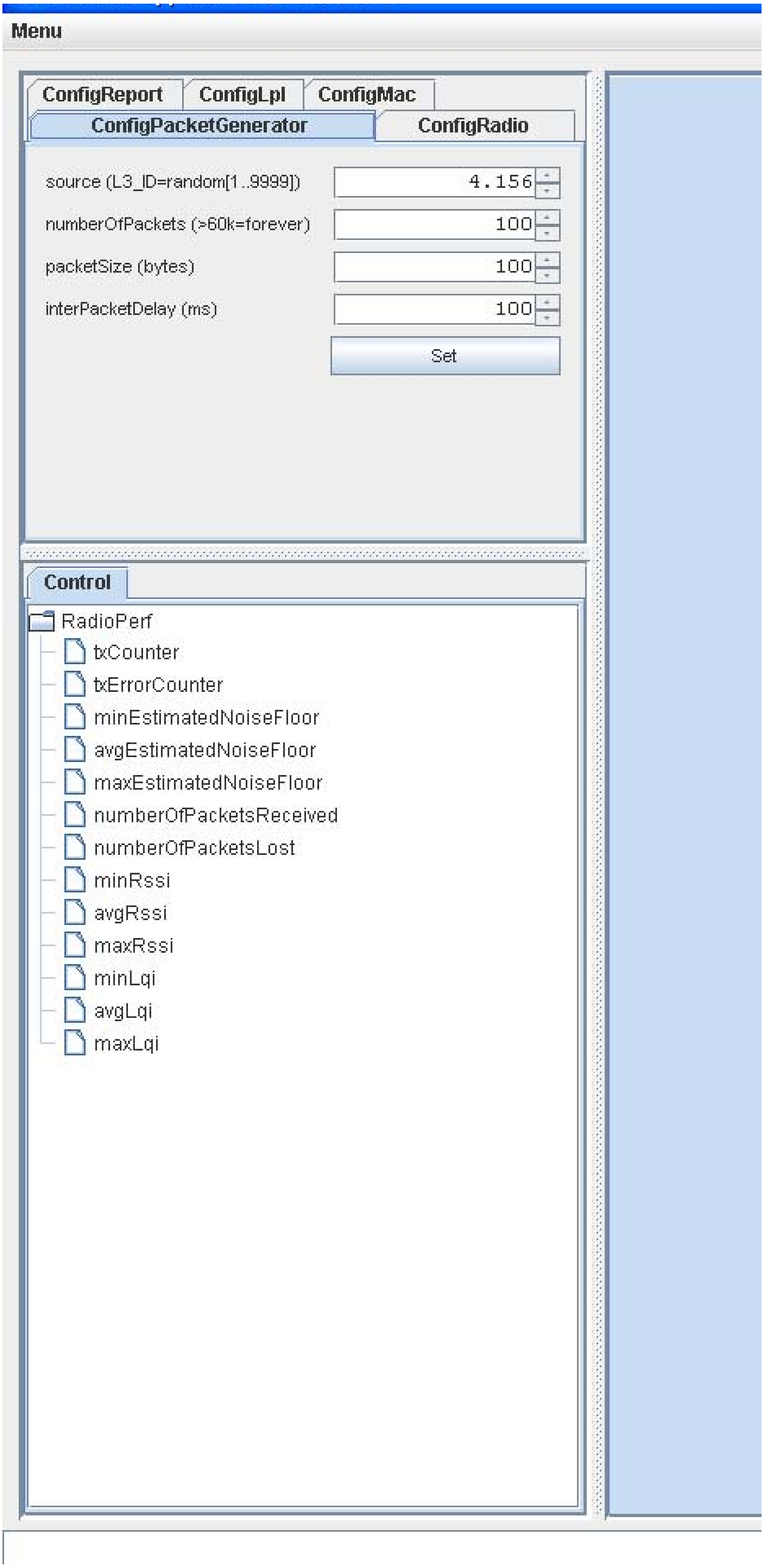
then your CLASSPATH was not correct. Make sure your CLASSPATH contains the **javalib** directory.

Running RadioPerf

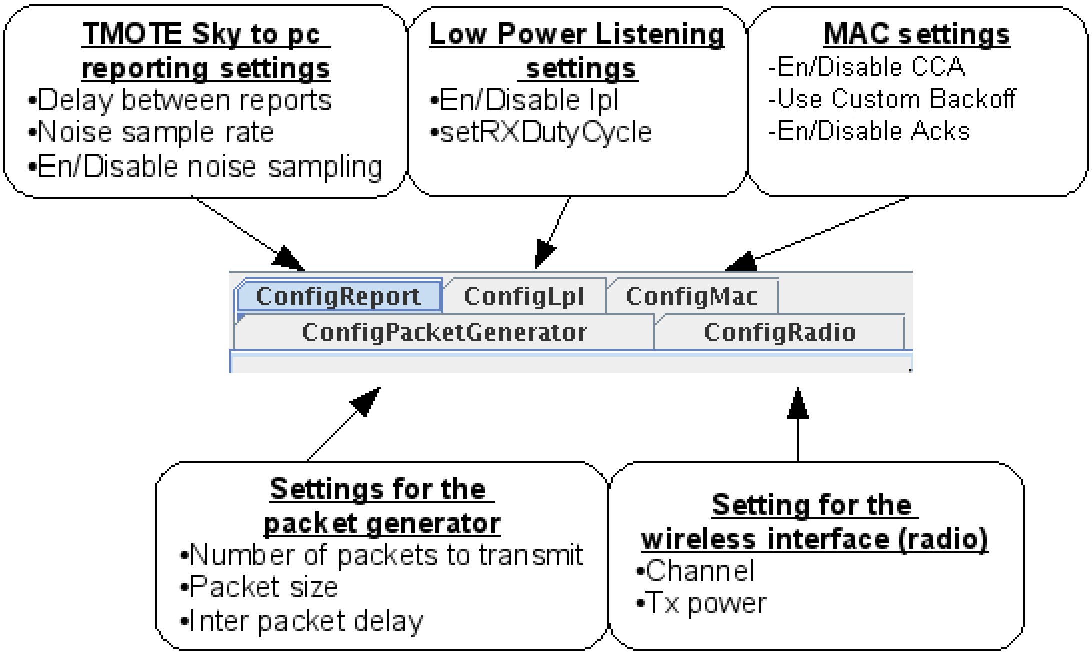
Step into the java_netbeans-5.5 directory and execute :

./run -comm serial@/dev/ttyUSB0:telosb

You should now get a screen such as shown in the figure below.


Menu
ConfigReport
ConfigLpl
ConfigMac
ConfigPacketGenerator
ConfigRadio
source (L3_ID=random[1..9999])
4.156
numberOfPackets (>60k=forever)
100
packetSize (bytes)
100
interPacketDelay (ms)
100
Set
Control
RadioPerf
txCounter
txErrorCounter
minEstimatedNoiseFloor
avgEstimatedNoiseFloor
maxEstimatedNoiseFloor
numberOfPacketsReceived
numberOfPacketsLost
minRssi
avgRssi
maxRssi
minLqi
avgLqi
maxLqi

The different possible settings can be adjusted in the upper left corner of the window. The figure below gives an overview of the most important settings per category.



The lower left window contains several measured statistics that can be shown by clicking on the relevant item:

- txCounter Indicates the number of packets which have been transmitted.
- txErrorCounter Indicates the number of packets that the transmitter could not send.
- minEstimatedNoiseFloor Returns the lowest result from the environment sampler.
- avgEstimatedNoiseFloor Returns the average result from the environment sampler.
- maxEstimatedNoiseFloor Returns the highest result from the environment sampler.
- numberOfPacketsReceived Indicates the number of packets that have been received by the node.
- numberOfPacketsLost Indicates the number of packets that have been lost as calculated by the receiver (each packet contains a sequence number. A missing sequence number is a lost packet for the receiver).
- min/avg/maxRSSI Indicates the RSSI (received signal strength) of arriving packets (in dBm).
- min/avg/maxLQI Gives an indication of the LQI (Link Quality Indicator) of arriving packets (between 0 and 255).

Setting up your own benchmarking experiments

The w-iLab.t testbed provides a set of tools to support benchmarking and repeatable experiments in general. Currently, these tools can be used separately or in conjunction to create a complete benchmarking workflow.

Repeatable Wi-Fi experiments

Repeatability is a strong concern when considering wireless experiments. Through the iPlatform concept the testbed provides the repeatable execution of scripts on the iNodes. Using iPlatforms, a user defines a remote mount of the w-iLab.t fileserver on each node, where an executable file, **start_mount_script**, will be executed after node booting.

By design choice there is no strict synchronization present in the execution of these start scripts, but the code for using the shared directories for synchronization is available for download here. By using this or a user chosen method is is possible to schedule and forget your benchmarks, and analyse them afterwards.

For more information on using the iNodes for your experiments, please see the detailed iNode documentation.

Creating a repeatable environment

Part of the CREW benchmarking goals is the creation of repeatable environments. On the w-iLab.t testbed we currently can provide a repeatable home environment that can be customized to a user's needs. Variations of this and future environments are to be released as well. To see a demonstration of this environment, please see the advanced tutorial.

To use the Home Environment in your experiments, download the iPlatform scripts here, unzip them in an iPlatform directory of your choosing and give **start_mount_code** executable permissions. Before running, please take a look at the variables file, containing all adjustable parameters for the experiment. The file is annotated, but the most important variables are listed in the following table

VARIABLE	Purpose
USERNAME	w-iLab.t database username
USERPASS	w-iLab.t database password
USERDB	w-iLab.t personal database (often equals username)
NCSERVERDIR	Your iPlatform directory
CHANNEL	The 802.11g channel used
TXPOWER	Transmission power
DURATION	Total runtime of the script
EMAILINTERVAL	Duration between email checks
DATAWAIT1	Start first data download after x seconds
DATADURATION1	First data download will take x seconds
DATAWAIT2	Start second data download x seconds after the first
DATADURATION2	Second data download will take x seconds
VIDEOWAIT	Start video stream after x seconds
VIDEODURATION	Video stream will take x seconds
VIDEOBW	UDP bandwidth used by the video stream in Mbps

We are currently transitioning to a new experimentation control framework for w-iLab.t (OMF), where the experiments themselves can be parametrized, allowing a more generic approach to defining an environment. When available, a detailed explanation to this new approach will also be available here.

Benchmarking Wireless Sensor Networks

The w-iLab.t testbed provides different facilities for a WSN protocol developer to benchmark their own code. The only requirement is that your developed code is compatible with the telosb mote. Any WSN code can be benchmarked using our repeatable environments if the variables that need to be varied are exposed as global variables in your WSN code (see how to change global variables at schedule time). However, a benchmarking API is provided that takes care of the repeatable execution of your WSN code and reports all logged data in a standardized format to the w-iLab.t database for quick visualization.

This API is implemented as TinyOS modules that should be included in your compiled TinyOS image, or for the IDRA framework, which is a networking focused modular development framework. The IDRA API is closely supported and follows the latest features of IDRA, and can be downloaded here. To learn more about IDRA, its purpose and how to configure it, please visit the official website, idraproject.net. The TinyOS modules are currently being updated to support the same features and will be available soon.

To schedule benchmarks using the provided API w-iLab.t uses a BNF syntax to define parameters or parameter traversals. More information on the BNF system is available here.

The full benchmarking API is given in the following table. for IDRA, these variables are available as Benchmarking_P.<parameter_name>

Parameter name (default value)	description	range
node_purpose (0)	Send packets? 0:no 1:yes	0 - 1
target (1)	Node id of packet destination	0 - 65535
data_size (15)	Size of application payload in B	6 - 255 (6B needed for tracking)
send_interval (15000)	Packet Interval (PI) in ms	0 - 2^{32}-1
send_variation (15000)	Wait before first packet in ms	0 - 2^{32}-1
random (0)	Random Packet Interval?	0 - 1
random_mindelay (500)	Minimal random PI in ms	0 - 65535
random_window (500)	Random PI window in ms	0 - 65535
retry (0)	Retry failed send attempt	0 - 1
retry_mindelay (150)	Minimal retry delay in ms	0 - 65535
retry_window (150)	Retry window in ms	0 - 65535

anycast (0)	Ignore destination at receiver	0 – 1
debug_mode (3)	Logging method	0:none - 1:aggregates - 2:only network info - 3:all
aggregation_interval (10000)	When to output aggregated logs	0 - 2^{32}-1

Benchmarking analysis

The final step in the benchmarking process, is also supported by the w-iLab.t testbed. When using the WSN API all the logs from a benchmark are automatically inserted into a separate database table using a fixed format. This table is not restricted to WSN results, but then the data has to be inserted following the described logging format below. This is however the only requirement to use the provided analysis tools

Column name	description	range
version	Versioning number	0 - 255
type	Type of log message	0 - 255
arg	Argument of message	0 - 65535
msg_uid	Uid of logged packet	0 - 65535
origin	Origin of logged packet	0 - 65535
other_node	Destination of logged packet	0 - 65535
Msg_seq	Sequence no. of logged packet	0 - 65535
seqno	Sequence no. of log message	0 - 2^{32}-1
motelabMoteId	Node id (db generated)	0 - 65535
motelabSeqNo	Global sequence no. (db generated)	0 - 2^{32}-1
insert_time	Log time (db generated)	0 - 2^{32}-1

Following events can currently be logged, according to the benchmarking API: node purpose, boot time, send attempt, send success, send failure, radio sleep duration, total sent packets, total received packets, debug statistics (total debug msgs/fails)

All log data is processed using sql instructions and presented in a barchart, linechart or a 2d map of a testbed using the analyser and visualiser tools. For more information how to use and configure these tools can be found here.

Following metrics and visualisations are implemented and available for download here

- **Reliability**: calculated on application level, each packet that is sent by a SUT should be received by the destined SUT to reach 100% reliability for the sending SUT. Available as analyser and visualiser tool.
- **Packets sent/received**: The total amount of packets sent or received by the radio adapter, also available as packets sent/received per second. Available as analyser and visualiser tool.
- **Radio sleep percentage**: The fraction of the benchmark that the radio adapter spends sleeping, this is the primary energy efficiency metric for WSN and similar networks of embedded devices. Available as analyser and visualiser tool.

- **Wifi throughput**: Visualises the network wide wifi throughput, as logged by the environment. Can only be used when using one of the repeatable environments. Available as analyser.
- **Application level events**: the amount of network wide events visualised over time, includes packet sending, receiving, errors and boot times. Available as analyser.

Attachment	Size
synchronization.zip	677 bytes
IDRA_Benchmarking.zip	1.99 MB
visualiser_analyser_config.zip	10.95 KB

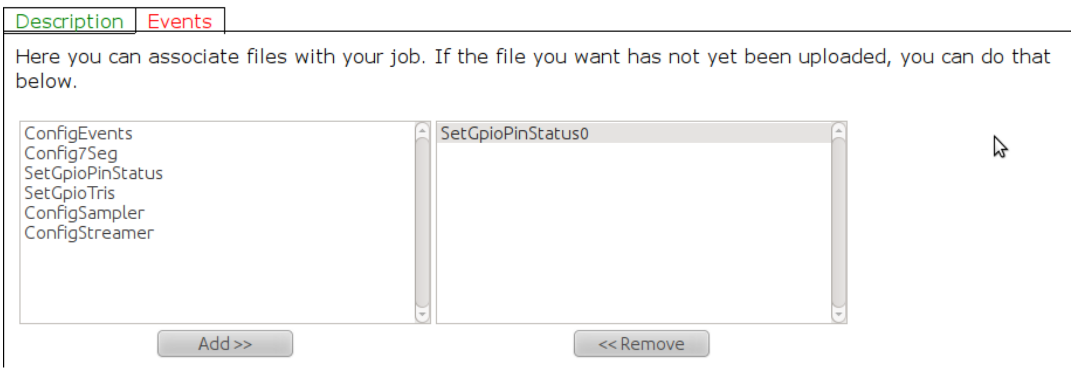
Using Environment Emulator events

Environment Emulator

First check out the EE hardware page for information about the capabilities and hardware specifications of the Environment emulator.

Create a scenario

This page will describe step by step how you can create a scenario which will disable the USB port on the Environment Emulator on some locations, thus cutting the power to the sensor node and re-enable the USB port after 5 minutes.

1. Go to the scenario page on Wilab.t and click **Create new scenario**.
2. Insert a name and description(optional) and click **Next**.
3. One scenario contains one or more events. In the list on the left, you can see all available events that can be defined on the Environment Emulator. Select **SetGpioPinStatus** and click **Add>>**.
4. 

   Scroll down to see more info on the event itself and for a list of all pins that can be set or cleared. If we want to disable the USB port to which the sensor node is connected, we need to **clear GPIO pin B**.

CLEAR_GPIO_A	EE Interrupt 3 (Make input or LOW!!!)	TMote Reset (Reset -Button- is ACTIVE LOW, button down gives a 0)	☐
CLEAR_GPIO_B	EE (Forced to output, active high) Enable USB straigth connector	TMote USB Power	☑
CLEAR_GPIO_C	EE (Forced to output, active high) Enable USB orthogonal connector	USB Power for other device	☐

5. Now scroll down to the bottom of the page to select on which nodes you want to execute the event. Node ID's in this list should be separated by comma's. The use of ranges is also permitted. (e.g.: 1-50,62,63)
6. The last item we need to specify is when the event is executed relative to the start of the experiment. Just choose a reasonable value here.
7. Now that we have an event that disables the USB port, we just need to add another similar event that will re-enable the USB port. Select SetGPIOPinStatus again, scroll down and now select **Set GPIO B** instead of Clear.
8. Click the **Submit** button to save your scenario.

Info on other possible scenario's can be found on the scenario page on Wilab.t. Just select an event, add it to your scenario and additional info on that event will appear.

Execute a scenario

Manually

If your experiment is running, you can choose to execute your scenario by selecting it on the scenario edit page and clicking the **Start Scenario** button.

Automatically

It is possible to connect your scenario to an experiment. Select one of your previously created jobs and go to the **Scenario tab**. In the list on the left, all your scenario's are displayed. Select one and click **Add>>** to connect it to your job.
It is possible to specify the offset for the execution of the scenario('s) (in seconds) relative to the start of the experiment. This value is default set to 45 seconds and should probably not be set to a lower value to be sure all your nodes are started before you execute your scenario.

Example

Check out the http://www.crew-project.eu/portal/wilab/power-measurements-sensornode page for an example experiment using scenario's.

Visualiser & Analyser tools

Introduction

Both of the tools described on this page are made to visualise data that was stored in the MYSQL database during an experiment. The only difference between the visualiser and the analyser is how they show the data to the user. Please take a look at some examples you can find on the toolbox page.

Java Installation

The applet should work by default in Windows. Just install Java JRE if it doesn't.

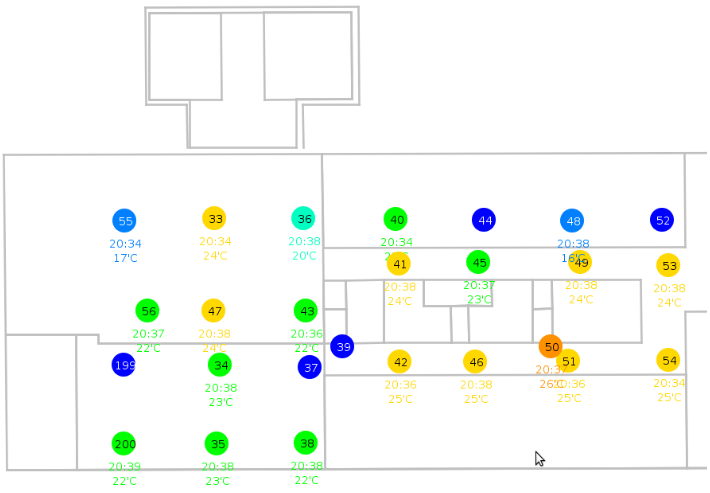
In Ubuntu you will need to install the sun-java6-plugin to get the applet working. The applet will NOT load with the alternative OpenJDK plugin (IcedTea). If you don't find the sun-java6-plugin (apt-get install), then execute following steps where you replace *lucid* by your own Ubuntu version.

```
sudo add-apt-repository "deb http://archive.canonical.com/ lucid partner"
sudo apt-get update
sudo apt-get install sun-java6-jre sun-java6-plugin
sudo update-alternatives --config java
```

The applet has been tested in both Firefox, Internet Explorer and Google Chrome, but should work in other browsers too.

Visualiser

The visualizer shows information collected from the sensor nodes. The type of information and the properties of the visualiser are defined in a XML-file. You can define your own XML file. Temperature readings across the building are shown by default.



A visualiser XML file should follow the structure as shown in the picture below.

```
-<wilabt_visual>
  +<dbConnection></dbConnection>
  +<graphics></graphics>
  +<map></map>
  +<nodeLocation></nodeLocation>
  +<timeSlider></timeSlider>
  +<nodeInfo></nodeInfo>
  +<linkInfo></linkInfo>
 </wilabt_visual>
```

Database connection

The first section defines the connection to the MySQL database. The info below shows how to connect to the **wilabinfo** database on the Wilab.t server as user wilabinfo. This database contains information about the location of the nodes and the temperature and humidity readings they perform when no other experiment is running.

```
<dbConnection>
    <host>wilab.atlantis.ugent.be</host>
    <port>3306</port>
    <database>wilabinfo</database>
    <user>wilabinfo</user>
    <password>wilabinfo</password>
</dbConnection>
```

Just change the database, user and password fields if you want to connect to your own user database. Your credentials can be found on the user-info page on Wilab.t.

Graphical Settings

The graphics section of the XML file allows you to adjust the looks of the visualiser. For most experimenters the default settings will be fine. If you want to play around with e.g. the diameter of the nodes, or the font of the text, just adjust the values in the right section and check out the result.

```
<graphics>
    <zoom>1</zoom>
    <heartbeat>1000</heartbeat>
    <nodes>
    <diameter>150</diameter>
    <zoomFactor>0.8</zoomFactor>
    </nodes>
    <font>
    <size>70</size>
    <zoomFactor>0.8</zoomFactor>
```

```
</font>
<lines>
<thickness>1</thickness>
<zoomFactor>1</zoomFactor>
</lines>
<links>
<bidirectional>true</bidirectional>
</links>
</graphics>
```

Map section

The map section tells the visualiser where it can find the coordinates of all the walls. A wall is stored in the database by two coordinates (X1,Y1) representing one end of the wall and (X2,Y2) representing the other end. This section should never be modified if you are using the Wilab.t testbed. You could modify this section if you wanted to visualise e.g. your own building.

```
<map>
<sql info="selects all the nodes">
select x1, y1, x2, y2, floor from map where floor&gt;0
</sql>
<column_x1>x1</column_x1>
<column_y1>y1</column_y1>
<column_x2>x2</column_x2>
<column_y2>y2</column_y2>
<column_floor>floor</column_floor>
</map>
```

Node locations section

Similarly to the map section, the node section defines the coordinates of all the nodes that you want to show on the visualiser. Every node is specified by an ID, an (X,Y) coordinate and a floor. This section should also not be modified for most experiments on Wilab.t.

```
<nodeLocation>
<sql info="selects all the nodes">
select id, x, y, floor from coordinates where floor&gt;0
</sql>
<column_id>id</column_id>
<column_x>x</column_x>
<column_y>y</column_y>
<column_floor>floor</column_floor>
</nodeLocation>
```

Timeslider

The timeslider on the visualiser (double-click on the map) allows you to visualise your experiment after it has been completed. It also has the ability to automatically replay your experiment. The XML for this should never be modified.

Node info

This is probably the section where you will have to modify some things to represent your own experiment. There are 3 separate properties you can control :

1. On every node you can define what label is shown on the node itself. This info is called the ID.
2. Additionally you can define what kind of information is shown under the nodes. This is called the info field.
3. Lastly you can choose what color is being used to display a node. This is the color field.

To show your own info, just write some SQL statement and give the result the alternative name as shown above (so id, info or color). The XML below shows an example of the temperature visualiser on Wilab.t.

```
<nodeInfo>
<sql>
SELECT
moteid as id,


CASE
WHEN avg(temp) = 0 THEN concat(hour(max(updated)), ":",
minute(max(updated)))
WHEN avg(temp) &gt; 50 THEN concat(hour(max(updated)), ":",
minute(max(updated)))
WHEN avg(temp) &gt; 0 THEN concat(hour(max(updated)), ":",
minute(max(updated)), "~", round(avg(temp)), "'C")
ELSE 'no info'
END
as info,


CASE
WHEN avg(temp) = 0 THEN '0x000000'
WHEN avg(temp) &gt; 50 THEN '0x000000'
WHEN avg(temp) &gt; 29 THEN '0xFF0000'
ELSE '0x0000FF'
END
as color


from sensorinfo, timeInfo
where @timeslider -7*60 &lt; UNIX_TIMESTAMP(updated) AND
UNIX_TIMESTAMP(updated) &lt; @timeslider + 3*60
group by moteid
</sql>
```

```
<column_id>id</column_id>
<column_info>info</column_info>
<column_color>color</column_color>

</nodeInfo>
```

Link info

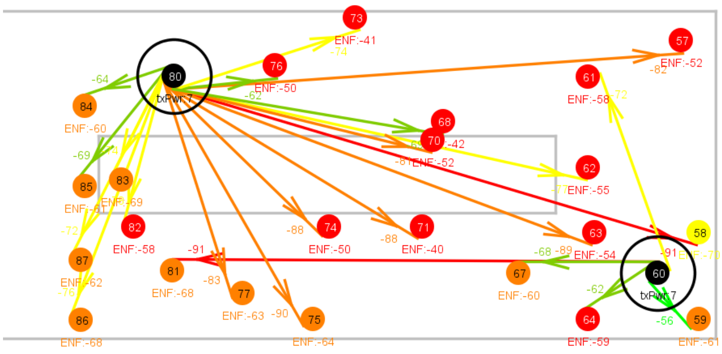
The visualiser is also able to show packet transmissions. A link has several properties :

1. id_begin : the ID of the sending node
2. id_end : the ID of the receiving node
3. info : some extra information shown on the link
4. color : the color of the arrow

If you don't want to show any link info, just leave the XML as shown below in the configuration file. If you want to show links, adjust the XML file in the same way as was shown for the Node info section.

```
<linkInfo>
    <sql>
    SELECT
    0 as id_begin,
    0 as id_end,
    '1' as info,
    '0x00FF00' as color
    from sensorinfo
    where id =0;
    </sql>

    <column_id_begin>id_begin</column_id_begin>
    <column_id_end>id_end</column_id_end>
    <column_info>info</column_info>
    <column_color>color</column_color>
</linkInfo>
```

The figure below shows what can be achieved with the visualiser. In this example node 60 and 80 are sending with transmit power 7 (txPwr) and all receiving nodes are pointed to with an arrow showing the receiving RSSI of that link. Also, every node shows its estimated noise floor (ENF) .

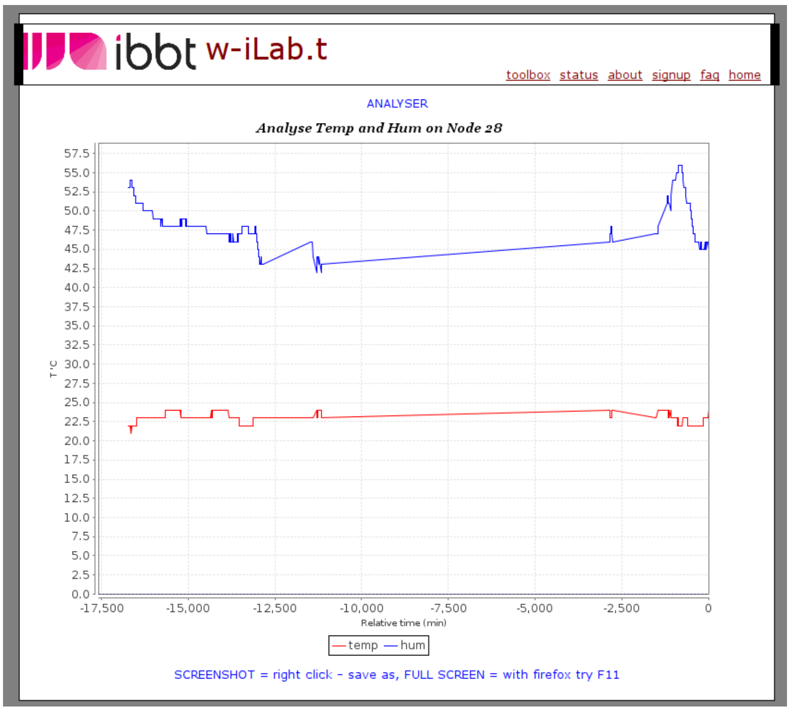


User Parameters

Whenever you want the visualiser to ask the user to fill in a parameter use three underscores before and after the parameter name (e.g. : ___MOTEID___ , ___Username___ , ___Password___ , ...).

Analyser

The analyser shows a chart with information collected from the sensor nodes. The type of information and the properties of the chart are defined in a XML-file. You can define your own XML file for the analyser. An example of the analyser showing temperature and humidity readings from node 28 is shown below.



The XML file looks like this :

```
-<wilabt_analyser>
  +<dbConnection></dbConnection>
  +<graphics></graphics>
  +<general></general>
  +<scatters></scatters>
  </wilabt_analyser>
```

Database connection

The database connection happens in the same way as for the visualiser.

Graphics

```
<heartbeat>10000</heartbeat>
<info>Analyse Temp and Hum on Node ___MOTEID___</info>
<xAxis>Relative time (min)</xAxis>
<xMin>0</xMin>
<xMax>0</xMax>
<xScaleStep>0</xScaleStep>
<yAxis>T 'C</yAxis>
<yMin>0</yMin>
<yMax>0</yMax>
```

- The heartbeat defines the refresh rate of the analyser (in seconds)
- Info defines the label that is shown at the top of the analyser.
- xAxis : label on the x axis
- xMin : start of the range for the x axis (can be 0 if x is defined in the scatter section)
- xMax : end of the range for the x axis (can be 0 if x is defined in the scatter section)
- xScaleStep : size of the steps on the x axis (can be 0 if x is defined in the scatter section)
- yAxis / yMin / yMax : Same as x axis

General

This section can be used to prepare some views that can be used in the scatter section.

```
<prepareView info="get last samplereport">
        CREATE OR REPLACE VIEW sensorinfofirst AS
                select *
                from sensorinfo
```

```
where motelabSeqNo <= 20
ORDER BY motelabSeqNo DESC
</prepareView>
```

In this example you could then use the view **sensorinfofirst** in the scatter section.

Scatters

In the example below two scatters are defined. One shows the temperature on the y-axis, the other one shows the humidity. Both scatters have relative time as x-axis.

```
<scatters>
<scatter>
<name>temp</name>
<color>0xFF0000</color>
<sql>
select TIMESTAMPDIFF( MINUTE, timeInfo.lastInsert, updated) as x, temp
from sensorinfo, timeInfo
where moteid=___MOTEID___
ORDER BY updated DESC, id DESC
</sql>
<column_x>x</column_x>
<column_y>temp</column_y>
</scatter>

<scatter>
<name>hum</name>
<color>0x0000FF</color>
<sql>
select TIMESTAMPDIFF( MINUTE, timeInfo.lastInsert, updated) as x, hum
from sensorinfo, timeInfo
where moteid=___MOTEID___
ORDER BY updated DESC, id DESC
</sql>
<column_x>x</column_x>
<column_y>hum</column_y>
</scatter>
</scatters>
```

User Parameters

Whenever you want the analyser to ask the user to fill in a parameter use three underscores before and after the parameter name (e.g. : ___MOTEID___ , ___Username___ , ___Password___ , ...).

Debugging

If you start writing your own visualiser or analyser XML configuration files, be sure to activate the java console. This way you can see that queries the applet will execute and track down errors if necessary.

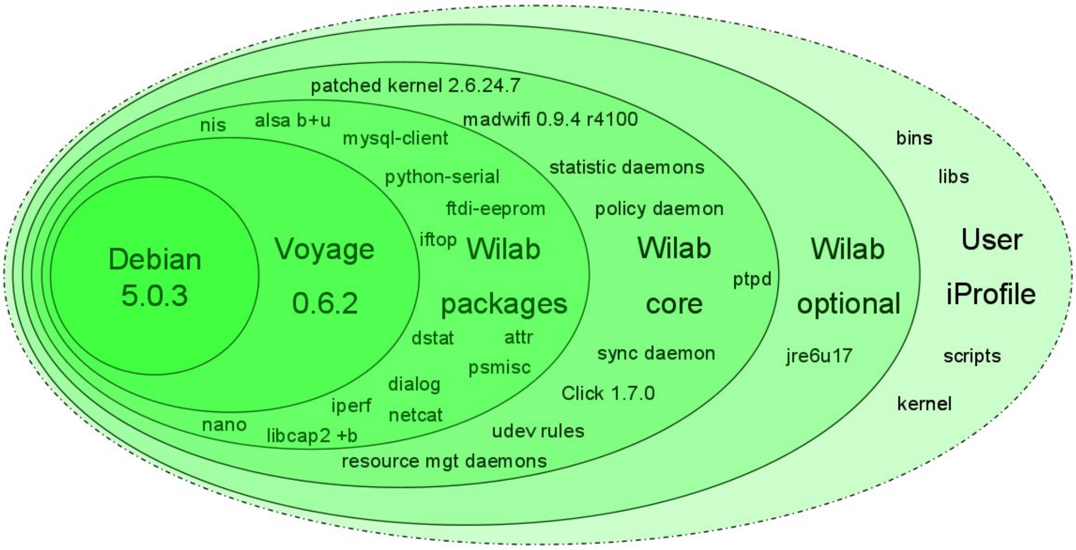
How to enable java console in

- Ubuntu :
    - Open a terminal and type : *jcontrol*
    - Go to the *advanced* tab and expand the *Java console* item.
    - Select *Show console* and click *OK*
    - Restart your browser just to be sure the new settings are activated.
- Windows :
    - *Start* > *Control Panel* > Search for *Java*
    - Click the *Java* icon
    - Go to the *advanced* tab and expand the *Java console* item.
    - Select *Show console* and click *OK*
    - Restart your browser just to be sure the new settings are activated.

iNode: use of embedded PC and Wi-Fi

iNode OS and software

- Debian 5.0.3 (Lenny) Voyage distro 0.6.2
- Patched kernel 2.6.24.7 for click
- Click modular router 1.7.0 with all elements available in user level and as kernel module
- madwifi driver with the accompanying tools
- Time synchronization via ptpd IEEE 1588 standard
    - Convergence (offset from master)
    - < 100 us after ca. 8 min
    - < 10 us after ca. 13 min



The user can get acces to every node via ssh by configuring an iPlatform which can contain bins, scripts, libs, configuration and an optional specific kernel and initrd. Once logged in via sudo the user has unlimited access to the system. We discourage the use of 'apt-get install' and 'remountrw' as it can have a big impact on the maximum write cycles of the compact flash. We prefer to prepare the bins, libs and confs in the iPlatform directory. There is one special binairy with the name start_mount_script that will be invoked by the linux rc.local (at the end of the linux boot proces) when it can find this file in the root of the iPlatform mount. For every experiment the /tmp/log is mounted to ./log/ScheduleID/NodeID/ under iPlatform root directory on wilabfs via nfs. Also the user's home directory is available on every inode.

At every time the user can power off and on again the inode as needed during the experiment or during the debugging fase via the status tab on wilab.

Developer documentation

The original version of w-iLab.t was based on Motelab (http://motelab.eecs.harvard.edu/). Over the years, the Motelab code was significantly extended (*). As major reconstructions of the code are currently taking place, the source of w-iLab.t and the w-ilab.t tools source is currently not publicly accessible. Please contact Bart.Jooris [AT] intec.ugent.be for more information. We aim to make the new release of w-iLab.t fully open source. Please check back here for updated information.

(*) Why w-iLab.t is not just a Motelab clone:

Motelab is a **passive sensor lab** where the DUTs are restricted to the Tmote Sky (DUT=device_under_test).
w-iLab.t is an **active wireless lab**.

Passive versus active

Active=interact on the Environment of the sensors

- Current measurement with sample rate up to 10kHz on all the DUTs
- Battery voltage can be adjusted at any time on all the DUTs (energy harvesting)
- Audio can be injected and acquired into and from the DUTs using the soundblaster of the iNode.
- Analogue and digital hardware events (like node reset) can be triggered on and acquired from the DUTs.

  All of this can be prepared before running an experiment or can be adjusted real time during the experiment by using the scenarios tab.

Sensor lab versus wireless lab

- The embedded PC acting as intermediate node (iNode) between the control server and sensor device can become an active member of the experiment (kernel adj., drivers adj., click router code, java)
- w-iLab.t supports WiFi, sensor, cognitive,... (and not only sensor...) experiments
- The lab can be extended with other wireless technologies (BT, IrDA, 3G,...)

FAQ

Before contacting us, please take the time to go through the FAQ list below. This list will be updated frequently.

Q: What does the W-iLab.t testbed do? What does it measure?

A: The w-iLab.t is a heterogeneous, generic wireless testbed. Basically, the testbed "does nothing" by itself, but can behave in lots of different ways, depending on how it is configured by the user. The w-iLab.t hosts sensor hardware, embedded PCs, Wi-Fi interfaces, cognitive radio platforms etc. The behavior is determined by selecting a subset of this hardware, and installing the drivers/software/scripts of your choice. To get you started, there are default configurations available; For example, when no experimenter is using the testbed, the sensor nodes are programmed by us to continuously measure the temperature in their environment; these measurements are stored in a database. Please check the tutorial section to get started with some default images.

Q: To what extent can I change the application layer / routing protocol / MAC layer / Physical layer / ... ?

A: The only things that are fixed in our testbed are the hardware components and their interconnections. You can make full use of the hardware as if it would be located on your desktop, but, obviously, you can not bypass the limitations of the testbed components. To be *more precise, for the sensor nodes and Wi-Fi cards, it is impossible to make PHY layer adjustments. MAC and higher layer modifications are supported, as long as you are able to implement the changes yourself. As a simple example: if you e.g. are able to create a new wireless Wi-Fi driver that allows you to change the output transmission power on a per packet basis and this driver works together with the testbed hardware, you will be able to use this driver on the testbed. To know whether our hardware supports your modifications, please check the data sheets of our hardware.*

Common data format

Many cognitive usage scenarios that take place can be 'recorded'. In our federation data recorded in one testbed is usable in other testbeds to support emulated usage scenarios (e.g. primary user data recorded in testbed A feeds into a sensing device in testbed B).

To this end, CREW defined data of interest, common structures for storing data and is in the process of creating a federation database for storage of any collections made.

Preliminary information on this common data format can is available in this document.

Attachment	Size
common-data-format.pdf	327.93 KB

9 Appendix B: BEE2 example .json


```
{
    "Experiment Abstract" : {
        "Title" : "Transmitter calibration of the radio Front Ends for
BEE2",
        "Tag" : "2011-1-Chwalisz",
        "Authors" : [
            {
                "Name" : "Mikolaj Chwalisz",
                "Email" : "chwalisz@tkn.tu-berlin.de",
                "Address" : "Einsteinufer 25, 10587 Berlin, Germany",
                "Phone" : "+49 30314 23824"
            },
            {
                "Name" : "Daniel Willkomm",
                "Email" : "willkomm@tkn.tu-berlin.de",
                "Address" : "Einsteinufer 25, 10587 Berlin, Germany",
                "Phone" : "+49 3031421980"
            }
        ],
        "Release Date" : "2011-04-07",
        "Experiment summary" : "The calibration is a process aimed to give
a meaningful comparison between measurements made by one device, with known
magnitude and correctness, and a second device. This step is essential to
be able to compare results with other experiments, specially with custom
made devices. The other goal of the calibration is to determine the
condition of the instrument to perform measurements. This also includes the
ability to transfer defined measurement units.\nIn order to calibrate the
receiver, it is necessary to have a calibrated transmitter.\nIn this
experiment we try to calibrate BEE2 Front End as the transmitter based on
signal received by the R&S FSV Spectrum Analyzer",
        "Collection methodology" : "Devices where set to one frequency and
the power level of the generic OFDM was measured. Whole experiments were
done with cable connection. Transmitting device is set to one center
frequency",
        "Further documentation" : {
            "Description" : "The measurements where published in master
thesis of Mikołaj Chwalisz.",
            "Bibtex" : [
                "bibtexentry"
            ]
        },
        "Notes" : "none"
    },
    "Meta information" : {
        "Devices" : [
            {
                "Name" : "BEE2 Board",
                "Description" : "The Berkeley Emulation Engine 2 (BEE2) was
developed to be a reusable, modular,  and scalable framework for designing
high-end reconfigurable computers at the Berkeley Wireless Research Center
(BWRC). It is supposed to help solving computationally intensive problems
such as: emulation and design of wireless communication systems, real-time
scientific computation,  high-performance real-time digital signal
processing.",
                "Datasheets" : [
                    "C. Chang, J. Wawrzynek, and R.W. Brodersen, BEE2: a
high-end reconfigurable computing system, Design Test of Computers, IEEE 22
(2005), no. 2, 114–125.",
                    "http://bee2.eecs.berkeley.edu/wiki/BEE2wiki.html",
```

```
                    "http://bee2.eecs.berkeley.edu/"
                ],
                "Software" : {
                    "Description" : "MSSGE (Matlab / Simulink / System
Generator / EDK) toolchain for FPGA designs. Used CASPER libraries and code
from BWRC (Berkeley).",
                    "OperatingSystem" : "Linux BORPH",
                    "Driver" : "none",
                    "Application Name" : [
                        "fpga1fe2011, ",
                        "cntrlfpga2009"
                    ],
                    "Code" : "Ask authors."
                }
            },
            {
                "Name" : "Radio Front End",
                "Description" : "The Berkeley Emulation Engine 2 (BEE2) was
developed to be a reusable, modular,  and scalable framework for desigThe
radio capabilities for BEE2 board in the CogRad testbed are provided by the
radio Front End. It is made of the baseband board performing data
processing, control and digital to analog conversion. The daughter card is
used to perform up/down signal conversion to 2.4 GHz.",
                "Datasheets" : [

"http://bwrc.eecs.berkeley.edu/Research/Cognitive/prototyping_platform.htm"
                ],
                "Software" : {
                    "Description" : "MSSGE (Matlab / Simulink / System
Generator / EDK) toolchain for FPGA designs. Used CASPER libraries and code
from BWRC (Berkeley).",
                    "OperatingSystem" : "none",
                    "Driver" : "none",
                    "Application Name" : "fe2011....",
                    "Code" : "Ask authors."
                }
            },
            {
                "Name" : "RS FSV Spectrum Analyzer",
                "Description" : "OTS Spectrum Analyzer",
                "Datasheets" : [
                    "http://www2.rohde-schwarz.com/file/FSV_dat-sw_en.pdf"
                ],
                "Software" : {
                    "Description" : "NA",
                    "OperatingSystem" : "NA",
                    "Driver" : "NA",
                    "Application Name" : "NA",
                    "Code" : "NA"
                }
            }
        ],
        "Space" : {
            "Mobility" : "none",
            "Layout" : "Cable connection between devices."
        },
        "Time" : "Couple of seconds per measurement",
        "Signal generation" : {
            "Description" : "For signal generation the FE was used. One or
two OFDM symbols stored in FE's FPGA fabric and send repeatedly. Resulting
```

```
in constant OFDM stream. Matlab file with I/Q samples is available as well
as the scripts to create it.",

            "Trace" : "OFDM generation Matlab code is available, contact
author."
        },
        "Radio Frequency" : {
            "Interference Sources" : " None, cable connection",
            "Operating Range" : "2.4 GHz ISM band, 2400 – 2483 MHz"
        },
        "Parameters" : [
            {
                "Description" : "Cable attenuation",
                "Name" : "Att",
                "Unit" : "dB"
            }
        ],
        "Trace Description" : {
            "Format" : "Text file with the following structure: \nParameter
listing:\nName; Value; (Unit)\nValues;Number of values; \nVector:
Frequency;dBm \nAdditional PNG file with spectrum analyzer screen shot",
            "Collected Metrics" : [
                {
                    "Name" : "RFPower",
                    "Unit of Measurements" : "dBm",
                    "Accuracy" : "+-0.28dB"
                }
            ]
        }
    },
    "Experiment Iterations" : [
        {
            "Description" : "10dB Attenuator added into cable",
            "Time" : "2011-01-20T16:05+02:00",
            "Parameters" : [
                {
                    "Name" : "Att",
                    "Value" : 10
                }
            ],
            "Trace files" : [
                "fec_att10dB_count500_swt_ms_clrw.DAT",
                "fec_att10dB_count500_swt_ms_clrw.png"
            ]
        },
        {
            "Description" : "signal was averaged over 500 sweeps",
            "Time" : "2011-01-20T16:05+02:00",
            "Parameters" : [
                {
                    "Name" : "Att",
                    "Value" : 0
                }
            ],
            "Trace files" : [
                "fec_att0dB_count500_swt1.1ms_rbw100khz_avg.DAT",
                "fec_att0dB_count500_swt1.1ms_rbw100khz_avg.png"
            ]
        }
    ]
}
```

10 Appendix C: Outdoor spectrum sensing with VSN


```
{
    "Experiment Abstract": {
        "Title": "Outdoor spectrum sensing with VSN",
        "UID": "2011-01-20T11:05+02:00::cfortuna::VSNMirenVarSweep",
        "Authors": [
            {
                "Name": "Carolina Fortuna",
                "Email": "carolina.fortuna@ijs.si",
                "Address": "Jamova 39, Ljubljana, Slovenia",
                "Phone": "+386 1 477 3114"
            },
            {
                "Name": "Zoltan Padrah",
                "Email": "zoltan.padrah@ijs.si",
                "Address": "Jamova 39, Ljubljana, Slovenia",
                "Phone": "+386 1 477 3114"
            },
            {
                "Name": "Marko Mihelin",
                "Email": "marko.mihelin@gmail.com",
                "Address": "Jamova 39, Ljubljana, Slovenia",
                "Phone": "+386 1 477 3114"
            }
        ],
        "Release Date": "2011-01-20T11:05+02:00",
        "Experiment summary": "Measurement of spectrum occupancy in a rural
area (Miren, Slovenia) using static low cost sensors called Versatile
Sensor Nodes. The measured spectrum will then be compared to the same
measurements performed using calibrated USRP. ",
        "Collection methodology": "Several sensors have been placed in
outdoor environment on a fixd frequency, no controled transmitters were
used - we just measure normal everyday power spectrum.",
        "Further documentation": {
            "Description": "Similar experiments were reported in ISABEL
2010 paper.",
            "Bibtex": [
                "bibtexentry"
            ]
        },
        "Notes": "http://sensorlab.ijs.si/publication/9/ism-bands-spectrum-
sensing-based-on-versatile-sensor-node-platform"
    },
    "Meta information": {
        "Devices": [
            {
                "Name": "VSN",
                "Description": "The JSI Versatile Sensor Node platform",
                "Datasheets": [
                    "http://sensorlab.ijs.si/publication/9/ism-bands-
spectrum-sensing-based-on-versatile-sensor-node-platform",
                    "http://xpack.ijs.si/svn/hardware/"
                ],
                "Data collection": "wireless",
                "Software": {
                    "Description": "Codesourcery toolchain, STM and
Sensorlab libraries with Contiki.",
                    "OperatingSystem": "Contiki",
```

```
                    "Driver": "custom",
                    "Application Name": [
                        "ssappv1.2"
                    ],
                    "Code": {
                        "URL":
"http://xpack.ijs.si/svn/vsndrivers/trunk/VSNDrivers/",
                        "RevisionNo": "435"
                    }
                }
            },
            {
                "Name": "USRP N210",
                "Description": "USRP N210 connected to a PC used for data
processing and connectivity. The PC runs GNU radio which then runs the
application. For the RF front end we use a WBX daughterboard.",
                "Datasheets": [

"http://www.ettus.com/downloads/ettus_daughterboards.pdf"
                ],
                "Data collection": "local storage",
                "Software": {
                    "Description": "NA",
                    "OperatingSystem": "Linux on PC",
                    "Driver": "UHD",
                    "Application Name": [
                        "usrpsa1.4",
                        "GNU radio"
                    ],
                    "Code": "NA"
                }
            }
        ],
        "Location": {
            "Mobility": "none",
            "Layout": "http://xpack.ijs.si/Miren.kml",
            "GeoLoc":
"http://api.geonames.org/hierarchy?geonameId=3167024&username=sensors_ijs"
        },
        "Time": {
            "StartTime": "2011-01-20T06:05+02:00",
            "EndTime": "2011-01-20T11:05+02:00"
        },
        "Signal generation": {
            "Description": "No signal has been generated.",
            "Trace": "NA"
        },
        "Radio Frequency": {
            "Interference Sources": "None",
            "Operating Range": {
                "StartFrequency": "815",
                "StopFrequency": "950",
                "Unit of Measurement": "MHz"
            },
            "Parameters": [
                {
                    "Description": "Bandwidth",
                    "Name": "Bandwidth",
                    "Unit": "Hz"
                }
            ],
```

```
            "Trace Description": {
                "Description": "csv file for frequency values and csv file
for measured power.",
                "FileFormat": {
                    "Header": "NA",
                    "Collected Metrics": [
                        {
                            "Name": "Frequency",
                            "Unit of Measurements": "Hz",
                            "Accuracy": "+1Hz"
                        },
                        {
                            "Name": "Power",
                            "Unit of Measurement": "dB",
                            "Accuracy": "+-0.28dB"
                        }
                    ]
                }
            }
        }
    },
    "Experiment Iterations": [
        {
            "Description": "sweep step 10MHz",
            "Time": {
                "StartTime": "2011-01-20T06:05+02:00",
                "EndTime": "2011-01-20T08:05+02:00"
            },
            "Parameters": [
                {
                    "Name": "Frequency",
                    "Value": 0.1
                }
            ],
            "Trace files": [
                "frequency2011-01-20T06:05iteration1.csv",
                "power2011-01-20T06:05iteration1.csv"
            ]
        },
        {
            "Description": "sweep step 20MHz",
            "Time": "2011-01-20T06:35+02:00",
            "Parameters": [
                {
                    "Name": "Frequency",
                    "Value": 0.2
                }
            ],
            "Trace files": [
                "frequency2011-01-20T06:35iter2.csv",
                "power2011-01-20T06:35iter2.csv"
            ]
        }
    ]
}
```