WiSHFUL

Wireless Software and Hardware platforms for Flexible and Unified radio and network control

European Commission

Project Deliverable D5.4

Implementation of optimized experimentation tools

Contractual date of delivery:	31-12-2016
Actual date of delivery:	23-12-2016
Beneficiaries:	IMEC, TCD, TUB, UFRJ, RUTGERS, NCENTRIC
Lead beneficiary:	IMEC
Authors:	Pieter Becue (IMEC), Vincent Sercu (IMEC), Bart Jooris (IMEC), Ingrid Moerman (IMEC), Peter Ruckebusch (IMEC), Spilios Giannoulis (IMEC), Diarmuid Collins (TCD), Ivan Seskar (RUTGERS), Mikolaj Chwalisz (TUB), Jose de Rezende (UFRJ)
Reviewers:	Ivan Seskar (Rutgers) and Diarmuid Collins (TCD)
Work package:	WP5 – Fed4FIRE compliance
Estimated person months:	10
Nature:	R
Dissemination level:	PU
Version:	1.5

Abstract:

This deliverable reports on the activities in the second year of the WiSHFUL project. The first activity was to bring all WiSHFUL testbeds up to FED4FIRE federation standards. It is shown that all testbeds (w-iLab.t, ORBIT, FIBRE Island @ UFRJ, IRIS, TWIST and the Portable Testbed are fully FED4FIRE compliant. This enables the use of unified experimentation tools across all testbeds: jFed, OMF6 and OML. Installation of the above tools and the WISHFUL framework is made easy through the use of the IT automation tool Ansible. This deliverable also reports on the integration of the WiSHFUL UPIs in testbeds. The proposed approach describes a clean interaction between jFed, Ansible and the WiSHFUL framework, which makes it very easy for the experimenter to use the WiSHFUL framework on F4F compliant testbeds. This deliverable is concluded by describing two tools to facilitate SDR and sensor experiments, as well as tools to enable generic measurement visualization and simple spectrum sensing.

Keywords:

Fed4FIRE compliance, Experimentation tools, jFed, OMF6, OML, Ansible, UPI testbed integration, WTA GitHub, MySlice

Executive Summary

This month 24 deliverable reports on the status of FED4FIRE compliance of all WiSHFUL infrastructures and the tools that are provided to the experimenters. All of the testbeds have reached full FED4FIRE compliancy, meaning they can all be accessed using the same tools:

- w-iLab.t at IMEC
- ORBIT at RUTGERS
- FIBRE Island at UFRJ
- IRIS at TCD
- TWIST at TUB
- Portable Testbed

Most of the WiSHFUL testbeds support the use of **jFed** for the provisioning of resources. This allows experimenters to use the same tool and only one certificate to access all testbeds in WiSHFUL. The **OMF6** framework allows experimenters to do uniform experiment control on WiSHFUL infrastructure, while the **OML** instrumentation tool offers a generic software framework for the collection of measurements.

Installation scripts for the tools mentioned above are provided in the form of Ansible playbooks and are available in the ExperimentationTools repository in the **WirelessTestbedsAcademy** GitHub account. Sample configuration files for jFed, OMF and OML can be found in the same repository.

This deliverable also lists the extensions that were made to WiSHFUL testbeds. The w-iLab.t reports the addition of new wireless nodes to increase the capacity of the testbed as well as the support for 802.11ac, LTE and the Adant RAS antenna. In addition, a secondary testbed location of the w-iLab.t was equipped with 44 new wireless nodes supporting 802.11n, 802.11ac, 802.15.4 and Bluetooth 4.0. The IRIS testbed reports the addition of two digital TV tuners. The nodes in the TWIST testbed were extended with a second WiFi interface, as well as a WiSpy spectrum sensor for every node. The ORBIT testbed reports the addition of a number of LTE base stations and clients as well as an increase in the number of SDR platforms.

To allow for an easy integration of **WiSHFUL UPIs** into the testbeds, Ansible playbooks are provided to the experimenter to facilitate the installation of all necessary software on any testbed in the federation. Furthermore, when the experimenter makes use of the **Ansible** support in jFed, **inventory files** can be automatically generated to serve as input for the WiSHFUL Global Monitoring and Configuration Engine (MCE). This way the experimenter can create logical groups of nodes in his experiment and configure every group according to his own wishes. Using this approach, WiSHFUL experiments can easily be re-run on different testbeds without having to modify the Global MCE.

To conclude this deliverable, several tools are described that can be used by the experimenter to do:

- SDR research using GNUradio and/or IRIS.
- in-depth debugging of sensor experiments using the Logic Analyser.
- easy visualization of measurement data.
- simple spectrum sensing using Wi-Spy or USRP B200-mini hardware.

List of Acronyms and Abbreviations

AM	Aggregate Manager
AMQP	Advanced Message Queuing Protocol
CBTM	Cloud Based Testbed Management
EC	Experiment Controller
FRCP	Federated Resource Control Protocol
F4F	Fed4FIRE
GCF	GENI Control Framework
GENI	Global Environment for Network Innovations
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
MCE	Monitoring and Configuration Engine
OEDL	OMF Experiment Description language
OMF	cOntrol and Management Framework
OML	(Orbit) Measurement Library
OMSP	OML Measurement Stream Protocol
RC	Resource Controller
RAM	Random Access Memory
Rspec	Resource Specification
SDR	Software Defined Radio
SFA	Slice-based Federation Architecture
SSH	Secure SHell
TCP	Transmission Control Protocol
UPI	Unified Programming Interface
XML	Extensible Mark-up Language

Table of contents

1 Introduction

This deliverable reports on the work done in WP5 “Fed4FIRE compliance” and gives an overview of the tools that are offered as part of the first experimentation toolset.

The document first gives an overview of all WiSHFUL testbeds and their status of Fed4FIRE compliance. As shown in the overview table in chapter 2, all of the testbeds have now reached full Fed4FIRE compliance, namely w-iLab.t, ORBIT, FIBRE Island @ UFRJ, IRIS, TWIST and the Portable Testbed.

Chapter 3 can be used as a guideline for experimenters. It contains information about all the necessary steps to run an experiment on WiSHFUL infrastructure, including the use of the Fed4FIRE tools. All WiSHFUL testbeds aim to support the use of jFed or MySlice for provisioning of the resources. The OMF6 and OML frameworks are supported on most of the testbed and allow experimenters to do uniform experiment control and measurement collection on WiSHFUL infrastructure. Install scripts for the tools mentioned above are provided in the form of Ansible playbooks and are available in the ExperimentationTools repository in the WirelessTestbedsAcademy GitHub account [2].

The next chapter (chapter 4) lists all the changes that were made to the hardware of WiSHFUL testbeds during Y2 of the project. While the w-iLab.t shows an increase in the total number of nodes and wireless technologies, the IRIS testbed reports the addition of two Digital TV tuners. To conclude the chapter, the TWIST testbed describes their extension by the addition of a second WiFi interface and WiSpy spectrum sensors to all nodes. The ORBIT testbed reports the addition of a number of LTE base stations and clients as well as an increase in the number of SDR platforms.

The final chapter in this deliverable (chapter 5) is dedicated to the integration of WiSHFUL UPI’s in testbeds (see section 5.1) and the integration of tools to support SDR and sensor experiments. The integration of WiSHFUL UPIs in testbeds is based on the support for Ansible in jFed. Using this feature, the experimenter is able to divide the nodes in his experiment into different logical groups. This separation into groups can be used by frameworks like Ansible, OMF6 and WiSHFUL to respectively install software, do advanced experiment control, or do advanced WiSHFUL radio and network control. To conclude the chapter, two tools are detailed which allow the experimenter to do generic measurement visualization and simple spectrum sensing using Wi-Spy or USRP B200-mini hardware.

2 Fed4FIRE compliance of WiSHFUL testbeds

This chapter gives an overview of all WiSHFUL testbeds and their status of Fed4FIRE compliance. As can be seen in Table 1, all WiSHFUL testbeds are fully FED4FIRE compliant. The following sections give more detailed information per testbed. To limit duplication, this document only mentions testbeds for which the level of Fed4FIRE compliance changed during Y2 of the project. For other testbeds, this document refers to D5.1 and D5.2.

Table 1: F4F compliance of WiSHFUL testbeds

	w-iLab.t (IMEC)	**ORBIT (Rutgers)**	**FIBRE Island @UFRJ**	**IRIS (TCD)**	**TWIST (TUB)**	**Portable testbed**
Discovery	SFA (GENI AMv3)	XML SFA Planned (GENI AMv3)	SFA (GENI AMv2)	SFA (GENI AMv3)	RESTful / SFA (GENI AMv3)	SFA (GENI AMv3)
Requirements	SFA (GENI AMv3)	XML SFA Planned (GENI AMv3)	SFA (GENI AMv2)	SFA (GENI AMv3)	RESTful / SFA (GENI AMv3)	SFA (GENI AMv3)
Instant Reservation	SFA (GENI AMv3)	Web based (XML)	SFA (GENI AMv2)	SFA (GENI AMv3)	Web based / SFA (GENI AMv3)	SFA (GENI AMv3)
Provisioning	SFA (GENI AMv3)	custom (reservation based exclusive access)	SFA (GENI AMv2)	SFA (GENI AMv3)	custom (reservation based exclusive access) / SFA (GENI AMv3)	SFA (GENI AMv3)
Experiment Control	SSH or FRCP (OMF6)	SSH or FRCP (OMF5.5 + OMF6.0)	SSH or OMF 5.4	SSH or FRCP	SSH/Custom	SSH FRCP (OMF6)
Facility Monitoring	Zabbix +OML	OpenNMS + OML	ZenOSS	OML	cacti + collected	None
Connectivity	Public IPv4 for AM Public IPv6 for resources	Public IPv4 + Firewalled IPv4/IPv6	Public IPv4 for AM Tunnel to private IPv4 for resources	Public IPv4 for AM Tunnel to private IPv4 for resources	Public IPv4 for AM, SSH Tunnel to private IPv4 for resources	Depending on location of installation
Documentation	OK	OK	OK	OK	OK	OK
Policies	OK	ORBIT+GENI	OK	OK	OK	OK
Federation Level	Fully F4F compliant	GENI	Fully F4F compliant	Fully F4F compliant	Fully F4F compliant	Fully F4F compliant

2.1 w-iLab.t (IMEC)

The w-iLab.t testbed is fully F4F compliant. Detailed information can be found in D5.1 section 2.6.

2.2 ORBIT (RUTGERS)

The ORBIT testbed is fully compliant with GENI. Detailed information can be found in D5.1 section 2.5.

2.3 FIBRE Island @ UFRJ

The FIBRE Island @ UFRJ is fully F4F compliant by the use of SFA AMv2. Detailed information can be found in D5.1 section 2.4.

2.4 IRIS (TCD)

The IRIS testbed is deployed at the CONNECT centre in Dunlop/Oriel House, Trinity College Dublin, Ireland. It offers virtualized radio hardware and software to support the experimental investigation of advanced research concepts in radio communication. The IRIS testbed combines flexible computational hardware and software resources with various hypervisors in the form of open source software radio frameworks such as IRIS, GNU Radio, and srsLTE (an open source version of the LTE standard developed by CONNECT researchers) to support the realisation of advanced radio research configurations. A high-level overview of the IRIS testbed is depicted in Figure 1.

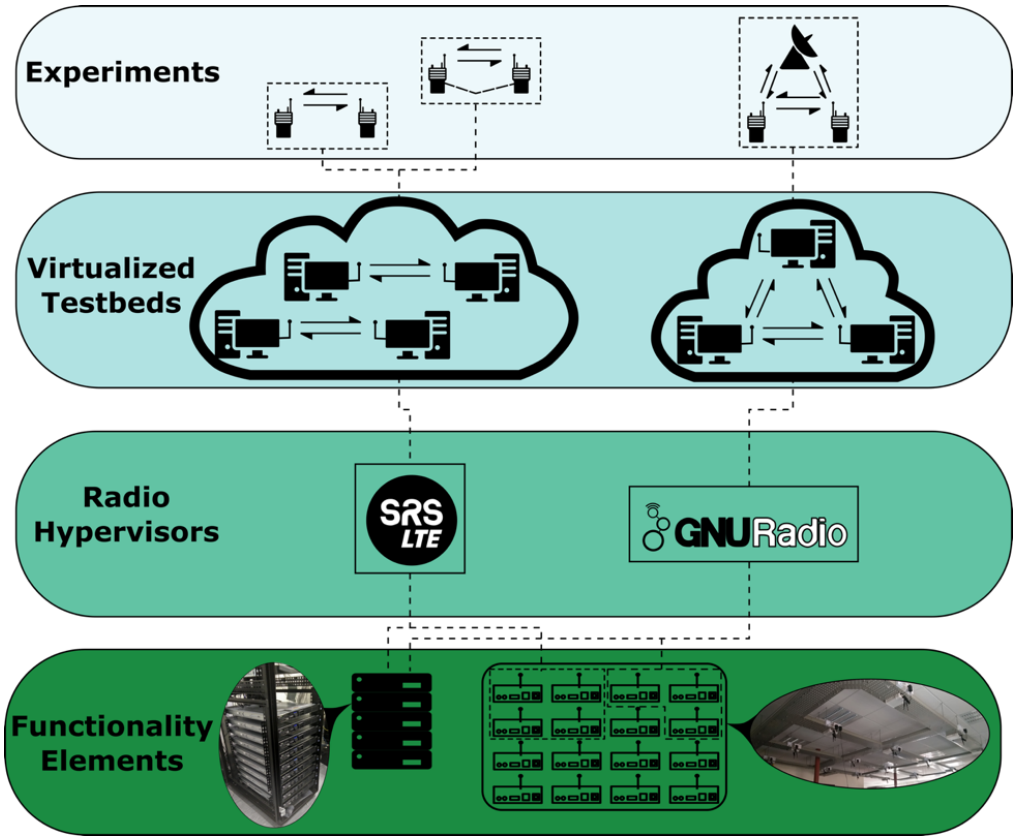

Figure 1 The functional layers of the IRIS testbed

There are four logical layers used in the IRIS testbed. The bottom layer consists of physical resources such as servers, N210 Universal Software Radio Peripherals (USRPs), networking equipment, storage, and so forth. These resources can be controlled by one of multiple hypervisor layers. The radio hypervisor layer offers SDR capabilities by providing support for either srsLTE, IRIS, or GNU Radio. Additionally, the virtualised testbed layer consists of virtual machines that are connected to physical radio units. Next, the experiment layer sits at the top and utilises the resources of the underlying layers. These layers provide the capabilities to construct a broad range of flexible radio systems.

Figure 2 IRIS radio grid at TCD's CONNECT Centre

The IRIS testbed consists of 16 USRP N210s ceiling mounted (see Figure 2) and equipped with SBX daugtherboards that can reach frequencies of between 40 MHz and 4 GHz as underlying radio resources. All 16 USRPs can be used for experiments. A typically experimentation unit is composed of Ubuntu 16.04, srsLTE supporting real-time radio reconfigurability, and a USRP for wireless communication. These units also include the following capabilities:

- virtual machine with 2 cores and 4GB RAM;
- a USRP N210 SBX120
- up to 20 MHz of Bandwidth per end-point
- carrier frequency from 400 MHz to 4.4 GHz

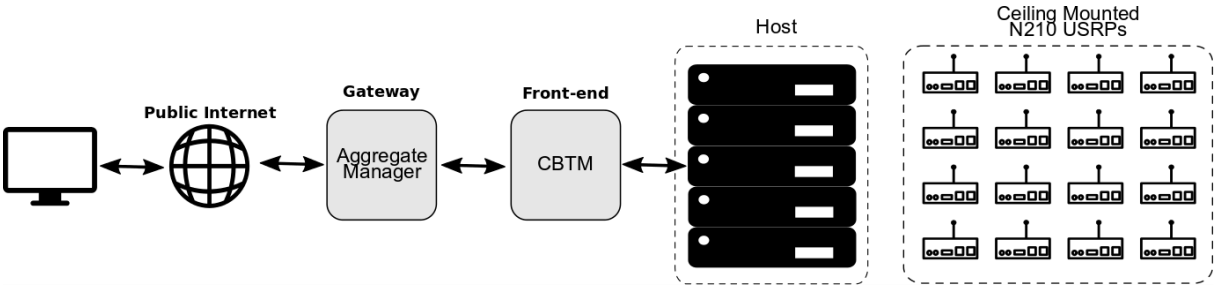

Figure 3 IRIS SFA 2.0 compliant testbed

2.4.1 SFA

The IRIS testbed Aggregate Manager is fully SFA 2.0 compliant. It uses a custom wrapper implementation of the GENI AG v3 API based on the reference GCF AG offered by geni-tools [1]. This resource runs on the IRIS testbed Gateway server. jFed clients communicate directly with the AM process for authentication, resource discovery, and provisioning. It interacts with the IRIS testbed Cloud Based Testbed Management (CBTM) process to set up Experimentation Units. The gateway server also runs a legacy reservation system, which is currently used by TCD staff and students to reserve testbed resources, which accesses the CBTM directly. A general view of the IRIS AG is available in Figure 3. This diagram also shows a high-level view of interaction with the testbed infrastructure.

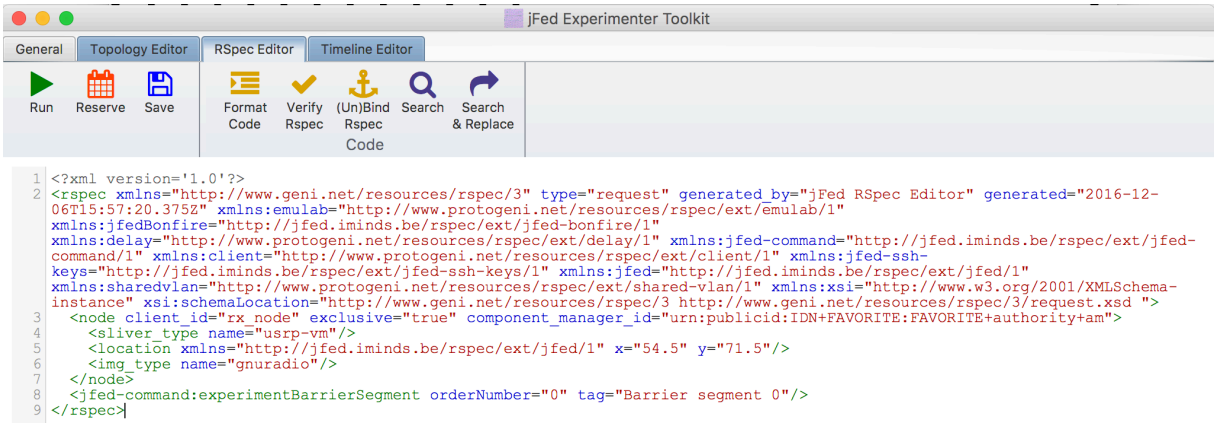

Figure 4 Sample IRIS RSpec file

2.4.2 Cloud Based Testbed Management (CBTM)

The CBTM v2.0 is a flexible resource that uses libvirt to interact with the KVM hypervisor to create virtual machines where experimenters can run experiment specific software. The CBTM protocol is responsible for interactions between the Aggregate Manager and CBTM resource supporting the allocate, delete, renew, shutdown, status checking, and so forth of experimental units based on RSpecs received. A sample IRIS RSpec file can be found in Figure 4.

2.4.3 Provisioning Process

The IRIS testbed has several different Ubuntu operating system images available – depending on the requirements of the experimenter. These include:

- Plain - Run your own software
- IRIS - Framework to develop SDR
- GNU Radio - Framework to develop SDR
- srsLTE - Open-source LTE library for SDR UE and eNodeB

Based on criteria specified in the RSpec received from the experimenter, the provisioning workflow works as follows:

- Install operating system image specified by the rspec
- Start and configure VM
- Add experimenter to VM i.e., the users public key
- After slice expires, delete the VM and release resources.

2.5 TWIST (TUB)

The TWIST testbed is fully F4F compliant. Detailed information can be found in D5.2 section 2.5.

2.6 Portable testbed

The portable testbed is fully F4F compliant. Detailed information can be found in D5.2 section 2.6.

3 Deployment and extension of existing Fed4FIRE tools

Fed4FIRE tools make use of three federation protocols: SFA, FRCP and OMSP. These protocols were described in detail in D5.1 chapter 3. For every federation protocol, a testbed can choose to support one of the client tools that implement these protocols. The table below shows all WiSHFUL testbeds and lists the tools that are supported on the testbed.

	w-iLab.t (IMEC)	**ORBIT (Rutgers)**	**FIBRE Island @UFRJ**	**IRIS (TCD)**	**TWIST (TUB)**	**Portable testbed**
SFA	jFed	XML/ jFed planned	MySlice / jFed planned	jFed	jFed	jFed
FRCP	OMF6	OMF6	OMF5.4	OMF6	SSH/Custom	OMF6
OMSP	OML	OML	OML	OML	Custom measurements	OML

For SFA, most testbeds make use of the jFed tool. FRCP is supported on most testbeds by the use of OMF6. Nearly all testbeds provide OMSP support by using OML. For every testbed a simple example is shown for every tool. This document can serve as a guideline for experimenters.

Based on feedback from open call experimenters, other tools might be chosen or modifications will be made to the existing tools.

Example configuration files for the tools listed in this chapter (jFed, OMF6, OML) are available through the WirelessTestbedsAcademy GitHub account, in the ExperimentationTools repository [2].

3.1 W-iLab.t (IMEC)

All F4F tools are supported by the w-iLab.t testbed. This has been extensively documented in D5.2 section 3.1.

3.2 ORBIT (Rutgers)

Usage of OMF & OML is described in D5.2 section 3.2.

3.3 FIBRE Island (UFRJ)

Usage of MySlice, OMF & OML is described in D5.2 section 3.3.

3.4 IRIS (TCD)

The IRIS testbed is fully SFA 2.0 compliant. To run an experiment, users need to register for a F4F account with the iLab.t authority [3]. The facility at TCD accepts all valid F4F certificates. Certificates issued by a F4F authority can access all testbeds in the federation. Experimenters can use the following Fed4FIRE tools to conduct an experiment on the IRIS testbed:

- Provision an experiment using jFed or jFed-cli tool (see section 3.4.1).
- Experiment control can be done using SSH or OMF6 (see section 3.4.2).
- Measurement collection with OML (see section 3.4.3).

3.4.1 Provisioning: jFed

jFed provides resource provisioning functionality in a drag and drop interface. TCD provides resources in the form of experimentation units, which consist of computational support and flexible radio hardware. Users have the ability to select a particular radio node and computational disk image. Once resources are provisioned, users will be able to double click an experimentation unit to gain SSH access.

The IRIS testbed uses the same provisioning process as IMEC, outlined in D5.2 section 3.1.1. The following steps describe how to use the IRIS testbed with jFed.

- Go to http://jfed.iminds.be
- Install the jFed Experimenter GUI
- Start the jFed Experimenter GUI. Provide the user certificate created when registering for a jFed account
- Click New.
- Drag two generic nodes onto the canvas. Double Click these nodes for additional configurations.
- Select Testbed: Iris TCD.
- In disk image, select plain, iris, gnuradio, or srslte.
- Experimenter can click “Run” to start the experiment. Enter a unique name for the slice. Click “Start Experiment”
- Once resources are provisioned, double clicking the node in jFed will open a terminal that gives full user SSH access to the nodes created.

An experiment tutorial that uses the IRIS testbed is available in [4]. In this tutorial two virtual machines, one for transmission and another for reception, are used. Two stages for this tutorial will be considered, one where a simple sinusoid wave is transmitted and observed over the air, and another where an OFDM signal is transmitted using the GNU Radio software.

3.4.2 Experiment Control: OMF6

Experiment control can be achieved with SSH. OMF support is also available with the OMF6 framework (OMF EC, RC and AMQP server). OMF Resource Controller is preinstalled on all nodes. Any node can act as an OMF Experiment Controller. Instructions to install the OMF Experiment Controller are available on [5]. The install guides for the OMF Experiment Controller are available at [5]. All communication is achieved by the AMQP messaging server, which is accessible at amqp://134.226.55.214. All experimenters are required to use OEDL file for experiment specification.

3.4.3 Measurements: OML

The IRIS Testbed supports OML by offering a publicly available OML server (tcp:134.226.55.214:3003). All testbed nodes are preinstalled with the OML client library.

3.5 TWIST (TUB)

The specific implementation details of the F4F tools on the TWIST testbed have been described in D5.2 section 3.5.

The TWIST testbed received the following improvements in the support of federation protocols in Year 2.

- Easy selection for testbed supported disk images for all platforms. It is now possible to select desired image directly from jfed. Testbed currently supports:
  - [NUC] An ubuntu xenial (16.04)
  - [NUC] An ubuntu precise (12.04) (as part of the Open Call support)
  - [NUC] A 'wishful enabled' ubuntu xenial (16.04)
  - [TPLink] An OpenWRT Chaos Calmer
- Documentation has been extended with a tutorial how to build own custom images supported by the testbed.
- Jfed support to reboot nodes on user demand
- Graylog log message collection server has been deployed to store full information about testbed operations from all servers and nodes in the testbed.
- Various bug and stability improvements in the internal functioning of the testbed.

3.5.1 Creating custom disk images

One of the most requested feature by open call experimenters was the ability to use their own kernel version and system for experiments. That also adds a requirement to not only provide ability to deploy such image, which was already supported in TWIST since initial Fed4FIRE compliance development, but also ability to prepare the own images by experimenters. Such image needs to fulfil two sets of requirements. First, it needs to be compatible with deployment and configuration strategies of the TWIST testbed. Second, it needs to be customized to experimenter requirements.

Usage of custom images indicates already advanced requirements from the experimenter. In most cases, it is enough to install additional software just before running the experiments, for example by using Ansible to automate experiment deployment phase.

We provide instructions [6] and preparation scripts [7] for two platforms:

- TP-Link TL-WDR4300 Routers, where the main operating system is OpenWRT. In this case we propose to compile the whole system from scratch. We provide configuration files for the OpenWRT system and kernel.
- Intel NUC Embedded PCs, where the default operating system is Ubuntu. We provide a `debootstrap` based method to prepare new system configuration. Default one is based on Ubuntu 16.04, which is the newest long term support release of the system. We also provide configuration for 12.04 (precise) release, which was requested by one of the open call experiment partners.

In fact, the same scripts are used for preparation of default images. In order to make the whole build system more robust we use Ansible as a general build tool. In order to use it experimenters need to have Ansible up and running on the build machine. Those Ansible playbooks take care of collecting necessary source code, building the image, configuring it and in the end delivering ready to use compressed image file. For usage in the testbed such image has to be posted on the publicly available HTTP server. Link to the image has to be provided in the jfed configuration. Note, that services like Dropbox allow for public sharing of the files, which make them a possible hosting service.

We provide more details on how to build custom images in the testbed documentation and repository with Ansible playbooks (that are both publicly available).

3.5.2 Testbed logging functionalities

TWIST testbed has been extended with graylog [8] based log management service. It is a testbed internal function, but provides great help in debugging and management of the whole testbed with multiple nodes. It collects all logging data from all servers and nodes in the testbed in one place and allows for further automatic analysis and processing.

Figure 5 shows example screen from TWIST instance of graylog. In this example we see only messages indicating failures in the deployment process. Those messages are automatically filtered by the server and thus don't require user interaction. Those messages are also automatically forwarded to a testbed management team, allowing them to react accordingly.

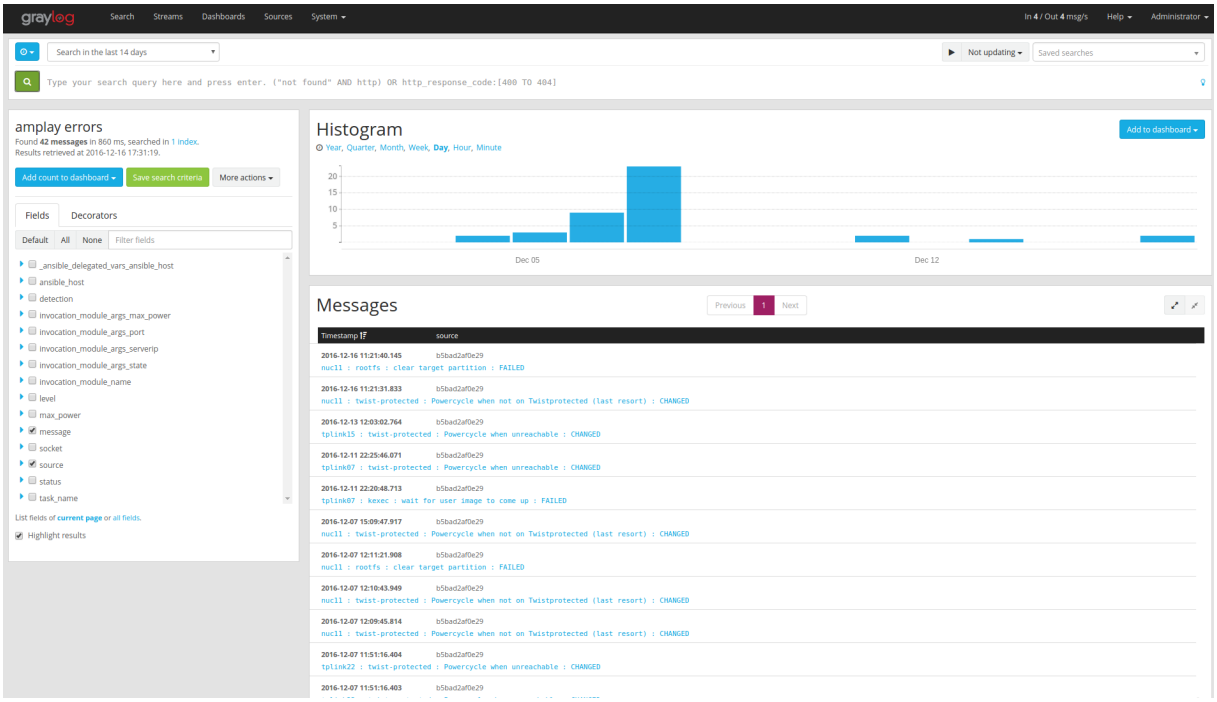

Figure 5 Example graylog output

With the help of netconsole kernel module it is even possible to stream low level system boot logging information, even before usual file systems are mounted. This functionality requires support for UDP based syslog data collection, as usually the TCP stack is not yet fully loaded in the kernel. It allows to get information from the initial booting sequence and is extremely important while preparing a new kernel or image versions for remote and embedded devices that normally don't have a screen connected. Without netconsole support it would not be possible to remotely debug kernel boot failures as there would be no ssh access to the device. Merging both netconsole and graylog functionalities allows for such remote debugging. This functionality is now deployed and available in TWIST testbed for both OpenWRT based routers and Ubuntu based devices.

Collecting all logging information in one place greatly improves the experience of debugging the testbed as a whole. Current graylog instance in TWIST testbed is logging and analysing over 10k log messages per hour.

3.6 Portable testbed

All F4F experimentation tools are supported by the Portable testbed. This has been extensively documented in D5.2 section 3.6.

4 FED4FIRE compliant testbed extensions

This chapter describes the hardware extensions that were made to WiSHFUL testbeds during the last year.

4.1 W-iLab.t (IMEC)

Before the start of the WiSHFUL project, several wireless technologies were available in the w-iLab.t testbed. Below is a short summary of the testbed hardware and the wireless technologies they enable.

- 60 embedded nodes (ZOTAC/DSS)
  - 2x 802.11a/b/g/n
  - 1x Bluetooth 2.0
  - 1x 802.15.4 (RM090)
- 17 mobile nodes (DSS)
  - 1x 802.11a/b/g/n
  - 1x 802.11ac
  - 1x Bluetooth 2.0
  - 1x 802.15.4 (RM090)
  - 1x LTE usb stick
- 2 LTE femtocells
- SDR hardware
  - 8x USRP
  - 3x WARP

Figure 6 gives an overview of the available hardware in the w-iLab.t testbed prior to the extensions.

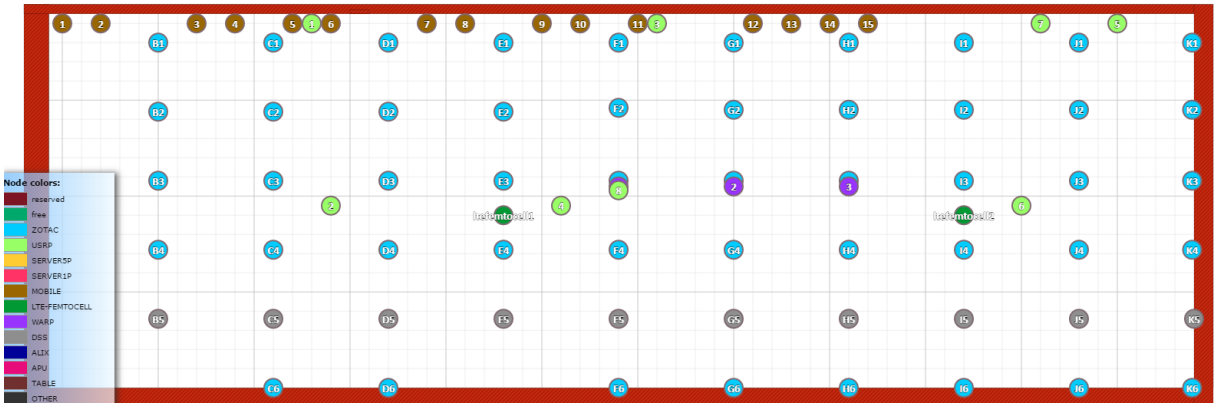

Figure 6: w-iLab.t hardware before extensions

Based on feedback from experimenters, some extensions were installed in the w-iLab.t testbed. The goal of these extensions was to increase the number of wireless nodes, as well as to support more different technologies. Below is a short summary of the hardware that was added to the testbed as part of the extension.

- 40 embedded nodes (APU)
  - 2x 802.11ac
  - 1x 802.15.4 (Zolertia Re-Mote)
  - 2 wired Ethernet interfaces (can be used as private network by the experimenter)

- 1x LTE usb stick (10 in total)
- 1x Adant RAS antenna (result of OC1)
  - 20 nodes are equipped with 2.4GHz RAS antenna
  - 20 nodes are equipped with 5GHz RAS antenna

- 14 embedded nodes (ALIX)
  - 1x 802.11b/g (Broadcom cards to support CNIT Wireless Mac Processor)
- 15 mobile nodes were equipped with smartphones (Android Nexus 6P) to support LTE experiments (see Figure 8).

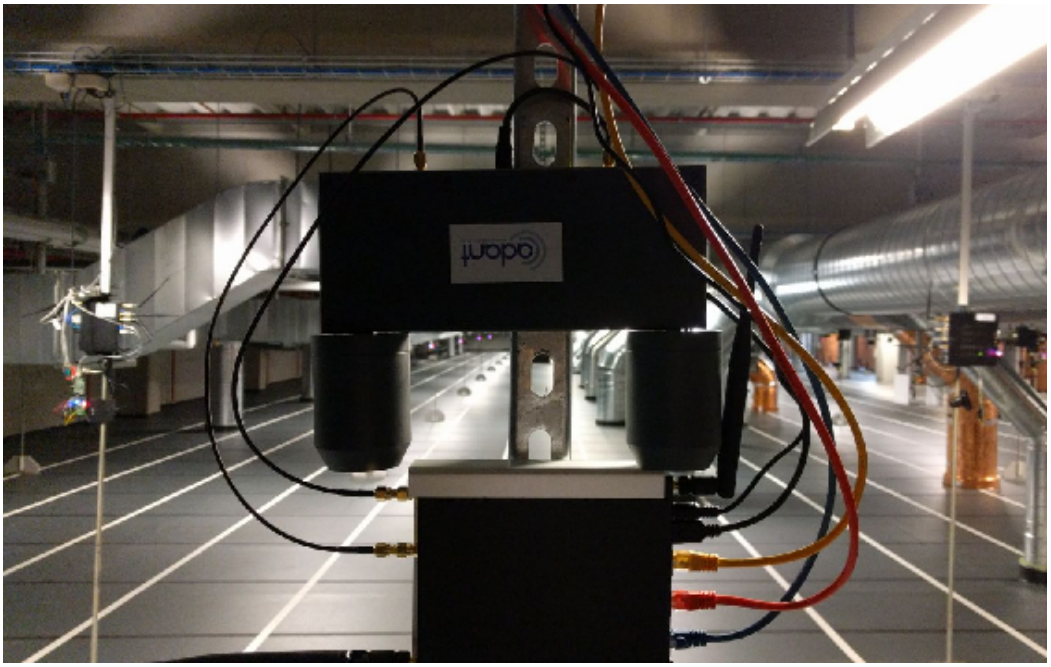

Figure 7: APU node with Adant RAS antenna

Figure 8: Smartphone deployed on mobile node in w-iLab.t

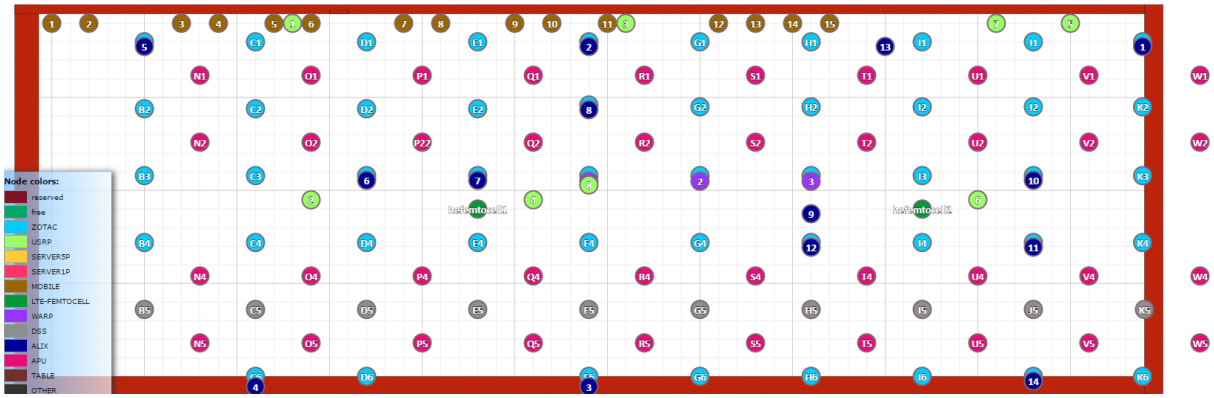

Figure 9: w-iLab.t hardware including extensions

Because of the intensive usage of the w-iLab.t testbed, a secondary location was equipped with wireless hardware. This new testbed was deployed in the data centre (30x12m) of the iGent building in Ghent, Belgium. Below is a short summary of the hardware that was installed. Figure 10 shows the topology of the deployed nodes in pink.

- 44 new embedded nodes
  - Intel NUC D54250 (similar to the Portable testbed nodes)
    - Intel i5-4250U (2.6GHz), 8GB DDR3 RAM, 1x Gigabit LAN
  - 1x 802.11a/b/g/n (2x2 MIMO) + Bluetooth 4.0 (BLE)
  - 1x 802.11ac (3x3 MIMO)
  - 1x 802.15.4 (Zolertia Re-Mote)

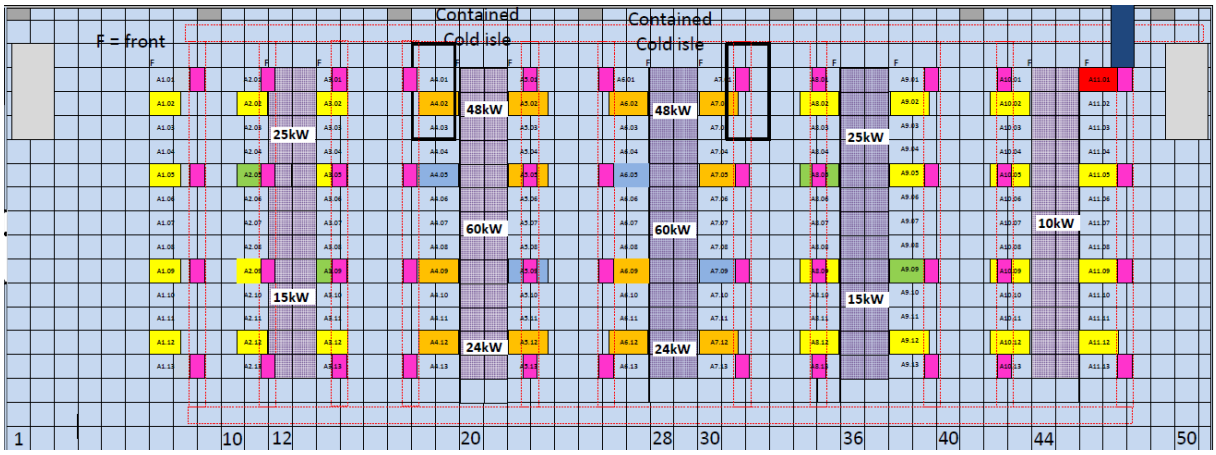

Figure 10: w-iLab.t deployment at data centre iGent

This new deployment of the w-iLab.t testbed is fully Fed4FIRE compliant, as it supports all federation protocols like SFA, FRCP and OMSP and thus supports the F4F experimentation tools like jFed, OMF6 and OML.

Figure 11: w-iLab.t iGent wireless nodes

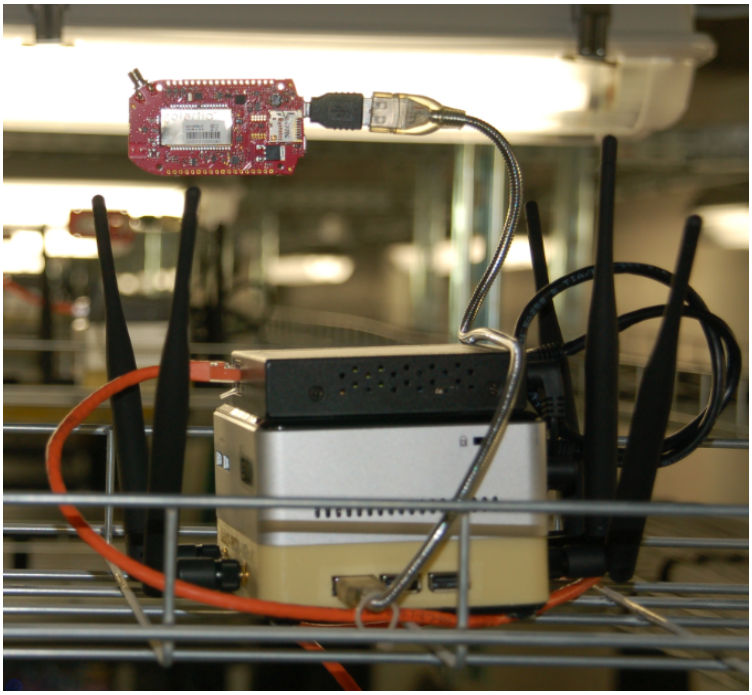

Figure 12: w-iLab.t iGent close-up

4.2 ORBIT (Rutgers)

In the second year, relevant extensions to the ORBIT testbed include:

- LTE deployment in ORBIT SandBox
- Outdoor LTE base stations deployment
- Development of massive MIMO rack
- Development of cloud RAN extension

Examples of experiments to be supported by the new LTE capability include evaluation of WiFi/LTE coexistence in dense environments, or the study of content delivery techniques to LTE-based devices in both emulated and real-world settings. The introduction of radio cloud upgrade is focused on supporting new classes of experiments including cooperative spectrum sensing, distributed MIMO or similar cooperative processing functions, centralized radio resource management and low-latency cloud processing.

4.2.1 LTE Deployment

During the last year significant effort was invested in expanding the experimentation capabilities with LTE radio technology which is becoming ubiquitous as the next-generation radio air interface for cellular systems worldwide. In addition to platforms that were previously available for LTE experimentation, the testbed was expanded with a number of production grade small and medium coverage basestations. The two models (deployed as part of GENI Wireless project), the Airspan AirHarmony 1000 and Airspan AirSynergy 2000 (Figure 13) were deployed in sandbox 4 and outdoor domains.

Figure 13: AirSpan LTE Basestations and Client Devices

a. Sandbox (SB4) LTE Deployment

Two AirHarmony 1000 basestations were deployed in ORBIT sandbox 4 (Figure 14) to support LTE experiments. Given that these are fairly high output power units meant for outdoor installation, the two antenna ports are each passed through 30dB 50 Watt attenuators before the combiner which connects to an RF switch. The switch allows the user to select between having a basestation (LTE or WiMAX) or a node connected to the JFW attenuator matrix.

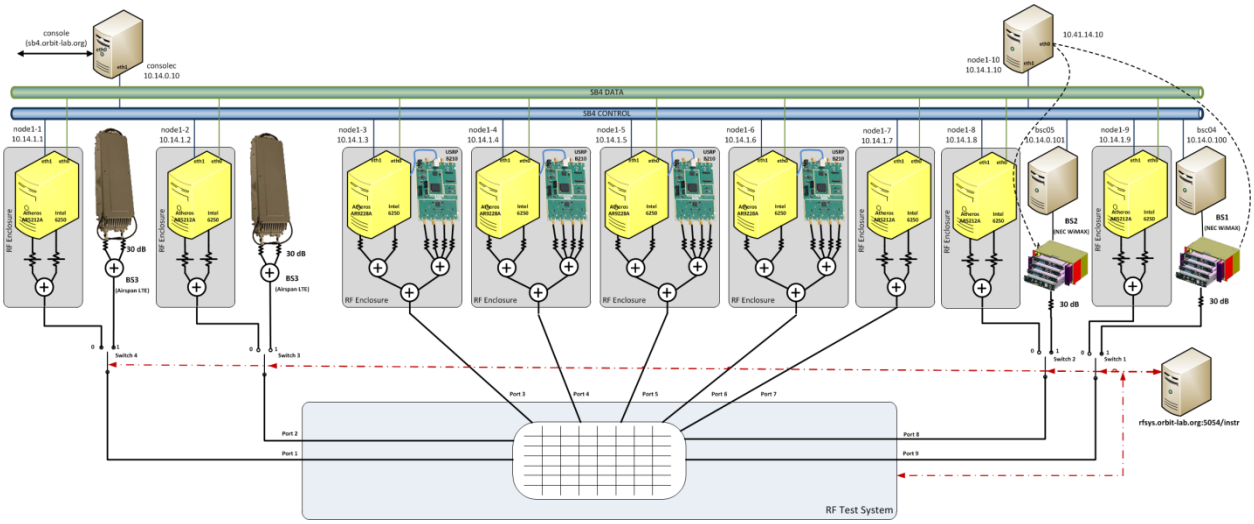

Figure 14: ORBIT Sandbox 4

Of the 9 RF capable nodes in sandbox 4, up to four can be "replaced" by the experimenter with LTE or WiMAX base stations as desired. For example, the testbed can be reconfigured easily as 9 nodes, 8 nodes and an LTE base station, or 5 nodes and 2 WiMAX and 2 LTE base stations. The configuration is done via a simple REST call to the instrumentation service (OMF AM service) which will instruct the network connected RF switch to toggle one (or more) of its matrix attenuator facing ports to be connected to either a node or a base station. This allows for an experimenter to change the topology of the testbed in terms of combinations of nodes and base stations without necessitating physical modification of the testbed. Since the configuration of the RF switches is handled programmatically, not only can the topology of a sandbox be easily customized but it can also be reconfigured mid experiment by coordinating the capabilities of the JFW matrix attenuator.

b. Outdoor LTE Deployments

As part of larger GENI wireless deployment, project one Airsynergy 2000 base station was deployed on each of the two Rutgers campuses that are part of ORBIT outdoor deployment. The two base stations were deployed with 2.3-2.7GHz 90deg 17dBi Dual X-Polar Antennas.

4.2.2 ORBIT Massive-MIMO Extension

The ORBIT massive-MIMO extension design is based on the concept of wheeled mini-rack with four racks deployed in four corners of the grid. This allows for easy reconfiguration of the setup to support experiments with varying number of antennas as well as varying number of array locations at the expense of wiring infrastructure.

This four racks deployment is enabling a number of interesting topological combinations including: a.) one corner with large number of antennas, b.) two corners with modest number of antennas aimed at experimentation with point-to-point massive-MIMO type of systems, and c.) four corners with smaller number of antennas each for system testing. Number of existing SDR units that are part of the ceiling nodes are intended to be used as clients. Each rack is connected by 2 RF cables for timing synchronization, 16 duplex multimode fibers for backhaul to the SDRs, 2 cat6 Ethernet jacks for management, and a 20A 120v circuit for power. Installed timing cabling is equal length, others are asymmetric length as needed. Quantity of cables run depends on corner, as one corner has support for all 4 racks, one has support for 2 racks, and the remaining two support a single rack each. All of

the 64 higher-end SDR units in ORBIT can now be fully synchronized with a newly introduced reference clock distribution system that is based on Ettus Octoclock hardware (Figure 15).

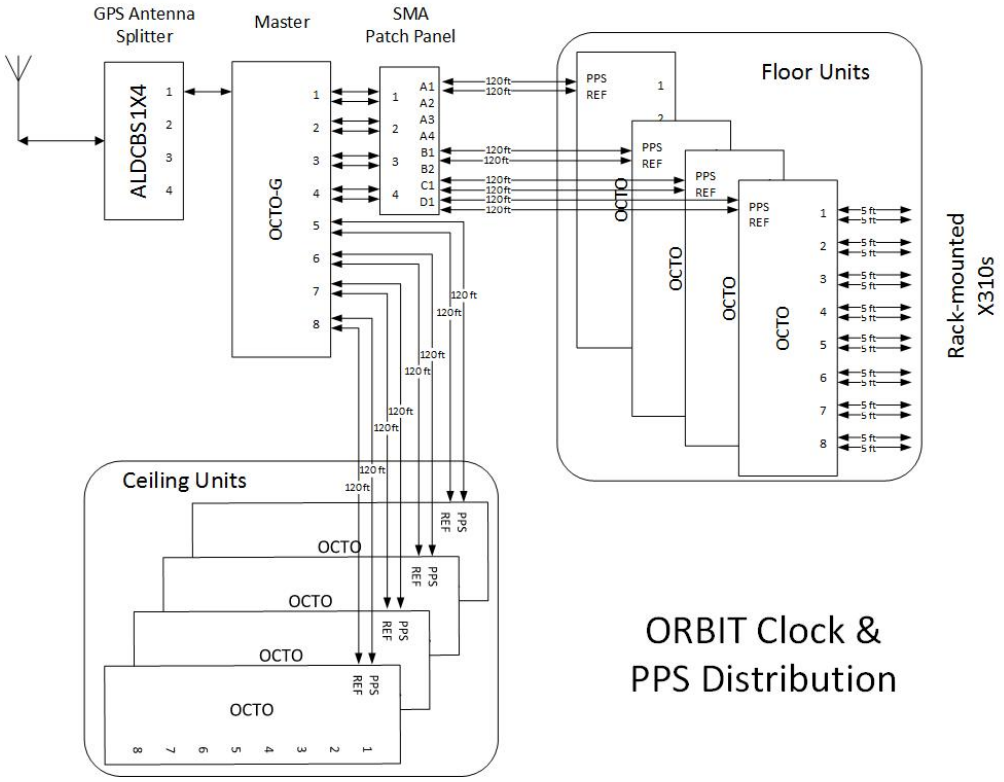

Figure 15: RF Synchronization Subsystem

4.2.3 Mini-rack Massive MIMO

The Massive-MIMO setup is based on a concept of mini-rack. That contains 8 Ettus USRP X310 software defined radios outfitted with two UBX 160 radio daughterboards each as shown in Figure 16. Unlike USRPs in the main 20x20 Grid which are directly attached to individual nodes, each USRP in the mini-rack is connected via 2x 10G Ethernet fiber to the top-of-the-mimo switch that is in turn directly connected to the top ORBIT aggregation switch. PPS and 10MHz synchronization signals are distributed among the USRPs in each rack via an Octoclock based timing distribution system. Equal length cables are used to ensure minimum timing deviation within and between the racks allowing for all USRPs across the four racks to by synchronized. Additional hardware is present in each rack to allow for power-cycling and "imaging" of the USRPs in a similar manner to standard nodes.

Figure 16: ORBIT Massive MIMO Mini Rack

4.3 FIBRE Island @ UFRJ

No changes are reported for the FIBRE testbed at UFRJ during Y2.

4.4 IRIS

The IRIS testbed received the following hardware extensions in Year 2.

- External HDs and USB Connector for backing up rack servers
- Power Adapters to support WiSHFUL Demos
- Two TERRATEC Cinergy S2 USB - Digital TV tuners
    - Connections: 1 x USB, Included Accessories: Remote control, Cables Included: 1 x USB cable, OS Required: Windows XP SP3, Windows Vista (32/64 bits), Windows 7 (32/64 bits) SP1, 8 (32/64 bits).

Figure 17 TERRATEC Cinergy S2 USB - digital TV tuner

4.5 TWIST (TUB)

Second Wi-Fi interface based on Qualcomm Atheros AR928X chipset with external antennas was added to all NUC's. That means that currently Intel NUC devices are equipped with two cards, where each allows for operation in 2.5GHz and 5GHz bands. For practical reasons one card is using an internal antenna and the second custom mounted external antennas (two per card). This has been achieved by drilling new antenna holes in the standard Intel NUC box, which didn't require adding additional enclosures.

Furthermore, all Intel NUC devices have been equipped with Metageek Wi-Spy 2.4x devices, that allow for easy spectrum monitoring.

4.6 Portable testbed

The hardware extensions to the portable testbed are documented in D6.4 and are therefore not repeated here.

5 Development & Fed4FIRE compatibility of new tools

5.1 Tools for the integration of WiSHFUL UPIs in testbeds

This chapter describes the tools that are used to allow easy integration of WiSHFUL software onto the FED4FIRE compliant testbeds. Several major modifications were made to the approach proposed in D5.2, based on valuable feedback from the experimenters. To be able to describe the entire experiment flow and the integration of the WiSHFUL UPIs into this flow, some of the information from D5.2 is repeated in these sections.

Conceptually, the integration can still be seen as depicted in Figure 18.

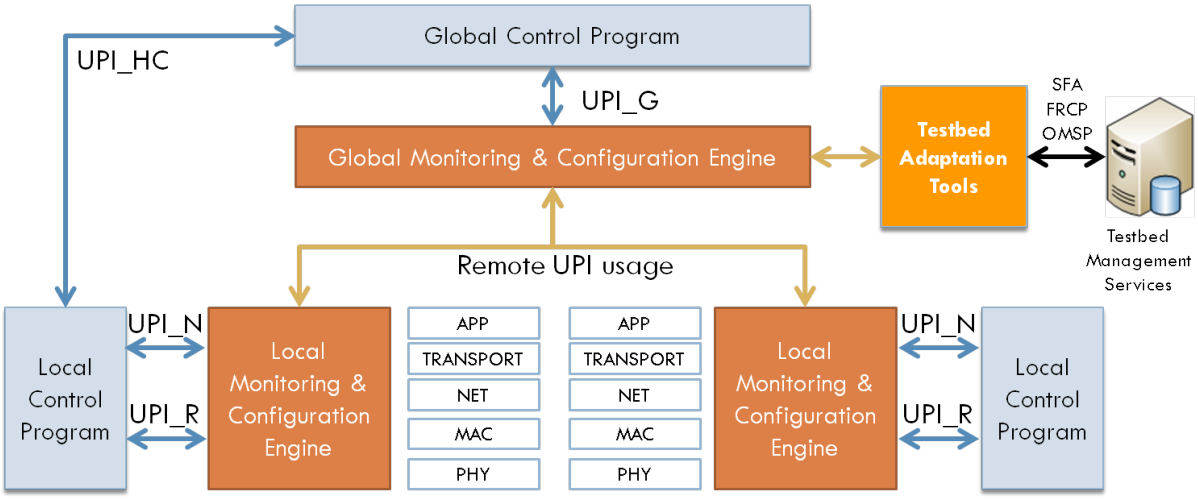

Figure 18 WiSHFUL UPIs in testbeds

Because all Fed4FIRE compliant testbeds have a specific implementation (e.g. Rspec extensions) for the Fed4FIRE interfaces (SFA, FRCP, OMSP), testbed adaptation tools need to be developed that offer a generic interface to the global monitoring and configuration interface.

The following abbreviations are used below:

- Global MCE: Global Monitoring & Configuration Engine
- Local MCE: Local Monitoring & Configuration Engine

The following requirements were identified as being crucial for successful integration of WiSHFUL UPIs in testbeds:

- Node discovery (see section 0): The Global MCE requires knowledge on the number and type of nodes that are part of an experiment. This information should not require manual interaction from the experimenter. Note that the testbed node discovery is supplementary to the WISHFUL framework node discovery and offers the global MCE a view on the available testbed nodes which might not all be required to conduct the WiSHFUL experiment.
- Experiment setup (see section 5.1.2): The installation of Global and Local MCE's on testbed nodes should be automated.
- WiSHFUL control (see section 5.1.4): All tasks regarding WiSHFUL control should be cleanly separated from end-user applications (experiment control).
- Monitoring & measurements (see section 5.1.5): OML (OMSP) can be used for gathering monitoring results from each testbed node. It would be nice if WISHFUL control programs could use the OML library for logging information, allowing to easily evaluate and demonstrate the impact of the control program logic.

The following sections describe every step in the experiment lifecycle and give a detailed description of how the WiSHFUL UPIs can be cleanly integrated into this workflow.

5.1.1 Node discovery

SFA can be used for resource discovery, e.g. knowing which nodes are in an experiment. Using the resource descriptions advertised by the testbeds AM (in Rspec format) the type of nodes and their capabilities can be retrieved. JFed can be used to select & provision the discovered resources (nodes) that are needed to conduct the experiment.

5.1.2 Experiment Setup

Scripts can be configured in jFed so that on start-up extra software is installed on the nodes. If this software is complex to install and depends on lots of libraries, tools like Ansible could be used. This feature will be exploited to install the software components for the following control functionalities:

- Experiment Control (see section 5.1.3): An Ansible script is provided to the experimenters to install an OMF Experiment Controller (on one node) and OMF Resource Controllers on all other nodes.
- WiSHFUL Control (see section 5.1.4): An Ansible script is provided to allow installation of WiSHFUL software onto testbed nodes. This might install the Global MCE on one node and Local MCE on all other nodes.

In contrast to the integration approach which was described in D5.2, it is now no longer required to use OMF in order to do WiSHFUL experimentation. This reduces the amount of new frameworks to which the user has to become acquainted. Also, as OMF6 is written in Ruby and WiSHFUL in Python, it becomes overly complicated for the experimenter. Of course, the experimenter can still chose to run both OMF6 and WiSHFUL in parallel if the experiment requires it. However, it is no longer a strict requirement. Furthermore, users of OMF6 can also benefit from the approach proposed below.

a. Ansible support in jFed

jFed (v5.7) now natively supports the Ansible framework to allow easy installation of complex software packages. When using Ansible on testbeds, the most common approach is to allow one of the nodes in the experiment to act as Ansible controller. To turn a normal node into an Ansible controller, add the following Rspec tags to a node:

```
<services>
      <install install_path="/local" url="http://doc.ilabt.iminds.be/jfed-
documentation-5.7/_static/install-ansible.tar.gz" />
      <execute shell="sh" command="cd /local &amp;amp;&amp;amp; sudo /bin/bash
install-ansible.sh"
jfed:finished_flag="/tmp/ansible-install-finished"/>
</services>
```

To allow the Ansible controller (one node of the experiment) to log into all the other nodes in the experiment over SSH, to be able to install all software listed in the Ansible playbook, add this line:

```
<jfed:distribute_ssh_keypair />
```

jFed also has the ability to automatically create an inventory file of the experiment. In the inventory file, different groups can be created. The experimenter can decide which software will be installed on nodes of a certain group. For example: one group could be created for all nodes on which a WiSHFUL Local MCE should be installed. Another group could be created for one node where the Global MCE

should be installed. To allow the creation of groups in the inventory file, jFed supports a new *ansible-group* tag in the Rspecs. The following code fragment shows the Rspec for a node in the w-iLab.t testbed that is part of the WISHFUL Local MCE group.

```
<node client_id="accesspoint1" exclusive="true"

component_manager_id="urn:publicid:IDN+wilab2.ilabt.iminds.be+authority+cm"
>
       <sliver_type name="raw-pc"/>
       <jfed:ansible_group name="wishful_local_mce"/>
</node>
```

The resulting inventory file would then look like this:

```
[wishful_local_mce]
accesspoint1  ansible_host=zotacB1.wilab2.ilabt.iminds.be
```

The same node can be part of several different groups. This functionality can be used by the OMF EC and the WiSHFUL Global MCE to get a list of all nodes in the experiment (or only nodes belonging to a certain logical group).

By using the following Rspec tag, the user can specify in which directory the inventory file should be uploaded:

```
<jfed:ansible inventory="/local/my_repo/playbooks">
```

b. Example Ansible playbooks

Several example Ansible playbooks are available in the ExperimentationTools directory of the WirelessTestbedAcademy [2]. Playbooks are available for the installation of:

- OMF6 Experiment controller
- OMF6 Resource controller
- OML Measurement Server
- WiSHFUL Global MCE
- WiSHFUL Local MCE

To execute a specific Ansible playbook, some extra Rspec additions are required. The example below shows the commands required to run the *wishful.yml* playbook on the nodes defined in the inventory file *wishful_hosts.*

```
<jfed:ansible
       galaxy-command="sudo ansible-galaxy"
       install_requirements="/local/my_repo/playbooks/ansible-
requirements.yml"
       inventory="/local/my_repo/playbooks/wishful_hosts"
       playbook-command="sudo ansible-playbook"
       execute_playbook="/local/my_repo/playbooks/wishful.yml”
       debug="true"
 />
```

5.1.3 Experiment Control

OMF6 is used to define the functionality of the nodes in the experiment and to construct a timeline (time and/or event based) of the experiment. In practice, this will result in giving the testbed resources a meaningful name and assigning applications to run on them. It is possible to use the information from the Ansible inventory file (see 5.1.2a) in the OMF experiment description.

The experiment script can then filter the list of resources in the inventory file based on the group they belong to. Below is an example inventory file that could be used by OMF to configure two nodes as wireless access point, while the other two nodes are configured as clients. Note that the OMF RC should be installed on all nodes, which is why all nodes are member of the OMF_RC group.

```
[omf_ec]
expcontroller  ansible_host=dssK5.wilab2.ilabt.iminds.be

[omf_rc]
accesspoint1  ansible_host=zotacB1.wilab2.ilabt.iminds.be
accesspoint2  ansible_host=zotacB2.wilab2.ilabt.iminds.be
client1  ansible_host=zotacB3.wilab2.ilabt.iminds.be
client2  ansible_host=zotacB4.wilab2.ilabt.iminds.be

[accesspoints]
accesspoint1  ansible_host=zotacB1.wilab2.ilabt.iminds.be
accesspoint2  ansible_host=zotacB2.wilab2.ilabt.iminds.be

[clients]
client1  ansible_host=zotacB3.wilab2.ilabt.iminds.be
client2  ansible_host=zotacB4.wilab2.ilabt.iminds.be
```

5.1.4 WiSHFUL Control

To the experimenter, the WiSHFUL software can be considered as extra functionality that can be included in his experiment to allow for more advanced control strategies.

The global and local control programs define the network/radio control logic. The Global MCE should be able to retrieve the nodes in the experiment. In contrast to the approach proposed in D5.2 section 5.1; this approach no longer requires OMF to be used to start WiSHFUL control on testbeds. In this approach, OMF and WiSHFUL can be used as two separate frameworks. Depending on the needs of the experimenter, a combination of both frameworks can be advised.

It is possible to use the information from the Ansible inventory file (see 5.1.2a) in the WiSHFUL Global MCE. The Global MCE can then filter the list of resources in the inventory file based on the group they belong to. Below is an example inventory file that could be used by WiSHFUL to configure two nodes as wireless access point, while the other two nodes are configured as clients. Note that the WiSHFUL Local MCE should be installed on all nodes, which is why all nodes are member of the WiSHFUL_Local_MCE group. Only one node will act as WiSHFUL Global MCE. To simplify the inventory file, we use the groups-of-groups functionality in Ansible.

```
[wishful_global_mce]
globalcontroller  ansible_host=dssK5.wilab2.ilabt.iminds.be

[wishful_local_mce:children]
accesspoints
clients

[accesspoints]
accesspoint1  ansible_host=zotacB1.wilab2.ilabt.iminds.be
accesspoint2  ansible_host=zotacB2.wilab2.ilabt.iminds.be

[clients]
client1  ansible_host=zotacB3.wilab2.ilabt.iminds.be
client2  ansible_host=zotacB4.wilab2.ilabt.iminds.be
```

Note that adding nodes to a group can easily be done by using the a*nsible-group* tag in jFed. All topology information is thus constructed by jFed and can afterwards be queried by other frameworks such as OMF and WiSHFUL. Topology changes will therefore not affect the experiment description in OMF, nor will they change the global control program. This is because selecting a different physical

resource will not change the logical groups formed in the inventory file. Also running the same experiment on different testbeds should be significantly simplified by this approach.

5.1.5 Monitoring & measurements

The OMF experiment controller can define events based on measurement information stored by OML. By default, OML clients send their data to an OML server, which stores the data into a database. The OMF EC can query this database and then act if the measurements exceed a certain threshold. However, the delay that is introduced by this method might be too high to quickly react to changing situations.

To solve this issue, the WiSHFUL Global MCE will serve as an OML proxy server. This way, it will be able to parse all data before passing it on to the real OML server. The Global MCE will not store the data in a database, but keep the last received measurements available in RAM (e.g. Python associative array). Using this solution, no modifications are needed to the OML framework. Instead, the WiSHFUL Global MCE will be extended to support parsing and forwarding of OML measurement streams. The OML server now serves as a pure measurement logger.

The measurement logger library was added on WirelessTestbedAcademy under EperimentationTools/measurement_logger. Beside support for OML, also measurement loggers were created that log data into a CSV file, MySQL database, a log file or to the standard output. Information about the type of measurements and the active logger type is defined in a YAML measurement configuration file. This information should be passed to the WiSHFUL control program at start-up. The library will then spawn and prepare the correct logger based on the YAML input.

5.1.6 Summary

This section serves as a summary by listing all the steps in the experiment life cycle assuming WiSHFUL is used as the experiment control framework. Therefore OMF is not mentioned in this experiment flow.

- The experimenter uses his F4F certificate to start jFed.
- The experimenter chooses a selection of resources from a certain testbed, assigns them to different logical groups and activates his experiment using jFed.
- By using simple Ansible install scripts, the user enables WiSHFUL control on the nodes in his experiment. The install scripts are automatically executed by jFed at the start of an experiment.
- The WiSHFUL Global and Local MCEs are automatically started by jFed when the installation is finished.
- The user creates/modifies/activates the WiSHFUL Global and/or Local Control Programs. Information about the logical groups of the nodes can be queried from the inventory file.
- OML monitoring/measurement streams originating from end-user applications or Local MCEs are parsed by the Global MCE before being forwarded to the OML server. The Global MCE can take decisions based on thresholds defined for these measurements.

5.2 SDR Frameworks

Software-Defined Radio (SDR) refers to platforms where wireless signal processing is performed by software modules. A detailed description of SDRs is available in D2.4 section 2.2. The two most common SDR frameworks utilised by the SDR development community and available to WiSHFUL users include: GNU Radio and IRIS.

5.2.1 GNU Radio

GNU Radio is an open source and free to use SDR framework that is flexible and adaptive for advanced wireless research, experimentation and education [4]. It is written in C++ and Python and has a very large development community with an active user support network. It offers a simple reusable model for developing new components allowing experimenters to build software radio and signal processing applications. This is achieved by writing new signal processing blocks and supporting the reuse of existing blocks. Blocks are joined into connected flowgraphs through which signal data flows are processed. One of the most appealing aspects of GNU Radio is that open source components developed by other experimenters can be easily be incorporated into new experiments, which can reduce experiment setup, development and testing time.

GNU Radio is currently used by the IRIS testbed at Trinity College Dublin for advanced experimentation and also education. A detailed experiment scenario showcasing of some of GNU Radio's capabilities in Radio Virtualisation is outlined in D2.4 section 2.2. The GNU Radio WiSHFUL UPIs utilised by the Radio Virtualisation experiment are outlined in D2.4 section 2.2.1.

5.2.2 IRIS

The IRIS SDR system is flexible and adaptive open source framework for advanced software radio research [9]. It is a component-based/modular-based system that supports dynamic runtime reconfiguration, which is written in C++. A component is an element or stage in the signal-processing chain. The radio designer determines each components level of functionality and complexity. A radio can be created using a combination of existing or new radio components by experimenters. Components are key to IRIS's reconfigurability and support reuse among different applications. Complex flow-graph structures are connected together to support radio transmit and receive chains, which are unambiguously represented using Extensible Markup Language (XML) [9]. Human readable XML supports a simple interface for external reconfiguration of runtime radios.

IRIS is currently used by the IRIS testbed in Trinity College Dublin for advanced radio experimentation and education. Since its inception in 2004, there have been many experimental SDR applications using IRIS including [9]:

- Using OFDM waveforms in a TV white space network,
- Using novel physical (PHY)-layer signalling techniques for network rendezvous and coordination in dynamic spectrum access networks,
- Using a Cell Broadband Engine (CellBE) platform implementing real-time cyclostationary analyser.

The IRIS WiSHFUL UPIs are outlined in D2.4 section 2.2.2. A experiment scenario showcasing of some of IRIS's WiSHFUL capabilities is outlined in D2.4 section 2.2.

5.3 Tools for sensor experiments

This section describes two tools to facilitate sensor experiments. The first subheading describes how to use the TAISC parser to create a single sensor binary, which contains all required libraries. The second subheading describes the usage of a Logic Analyzer to enable TAISC MAC debugging.

5.3.1 TAISC Parser as a single binary

The TAISCParser is a Python script which generates byte code, which can be interpreted by a TAISC Virtual Machine on a sensor node, from a TAISC Chain written in a C-dialect-source file. As this script uses specific libraries, and a specific Python version, its portability is reduced. In order to cope with

the differences in setups at the end-user side, or on different testbeds, this script has been converted into a single binary. This significantly simplifies the execution of sensor experiments on WiSHFUL testbeds.

a. Dependencies

The TAISCParser script depends on the following libraries:

- Python2.7
- crcmod
- crc16
- python-clang-3.4

If the end-user has other versions of the above libraries installed, then the TAISCParser will fail to install. A virtual environment, with the correct packages installed, is needed to cope with this.

b. PyInstaller

The open source platform PyInstaller [10] can be used to compile the TAISParser into a single binary.

Standard usage of PyInstaller gives the following result:

```
root@wulf:~/staticpytest# python2.7 PyInstaller-3.2/pyinstaller.py TAISCParser.py
```

This command will result in lots of separate files: one file for every dependency. To combine all the shared libraries into one single binary, the following command can be used:

```
root@wulf:~/staticpytest# python2.7 PyInstaller-3.2/pyinstaller.py TAISCParser.py --onefile
```

The compiled file can then be distributed among end-users, while still being flexible for developers to modify the Python code if needed. Note that the compiled version is OS dependent.

5.3.2 Logic Analyzer for TAISC MAC debugging

The Saleae Logic Analyzer [11] can be used to do in-depth debugging of TAISC MAC protocols. 16 GPIOs (general purpose inputs and outputs) are selected on the target platform to observe the TAISC activities via a Logic Analyzer like: chain and instruction envelopes, radio activity, and so forth. For now the RM090 and the Zolertia Re-Mote sensor nodes are interfaced in a uniform way, but this approach can be applied to any other target platform.

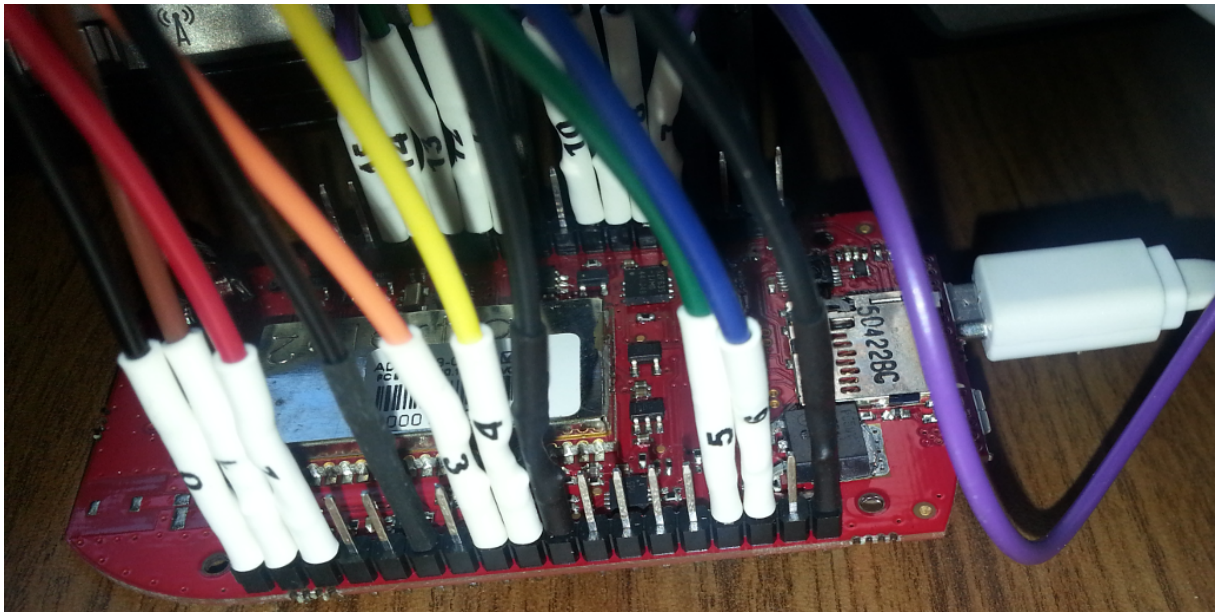

Figure 19: LA wires on the Zolertia Re-Mote

In combination with the Saleae software it enables the experimenter to monitor a MAC for a long period (typically in the order of hours) with lots of detail (sub-microsecond execution).

A more detailed list of the 16 LA channels can be found here:

- **0..2**: three wire (software) SPI (data, clock and enable) with 16 wide data which is used to reports on the active chain and instructions
- **3**: tasks invoked by TAISC out of interrupt context (similar to TinyOS tasks)
- **4**: BAD if this line goes high something (defined in software) went bad. It is used to trigger the LA software if something goes bad.
- **5**: TAISC train: shows the chain envelope. Becomes high on the first instruction of a chain and goes low again on a stop instruction.
- **6**: STATS: this channels informs about reports and assign which are invoked on the control plane interface and also the activities of the data plane interface like when a frame is passed for transmission, rxTrigger, txTrigger,...
- **7**: Instruction envelope. It shows the different execution parts of every instruction. First part is the fetching of the instruction, second part is the execution of the instruction itself, third part is used to report chain and instruction ID on the SPI (first 3 LA lines) of the current instruction and the last part is the rescheduling of the next instruction.
- **8:** becomes high when frames are being received or transmitted on the RF part of the radio. So could be seen as a frame envelope.
- **9**: this line displays if the radio oscillator is active.
- **10**: this line shows if the radio is transmitting (only cfr. Line 8).
- **11**: this line represents if the RF part of the radio is enabled.
- **12**: this line informs when events are triggered towards the TAISC core.
- **13**: displays the activity of the core scheduler
- **14-15**: these lines reveals the serial communication to and from the target platform

This has been an extremely important debugging tool especially for long time stability testing of TAISC in all kind of situations: MAC switching, different interference patterns, channel switching, and so forth.

As an example a brief workflow is described: in Figure 20 one can notice a flat line on the instruction channel 8. It happens after receiving an ACK on the last transmitted frame see channel 8. After zooming in, the ACK envelope is still visible on channel 8, around where the flat line begins one can

notice that instruction **33** (grabframe) freezes the train on TAISC chain 4 (0x04**33** was diplayed on the first 3 LA channels). So we know now that we can start debugging the grabframe instruction.

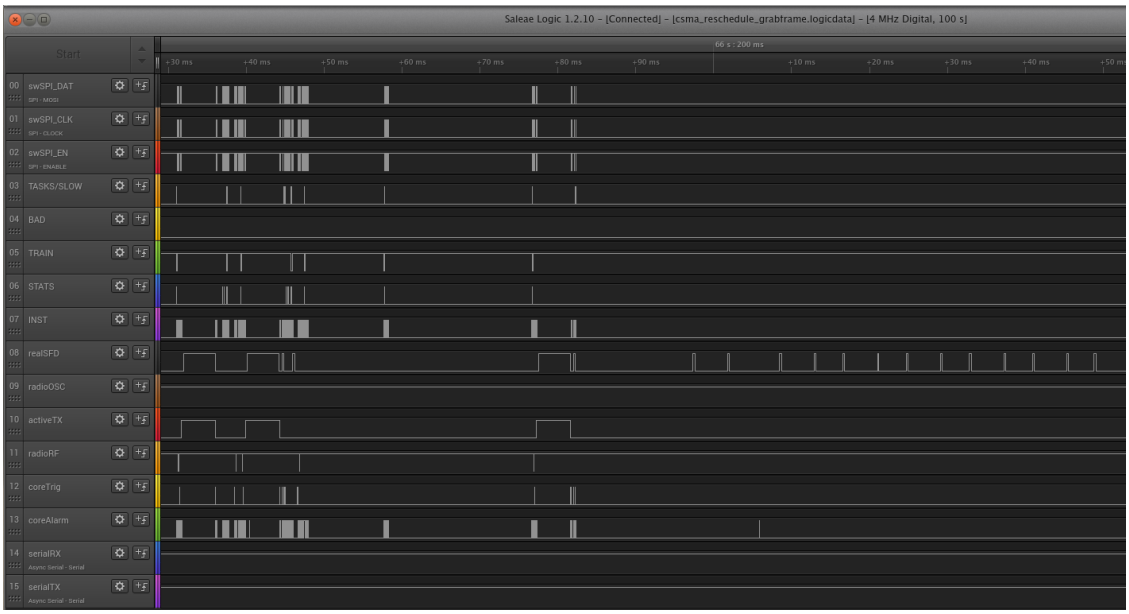

Figure 20: LA example 1

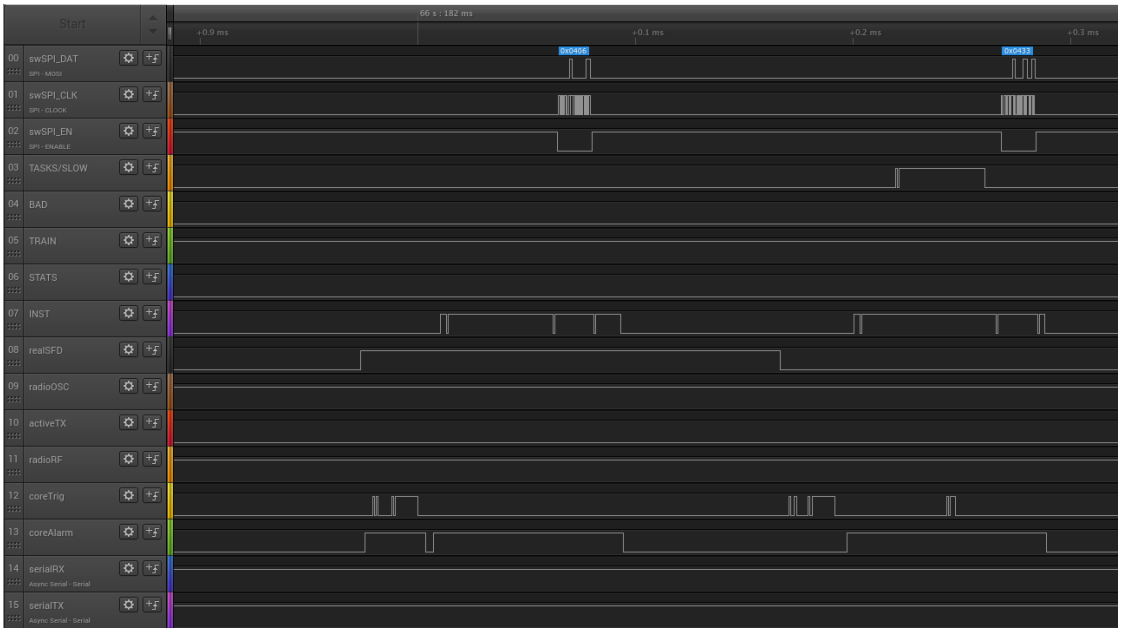

Figure 21: LA example 1 (zoomed in)

This hardware has been deployed on 6 sensor nodes in the w-iLab.t testbed and can be accessed by using the standard F4F testbed tools. This enables the users to do very detailed debugging of their sensor protocols.

5.4 Generic measurement visualization

To allow the experimenter to easily visualize the results obtained from experiments, a generic measurement visualization tool was developed. This tool operates on top of any database and is thus compliant with the OML measurement library. Currently, two database flavours are supported: MySQL and PostgreSQL. The tool does not require the experimenter to adopt a new programming language, since the input of the user is limited to SQL commands. Furthermore, since the tool is completely web based, it can easily be used on any testbed. It is currently in use in the w-iLab.t and the portable testbed. Figure 22 shows an example SQL query that can be inputted to the visualization tool.

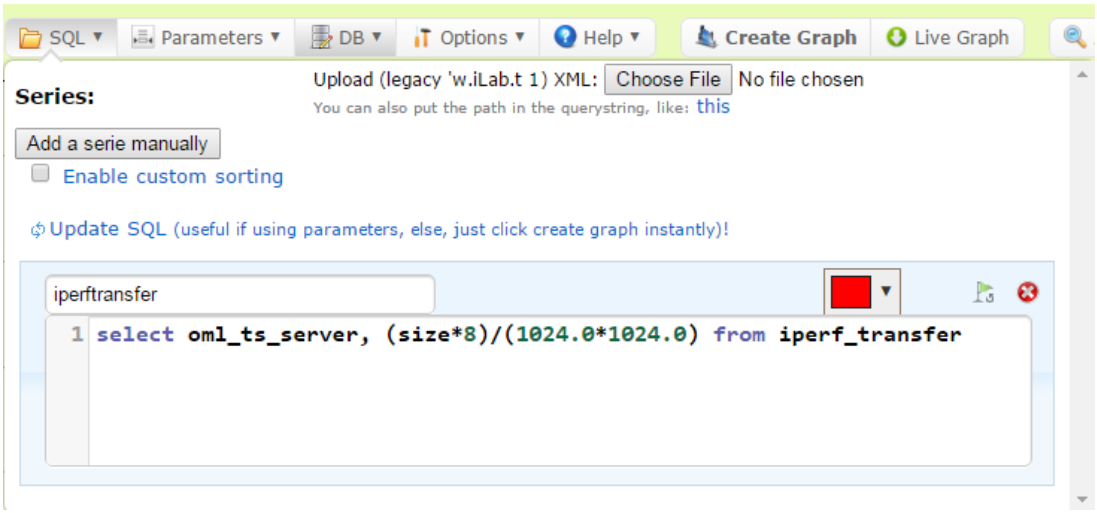

Figure 22: SQL command for measurement visualization

Figure 23 shows the throughput of an 802.11g WiFi network. Using this tool, it is possible to zoom in and slide through the graph. It is also possible to visualize multiple series (lines) on one graph, which can be dynamically enabled or disabled. The tool can visualize both live and offline measurement data. By right-clicking, the user can save the graphs in the desired format.

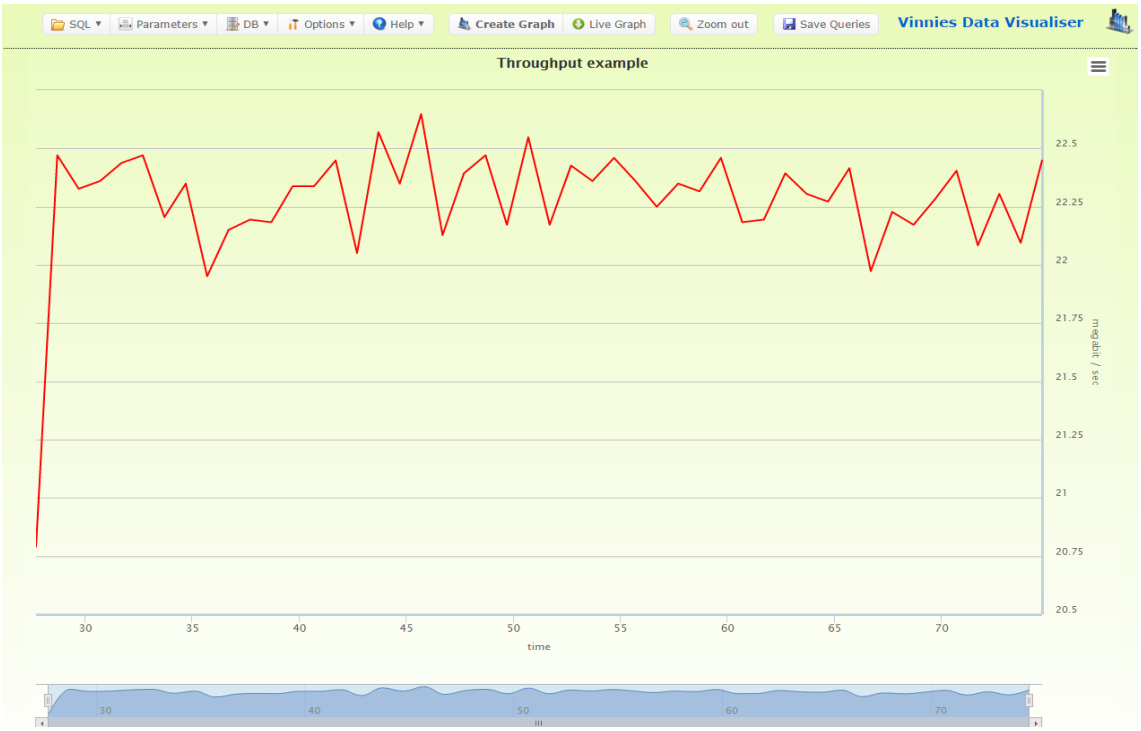

Figure 23: Example graph showing WiFi throughput

5.5 Spectrum sensing

Two types of hardware to do spectrum scanning are described in this chapter. The first type is the Wi-Spy spectrum scanner, which can be used to do very basic spectrum scanning. The second type is the more advanced mini USRP. Since both Wi-Spy and USRP mini can easily be connected over USB to any type of computer, the tools presented here can be applied to any testbed.

5.5.1 Wi-Spy

The Wi-Spy gives the experimenter a quick and basic view on what the current channel occupation is. This low-cost spectrum analyzer supports both 2.4 and 5GHz. Tools are provided for the experimenter that allow Wi-Spy sensing data to be stored in a database and easily visualize the measurements on a web page (called 'spectro'). More details on this tool can be found in D5.1 section 4.1.1. Figure 24 shows activity on 802.11g channel 1 (2412MHz).

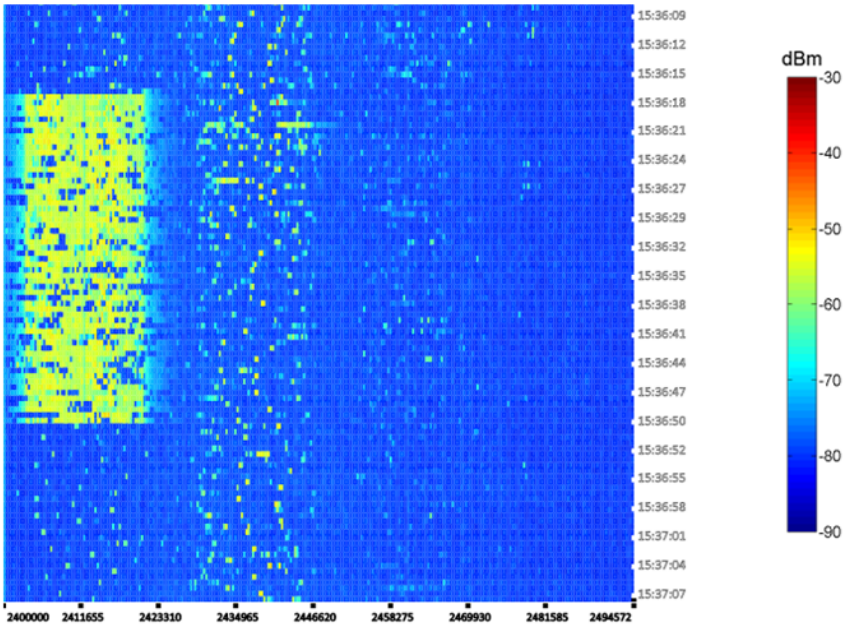

Figure 24: Wi-Spy output using the spectro tool

5.5.2 USRP mini

With its very small dimensions (8x5cm) and its USB3.0 interface, the USRP B200-mini (Figure 25) is a very portable spectrum scanner that can easily be plugged into any testbed.

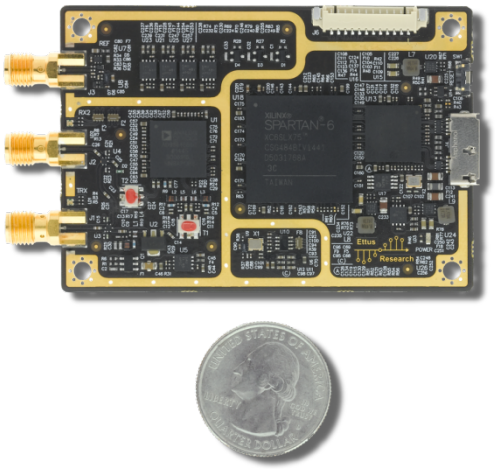

Figure 25: USRP B200-mini

In combination with the SDR framework GNUradio (see 5.2.1), the USRP mini allows the experimenter to easily visualize the spectrum from 70MHz to 6GHz, with up to 56MHz bandwidth. A simple screenshot from GNUradio can be seen in Figure 26. It shows simple energy detection on a given frequency, which makes it possible to e.g. detect WiFi or ZigBee packets.

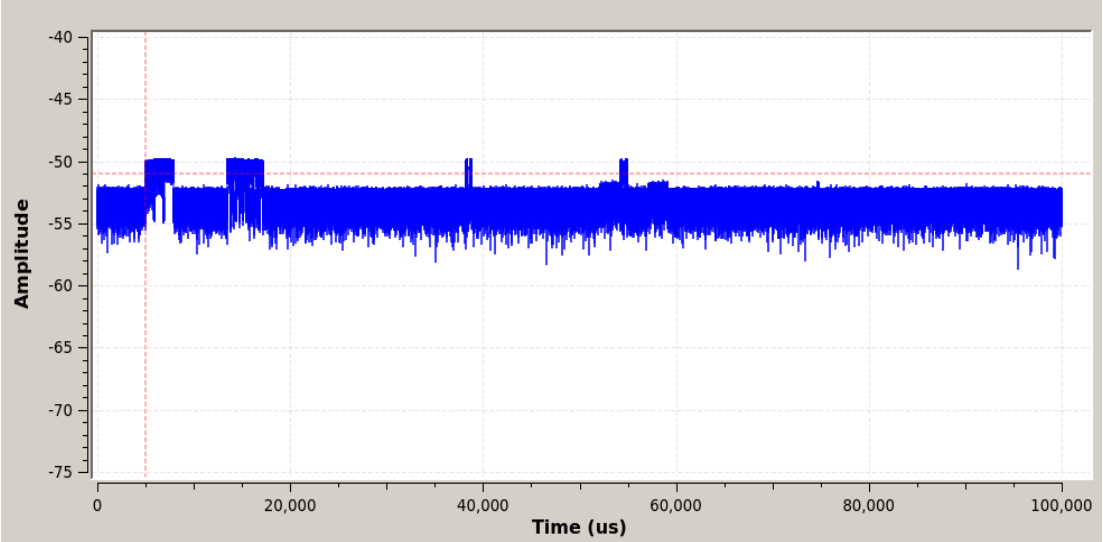

Figure 26: GNUradio screenshot constructed with USRP b200-mini

6 Conclusion

This deliverable shows that all of the testbeds in WiSHFUL (w-iLab.t, ORBIT, FIBRE Island @ UFRJ, IRIS, TWIST and the Portable testbed) have reached FED4FIRE compliance on the level of the AM API. This API enables the use of existing FED4FIRE tools on WiSHFUL testbeds. It also enables the use of a single user certificate to access all testbeds in WiSHFUL and FED4FIRE. The use of one user account in combination with uniform tools greatly simplifies the experimentation life cycle for experimenters. During the second year of the project, the IRIS testbed was brought up to F4F federation standards by implementing the GENI AMv3 SFA API by using the Geni Control Framework (GCF) provided by geni-tools.

For the IRIS testbed, an example experiment life cycle is illustrated, making use of existing FED4FIRE tools. The TWIST testbed can now support the creation of custom disk images and now provides testbed logging functionalities. For other testbeds, we refer to D5.2, since these testbeds were already F4F compatible at the end of Y1. The jFed, OMF6 and OML tools are supported by most testbeds and allows the user to very easily provision resources, do automated experiment control and collect measurements in a uniform way.

Additionally, this deliverable gives an overview of the extensions that were made to WiSHFUL testbeds. The w-iLab.t reports the addition of new wireless nodes to increase the capacity of the testbed as well as the support for 802.11ac, LTE and the Adant RAS antenna. In addition, a secondary testbed location of the w-iLab.t was equipped with 44 new wireless nodes supporting 802.11n, 802.11ac, 802.15.4 and Bluetooth 4.0. The IRIS testbed reports the addition of two digital TV tuners. The nodes in the TWIST testbed were extended with a second WiFi interface, as well as a WiSpy spectrum analyser for every node. The ORBIT testbed reports the addition of a number of LTE basestations and clients as well as an increase in the number of SDR patforms.

Furthermore, a detailed description regarding the integration of WiSHFUL UPIs in testbeds is given. The approach defines a loose coupling between the testbed interfaces and the WiSHFUL UPIs, by using Ansible playbooks and inventory files. Ansible playbooks are provided for the experimenter to facilitate the installation of WiSHFUL Global and Local MCEs, while the usage of inventory files allows the experimenter to create logical groups of nodes in his experiment. Using the built-in support for Ansible in jFed, the experimenter can create these logical groups when provisioning the experiment. This means that it becomes very easy to map the experiment to a different testbed, or to change the topology of the nodes without having to change the WiSHFUL experiment, because it only makes use of the logical groups instead of the physical resources. The inventory files can also be used by the OMF experiment controller to configure the experiment.

To conclude this deliverable, several tools are described to facilitate SDR research on testbeds, as well as to do advanced debugging of sensor experiments, visualize measurement data and do simple spectrum sensing in the WiSHFUL testbeds. To enable SDR research, both GNUradio and the IRIS tool can be used on several testbeds. The in-depth debugging of sensor experiments is made possible by the addition of a Logic Analyzer to 6 sensor nodes in the w-iLab.t testbed. To simplify the deployment of sensor experiments using TAISC, the TAISC parser is able to create a single sensor binary that can easily be deployed using WiSHFUL. Simple measurement visualization is made possible through a web page that fetches data from a database. Using SQL queries provided by the experimenter, this generic solution can be used for all types of applications and is not bound to a single testbed or a single tool like OML. To finish the document, two methods are proposed to do spectrum sensing in WiSHFUL testbeds. One method uses the low-cost WiSpy USB dongles to visualize the spectrum on a web page or to store the measurements in a database. The second method uses the USRP B200-mini in combination with GNUradio. These approaches for spectrum sensing will be further extended during Y3 of the project, during which spectrum sensing techniques from the CREW project will be applied to WiSHFUL testbeds while ensuring F4F compatibility.

7 References

[1] Geni-tools, https://github.com/GENI-NSF/geni-tools, last access December 1st 2016.

[2] ExperimentationTools repository on the WirelessTestbedsAcademy Github account, available at https://github.com/WirelessTestbedsAcademy/ExperimentationTools, last access December 16th 2016

[3] iLab.t authority, available at https://authority.ilabt.iminds.be/getcert.php, last accessed December 2nd 2016

[4] Iris Experiment Tutorial, http://iris-testbed.connectcentre.ie/experiment_tutorial/, last accessed December 7th 2016

[5] w-iLab.t documentation website, available at http://doc.ilabt.iminds.be/ilabt-documentation/wilabfacility.html, last accessed December 2nd 2016.

[6] TKN instructions on creating custom disk images, https://www.twist.tu-berlin.de/tutorials/custom-disk-images.html, last accessed December 18th 2016

[7] TKN Testbed system images, https://gitlab.tubit.tu-berlin.de/twist-testbed/twist-img, last accessed December 18th 2016

[8] Graylog, available at https://www.graylog.org/, last accessed December 18th 2016

[9] P. D. Sutton et al., "Iris: an architecture for cognitive radio networking testbeds," in IEEE Communications Magazine, vol. 48, no. 9, pp. 114-122, Sept. 2010.doi: 10.1109/MCOM.2010.5560595

[10] PyInstaller, available at http://www.pyinstaller.org/ or at https://github.com/pyinstaller/pyinstaller, last accessed December 12th 2016

[11] Saleae Logic Analyzer, available at https://www.saleae.com/ , last accessed December 14th 2016