WiSHFUL

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

European Commission

Project Deliverable D10.2

Results of first set of showcases using basic intelligence

Contractual date of delivery:	31-12-2016
Actual date of delivery:	23-12-206
Beneficiaries:	IMEC, TCD, CNIT, TUB, NCENTRIC, RUTGERS
Lead beneficiary:	CNIT
Authors:	Ilenia Tinnirello (CNIT), Domenico Garlisi (CNIT) , Pierluigi Gallo (CNIT), Merima Kulin (IMEC), Spilios Giannoulis (IMEC), Ingrid Moerman (IMEC), Francisco Paisana (TCD), Ivan Seskar (RUTGERS), Changmok Yang (SNU), Sunghyun Choi (SNU), Piotr Gawłowicz(TUB), Mikołaj Chwalisz (TUB)
Reviewers:	Anatolij Zubow (TUB), Maicon Kist (TCD)
Work package:	WP10 – General Requirements and Showcases
Estimated person months:	25
Nature:	R
Dissemination level:	PU
Version:	1.0

Abstract:

This publication deliverable reports on the results of the intelligent showcases that have been implemented and on the intelligence modules that have been developed in the showcases. It also includes a brief presentation of the WiSHFUL framework for building intelligence. This deliverable reports on activities in WP10.

Keywords:

Showcases, Proof-of-concept, intelligence framework

Executive Summary

This documents reports on the showcases implemented within the second year of the project to display the functionalities of the WiSHFUL intelligence framework. The framework has been designed for facilitating experimenters in the definition of intelligent control programs, by means of elementary data processing blocks, learning algorithms, and adaptation procedures, and by means of a graphical interface for linking components and building intelligence control programs. **Intelligent control programs** do not implement simple intuitive algorithms or static rules defined by experimenter for adapting radio and network settings, but rather **implement advanced reasoning algorithms that capture the wireless network behaviour and learn the best strategy** for taking decisions on radio and network configurations in a certain network context for optimally supporting wireless applications.

Four showcases, dealing with completely different network problems, have been designed by following two different approaches for building the intelligence control program: **off-line learning**, which means collecting data from different experiments and processing these data off-line in order to design a data-driven model to be used for optimizing a desired network metric; **on-line learning**, where data becomes available during the experiment execution and is used to update the model, e.g. the predictor model, for future steps. As far as concerns off-line learning, two different data-driven models have been developed for **sensor networks** running different MAC protocols and for **WiFi networks with high-density of access points**. The models are then used for facilitating the identification of optimal MAC protocols under dynamic network conditions. MAC protocol optimizations have been also considered for on-line learning. In particular, a learning engine has been developed for **ranking different elementary MAC protocols** as a function of the network traffic and measurements of channel state (from slotted ALOHA in case of low traffic, to TDMA in case of greedy traffic), and for act consequently by enforcing the protocol with the highest rank. A similar approach has been proposed for cognitive networks, in which the secondary users select the operating channel **by learning the primary user behaviours** and by ranking accordingly the channels as a function of the expected time interval in which the channel will not be used by primary users.

A brief overview of the accomplishments within each showcase is provided alongside a description of the utility offered by the WiSHFUL intelligence. As such, this document provides an initial catalogue of the WiSHFUL intelligence general capabilities and software modules developed so far in the WiSHFUL intelligence repository.

List of Acronyms and Abbreviations

AM	Aggregate Manager
AODV	Ad hoc On-demand Distance Vector
AP	Access Point
API	Application Programming Interface
BAN	Body Area Network
BSS	Basic Service Set
CDF	Cumulative distribution function
CPU	Central Processing Unit
CSMA	Carrier Sense Multiple Access
CTS	Clear to Send
DMT	Discrete MutiTone
DSL	Digital Subscriber Loop
DSR	Dynamic Source Routing
DT	Delay Tolerant
EM	ElectroMagnetic
EMS	Experiment Management Server
EWMA	See Figure 8
F4F	Federation for FIRE (Future Internet Research Experimentation)
FBMC	Filter Bank Multi-Carrier
Fed4FIRE	Federation for FIRE (Future Internet Research Experimentation)
FNR	False Negative Rate
FRCP	Federated Resource Control Protocol
FSM	Finite State Machine
GPIO	General Purpose Input Output
HAL	Radio Abstraction Layer
HTTPS	HyperText Transfer Protocol Secure
I/Q	In phase / Quadrature
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
KPI	Key Performance Indicator
LTE	Long Term Evolution
LTE-A	Long Term Evolution – Advanced
LQI	Link Quality Indication

OBSSs	Overlapping BSSs
PRR	Packet Reception Rate
MAC	Medium Access Control
MTC	Machine-Type Communications
NOC	Network Operations Centre
OFDM	Orthogonal Frequency Division Multiplexing
OLSR	Optimised Link State Routing
OMF	OMF Measurement Library
OML	Orbit Management Framework
OODA	Observe, Orient, Decide and Act
PLC	Power Line Communication
QoS	Quality of Service
RAM	Random Access Memory
RF	Radio Frequency
ROC	Receiver Operating Characteristic
RP	Radio Processor
RSpec	Request Specification
RSSI	Received Signal Strength Indication
RT	Real-Time
RTS	Request to send
SDR	Software Defined Radio
SFA	Slice Federation Architecture
SNR	Signal to Noise Ratio
SSH	Secure Shell
STA	Station
TCP	Transmission Control Protocol
TDM	Time Division Multiplexing
TDMA	Time Division Multiple Access
TPR	True Positive Rate
TSMP	Time Synchronized Mesh Protocol
TSCH	IEEE 802.15.4e
TSF	Timing Synchronization Function
UC	Use Case
UDP	User Datagram Protocol
UMTS	Universal Mobile Telecommunications System (UMTS)
UPI	Unified Programming Interface

UPIR	Unified Programming Interface radio
UPIN	Unified Programming Interface network
UPIHC	Unified Programming Interface hierarchical control
URL	Uniform Resource Locator
USB	Universal Serial Bus
USRP	Universal Software Radio Peripheral
VPN	Virtual Private Network
WiFi	Wireless Fidelity
XFSM	eXtended Finite State Machine

Table of contents

1 Introduction

In this document we describe four experiments that have been designed for showcasing the WiSHFUL intelligence, as defined in D10.1. Indeed, the original WiSHFUL software architecture for radio and network control has been extended for supporting an intelligence framework devised to facilitate the building of solutions able to optimize network performance in different contexts. For facilitating this duty, the WiSHFUL intelligence framework provides a graphical interface for linking specialized software components and algorithms, as well as components for: i) collecting, filtering and aggregating data during experiments, in order to identify some relevant features that can be related to performance metrics of interest, ii) supporting learning, inference or other reasoning algorithms, in order to map data observations into actions as a function of the application requirements; iii) performing coordinated actions on single or multiple network nodes, which aggregate elementary UPI_R and UPI_N functions into more complex adaptation procedures (e.g. by synchronizing two overlapping Access Points with a common temporal reference such as the timestamp provided by one of the two nodes.).

During Y2, we envisioned two main methodological approaches for designing network intelligence: i) **building off-line a network model or classifier** devised to estimate a specific performance figure, by working on data collected under different operating scenarios, or on protocol models and simulation results (off-line learning), ii) **designing an on-line learning engine**, able to update in run-time the prediction model as a function of the observed data observed and act consequently (on-line learning).

The document is organized as follows. In Section 2, we present the general overview of the WiSHFUL intelligence architecture, the basic data collection, intelligence engine and action components developed within Y2 activities, and the graphical framework, e.g. the Node RED that can be used for connecting the basic components, managing the data flows and building the learning models. In Section 3 and Section 4 we present, respectively, the showcases based on off-line learning or on-line learning. For each showcase, we describe the network problem addressed by the WiSHFUL intelligence and provide an overview of the WiSHFUL functionalities used for addressing this problem. Moreover, the document provides an indication of potential extensions or continuations of the work completed so far. As presented, the material provided in this document is organized for demonstrating the potentialities of the WiSHFUL framework to the broader community of experimenters for building intelligent adaptations in real network contexts.

2 General overview of the WiSHFUL Intelligence

2.1 Conceptual intelligence framework

The connection between the WiSHFUL software architecture for radio and network control and the intelligence framework is made by the Unified Programming Interfaces. The generic functional view can hence be mapped to the conceptual framework for enabling intelligence shown in Figure 1.

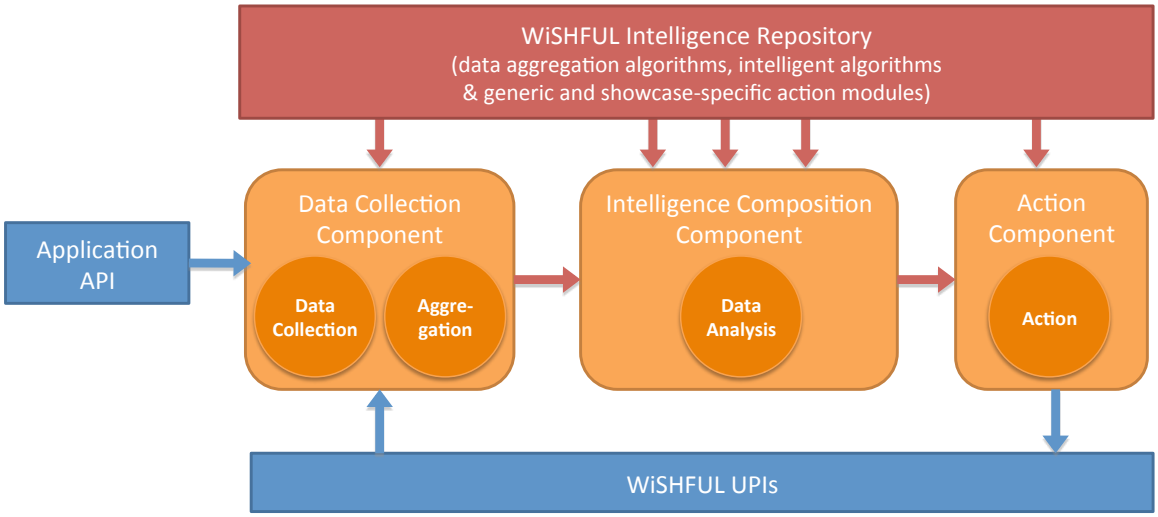


Figure 1: Conceptual framework for enabling intelligence

As the UPIs are unified abstractions that span several wireless technology platforms, the components of the intelligence framework are generic. The Data Collection Component is a generic software module that interacts with the WiSHFUL UPIs, UPI_R, UPI_N and UPI_G to retrieve data about radio and network state (i.e. channel occupancy, LQI, RSSI, PRR, topology, etc.), and with the Application to retrieve information about the application requirements (e.g. max delay, peak throughput, max PER). The Data Collection Component also implements aggregation functionality. The Intelligence Composition Module offers support for composing and configuring several algorithms available in the WiSHFUL Intelligence Repository into a self-contained intelligence engine that uses the data provided by the Data Collection Component and triggers network and radio configuration through the Action Component. The Action Component uses the WiSHFUL UPIs to adjust the configuration of radio and network. The radio and network configuration should be viewed as the output of the intelligence process. Such a configuration can deal with individual parameters (e.g. center frequency, backoff delay, etc.), radio processing elements (e.g. filter swapping), a waveform (e.g. a modulation and coding scheme) or a protocol (e.g. new MAC scheme).

The framework allows to support the usual Observe, Orient, Decide and Act loop (**OODA loop):** the data collection component is responsible of gathering data observations and aggregating and filtering the data for extracting the features used in the orient phase; the intelligence composition component is responsible of taking decisions on the basis of the previous observation and orient phase; the action component is responsible of implementing an adaptation decision by reconfiguring the wireless nodes.

The WiSHFUL intelligence framework offers a common set of tools that enable the realization of intelligent approaches using the algorithms from the repository. Together with the UPIs the WiSHFUL software architecture of the intelligence framework enables reasoning about the current network state and applying actions to change the configuration of radio and network.

2.2 Intelligence approaches and components developed in Y2

We envisioned two main methodological approaches for designing network intelligence: i) ***building off-line a network model or classifier*** devised to estimate a specific performance figure, by working on data collected under different operating scenarios, or protocol models and simulation results, ii) ***designing an on-line learning engine***, able to update dynamically a prediction model as a function of run-time observations and act consequently. The two approaches and their relationship with the generic WiSHFUL UPI are summarized in the following Figure 2.

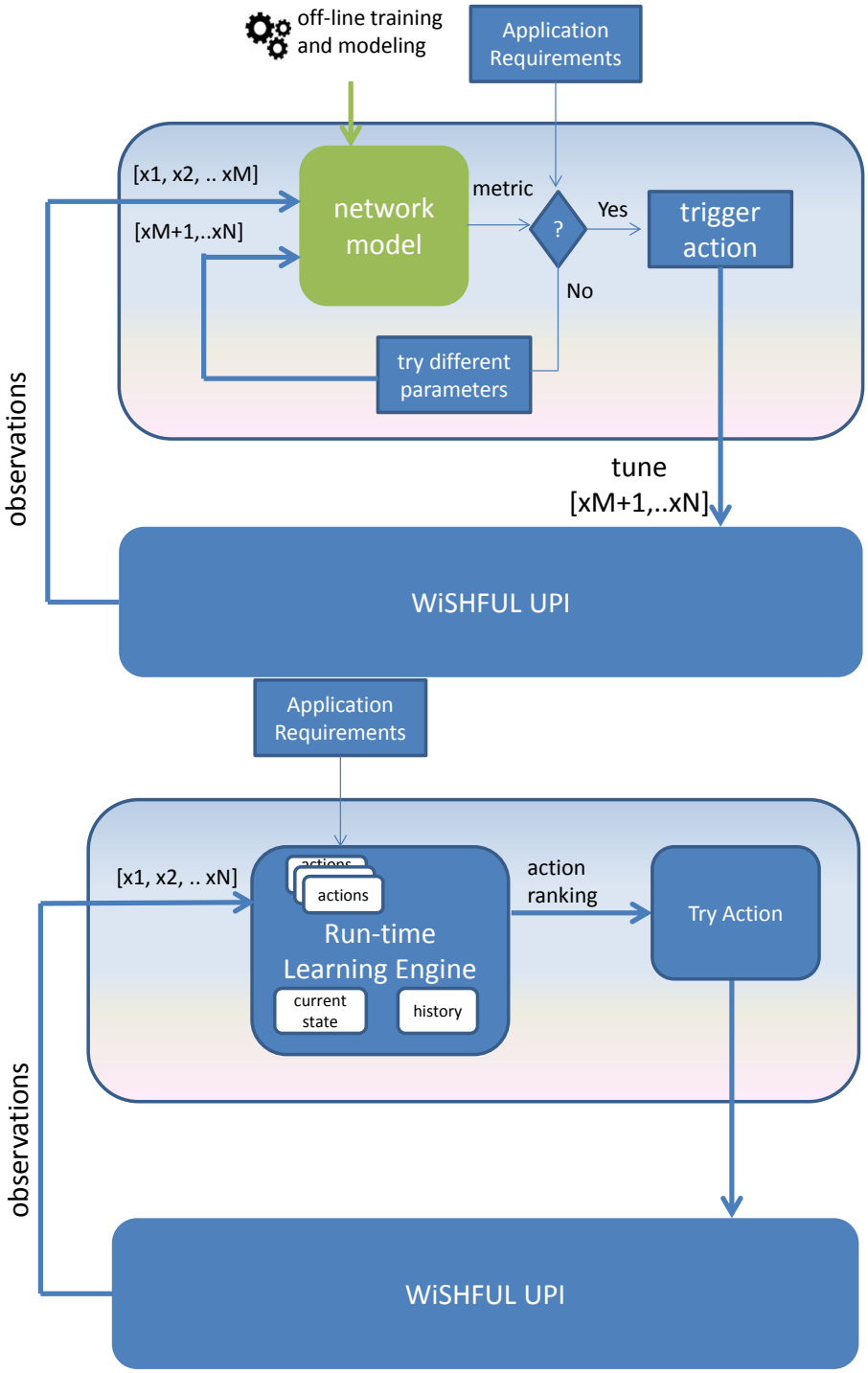


Figure 2 – High-level representation of off-line (top) and on-line (bottom) learning approaches for defining network intelligence.

According to the first approach, preliminary experiments are planned for validating analytical/simulation models, or for training a data-driven model by means of machine-learning techniques. Experiment planning is devised to cover the most representative operating conditions envisioned in the network problem under study, and to capture cross-correlations among the most relevant model inputs (i.e. the network operation conditions and configurable parameters) and the model output (i.e. the performance metric to be estimated). When ***the model*** is ready, it ***can be used for estimating the desired performance metric as a function of run-time observations of network state and configurable protocol parameters, in order to find dynamically the optimal configurations for these parameters***.

According to the second approach, ***the network intelligence works by updating a prediction model in run-time for estimating the effect of a given action on the basis of historical network observations. Multiple prediction models can be used in parallel for comparing the expected benefits of different actions, among a set of available ones***. The most important design aspects involve the identification of the learning scheme for updating the model and selecting the best actions (which may also correspond to the configuration of a given parameter). Network optimization is then achieved, step-by-step, as a sequence of action decisions.

Obviously, more complex structures, such as the parallel utilization of multiple network models (trained off-line), and even mixed approaches can be also considered. During Y2, we worked on both off-line and on-line approaches with four different showcases, which led to the initial population of the WiSHFUL intelligence repository. Indeed, although the models and the decision engines have been used for solving specific network problems, they can be potentially re-used in completely different contexts. In particular, we developed:

- a **WSN performance model**: this data-driven model is devised to estimate performance of WSNs under different MAC protocols, as a function of node density and interference conditions;
- a **WiFi performance model for multi-cell scenarios**: this data-driven model is devised to classify interference conditions experienced in different cells as hard, soft or zero interference and identify blocked cells;
- a **Meta-MAC engine**, for dynamically learning about the best medium access decision among a set of elementary protocol components, in WiFi fully-connected networks under varying traffic conditions;
- a **channel-hopping decision engine**, for inferring about the channel that will remain available for longer time intervals in cognitive networks, in case of predictable behaviours of primary users.

2.3 Implementation of the framework for building intelligence

As described in section 2.1 and 2.2, a set of algorithms and modules have to interact with each other through the WiSHFUL intelligence framework. A specified interface is required, so each module can cooperate with another, i.e. the input of one module should be able to understand the output of an algorithm. The linking of modules and algorithms should in turn also be checked for minimal consistency and compatibility.

Comparison of different suitable frameworks

There are already several drag and drop graphical user interfaces available that are open source and might suit the needs of the intelligence framework.

A comparison is given in the Table 1. Comparing several key points such as:

- Drag and drop features are one of the key points that the system should have. The ability to take a (few) component, drop it (them) on a sandbox dashboard and interconnect them as the user sees fit.
- Configurable components: the depth of configurability of components introduced into the framework.
- Component interaction describes the way in which the components communicate with one another. Here was investigated if there is a standardized protocol for this communication to take place, which is important if third parties develop new components.
- Web UI: is the application only client-side application or is the UI available via a browser? The latter is preferred, as it offers more portability to different platforms.
- Socket interaction: describes the possibility for inter-process communication (by means of TCP/IP sockets).
- Python support: does the framework support (user)code written in Python?
- Next, the programming language for developing new components to extend the framework was investigated.
- Subsequently it would be very convenient if the framework could call external scripts, located on local machine of the user. This enables interaction with complex algorithms, reasoning and statistics platforms (e.g. MATLAB).
- It is also important, for future development, that a wide user base and component-repository is available.
- Lastly, the system should be open source.

Functionality / Framework	**Node-RED**	**LimeDS**	**jsPlumb**	**ThreeNodes**
Application domain	IoT	Integration with internal or external data services	Library to link HTML structures	Visualizing geometric shapes / basic programming
Drag & drop				
Configurable components				Not easy
Component interaction (standardized?)			User Defined	Not modifiable
Web UI		Java engine		
Sockets interaction		No, only http webservices		
Python support	Indirect			
Programming language	JavaScript	JavaScript	JavaScript	non configurable
Call external scripts	with params, capturing outputs	?	Not Easily (implement yourself)	
Repository	http://flows.nodered.org/			
Open Source		Not open source		

Table 1: Comparing intelligence framework candidates

Based on Table 1, a choice was made. The choice was Node-RED, a tool designed by IBM to wire different components in the context of Internet of Things (IoT).

Drag & drop
The Node-RED UI is intuitive and features a toolbox with possible components on the left. By dragging them to the board and connecting those with others, complex interactions and ultimately a flow can be created.

Configurable components
The pre-existing nodes are open-source and can be modified.

Interaction among internal Node-RED components
Node-RED uses JSON as an intermittent protocol to communicate with different nodes. Messages are serialized in JSON and exchanged between components/nodes.

It would thus be possible to create a database block, configure it, and feed information to an intelligence block, resulting in changing a parameter via an action block.

Interaction with external components
As compared in Table 1, Node-RED can interact with external components by executing scripts. This can be implemented in two ways:

1. Running external scripts / binaries using an exec-node
2. Running external scripts / binaries using a daemon-node (plugin)

With the key difference of Unix STDIN / STDOUT pipes remain open in option 2.

Option 1 is also blocking, it will wait for the binary to execute completely and return to Node-RED, or using a timeout which will kill the external component.

Passing arguments to external scripts can be tricky and is limited. For simple text-based arguments this is pretty straightforward, but when complex JSON-messages have to be passed, it would be best to open a socket or pipe.

In Wishful context it would be very convenient if UPI functions could be called directly from a Node-RED component, interacting with a Global Controller that is compatible with Node-RED information stream.

Creating components
Several nodes already pre-exist in Node-RED (either native or via the extensive repository at http://flows.nodered.org/):

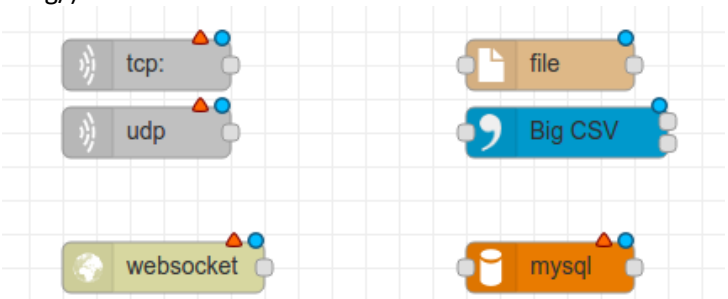


Figure 3: Pre-existing Node-RED nodes

One can use several methods to open sockets, connect to databases, and access the filesystem for files. These nodes are all open source and some can be downloaded through the Node-RED repository.

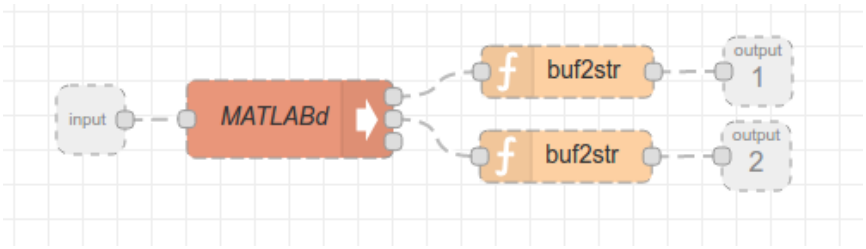
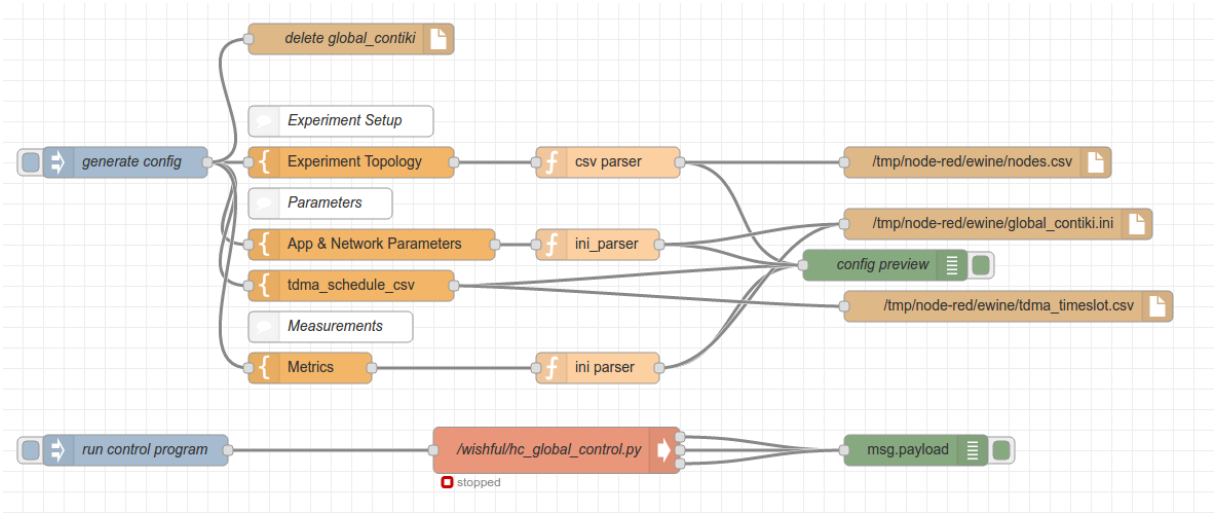


Figure 4: Sub-Flows

As shown in Figure 4, a user can group several nodes into a Sub-Flow. Sub-Flows can also contain other Sub-Flows. This makes it easy to preserve an overview of a complex system.

Using only basic components it is already possible to start and stop WiSHFUL Global Control Program:



The blue nodes trigger the function nodes that generate the configuration files for the Global Control program. If the program is running, all information is displayed in the Node-RED log-window.

In order to modify certain parameters, the Global Control Program script should be adapted slightly to accept communication from Node-RED. A handler was added that opens a socket and accepts JSON-data, which then translates to a UPI-call. The Node-RED component that supports this communication is a newly implemented UPI_Exec-node (of which the user interface is displayed in Figure 5).

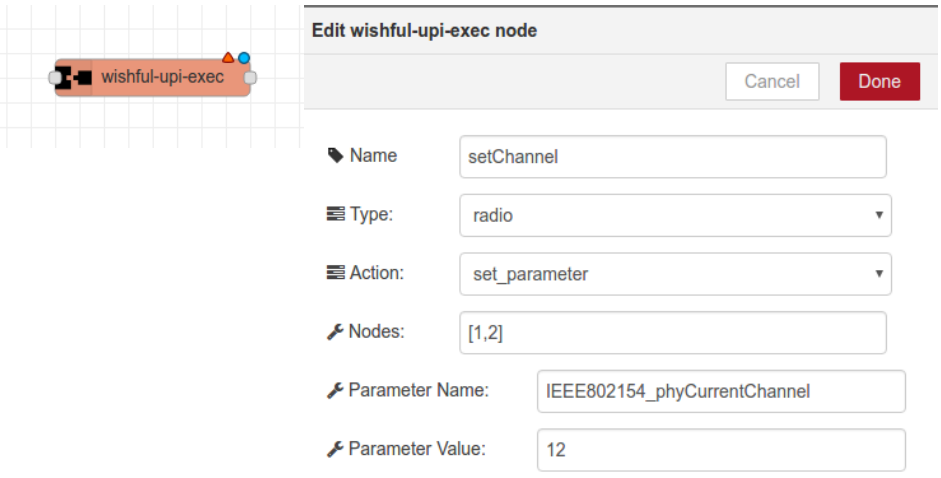


Figure 5: UPI-Exec node

This node can execute a set and get_parameter UPI-function. The result of the node is JSON, which will be accepted and interpreted by the Global Control program, which in turn executes the appropriate UPI call and returns its response to the Node-RED context and specifically to the UPI_Exec-node component (success or fail) that initiated the procedure.

request

```
{
    "execute_upi_function":{
        "upi_type":"radio",
        "upi_func":"set_parameters",
        "node_list":[
            1,
            2
        ],
        "args":{

"IEEE802154_phyCurrentChannel":12
        }
    }
```

response

```
{
    "1":{
        "IEEE802154_phyCurrentChannel":12
    },
    "2":{
        "IEEE802154_phyCurrentChannel":12
    }
}
```

A small demo flow demonstrates the use of the node (Figure 6). It sets the channel of the physical layer of a sensor node to a specific value.

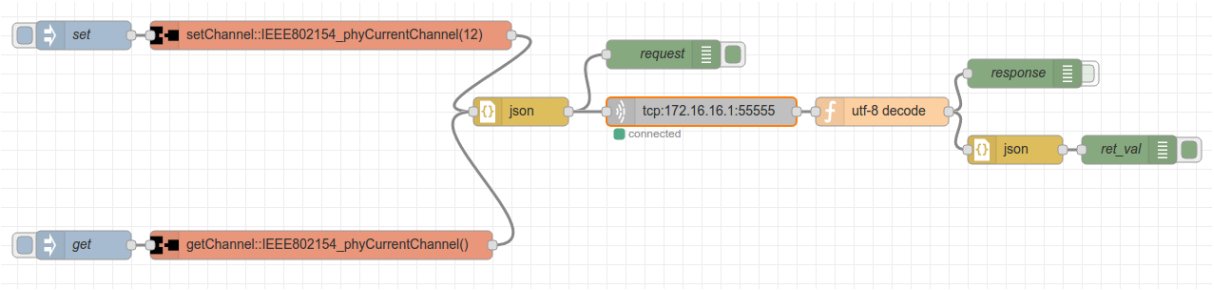


Figure 6: Modifying the PHY-channel using UPI_Exec nodes

In order to fully integrate the intelligence algorithms, it was necessary to create custom nodes. Several dummy nodes were implemented, a toolbox-screenshot is provided in Figure 6. Future implementation work will allow them to properly communicating with each other.

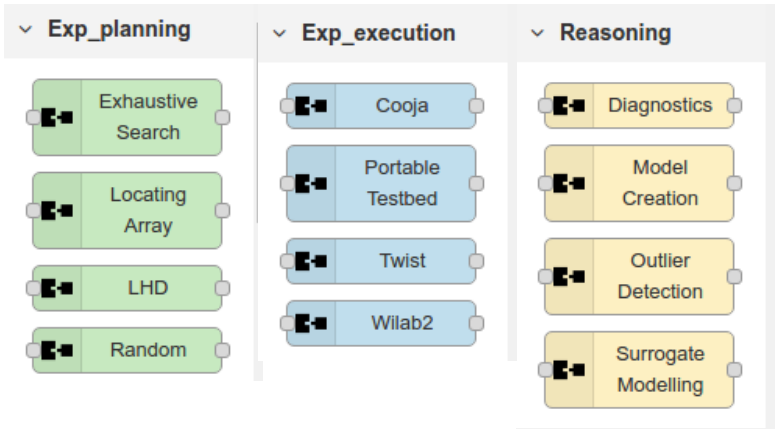


Figure 7: Dummy modules

The Node-RED framework will need to be extended further with custom nodes to make interaction with (1) data, (2) intelligence and (3) action components possible, as already described in 2.1.

3 Intelligence showcases based on off-line modelling

3.1 Cognitive MAC protocol selection

Motivation and problem statement

Wireless sensor networks (WSN) have experienced explosive growth due to the promising and innovative application scenarios arising in the context of the Internet of Things (IoT). The availability of inexpensive low-power sensor devices has led to an unprecedented surge in the number of connected devices. These devices generate traffic from heterogeneous radios that follow different medium access protocols and communication standards. A few examples in the 2.4GHz unlicensed band include IEEE 802.15.4, Bluetooth, WiFi, RFIDs, while in the sub-1GHz LoRA, SigFox, IEEE 802.11ah, etc., each suitable for a specific application domain. Such diversity of wireless devices/technologies, applications and services will pose several communication challenges, particularly, co-existence, cross-technology interference, spectrum scarcity and uncertainty in communication quality.

To tackle this problem, more sophisticated medium access (MAC) protocols are needed. Traditional wireless MAC protocols are typically designed to optimally perform for low-power operation (e.g. ZigBee, Bluetooth low energy), low latency (e.g. wireless HART), high throughput (e.g. WiFi) or high reliability (e.g. TSCH). We argue that rather than developing optimized solutions for specific IoT application domains, a more suitable approach may be a cognitive MAC layer that is able to dynamically choose the most appropriate MAC protocol and configuration parameters in order to adapt to observed network conditions and maintain the communication quality with regard to the application requirements.

The idea of cognitive communication originates from the cognitive radio (CR) community, however, to this end their efforts have been focused on prototypes for powerful radios such as SDR platforms [1]. In this work, we present a system design and first step implementation towards a cognitive MAC layer for constrained devices. We argue that the key component for an cognitive MAC layer is predictable performance. From a networking perspective, the most relevant aspect of reliable wireless communication quality is the packet delivery performance [2]. In particular, for energy-constrained devices such as in battery-powered wireless sensor networks (WSN), one of the main goals is to reduce the number of radio transmissions and to simultaneously increase packet delivery. Packet reception rate and throughput are important as they directly relate to energy consumption. Efficient performance estimation in wireless networks is crucial as it can reduce the energy consumption by alleviating situations with failing transmissions.

Therefore, we use machine learning techniques to model the packet delivery as a function of the collected measurements (number of neighbours, packet inter-arrival times, number of received packets, number of observed collisions) [3]. Although an essential part for our cognitive MAC layer, the created machine learning model can act as a standalone reusable performance prediction component that can be integrated into other systems where performance prediction is necessary.

3.1.1 General overview

Accurate MAC layer performance estimation in low-power wireless networks is a challenging task due to the notoriously dynamic and unpredictable wireless environment. In order to study the packet error rate experienced with a given MAC protocol we used machine learning as an effective technique for real-time characterization of the communication performance as observed by the MAC layer [3]. Our approach is data-driven and consists of three steps: extensive experiments for data collection, offline modelling and trace-driven performance evaluation [4]. The following sections explain the process of deriving the machine learning model.

In order to study the MAC layer performance, extensive experiments have been performed on the wilab2 testbed facility in Ghent.

We used up to 30 RM090 nodes with an IEEE 802.15.4 radio organized in a star-like network topology as shown on Figure 8. All nodes run a CSMA/CA MAC protocol that is developed in the TAISC framework [5]. They have been configured to periodically generate a *100B* message to a single receiver located in the centre of the topology. The transmission power is set to the maximum, i.e. *5dBm*, to ensure that all nodes are in communication range. To incorporate all factors that impact the MAC performance we setup several experiments varying the number of sending nodes (2-30 nodes), and the application traffic load (1 *pckt/2s*, 1 *pckt/s*, 2 *pckts/s*, 4 *pckts/s*, 8 *pckts/s*, 16 *pckts/s* and 64 *pckts/s*). In order to study the MAC-level performance under high interference, we used an USRP B210 to generate controllable interference patterns by transmitting a modulated carrier for 2 ms, followed by an 8 ms idle period (repeating this over time).

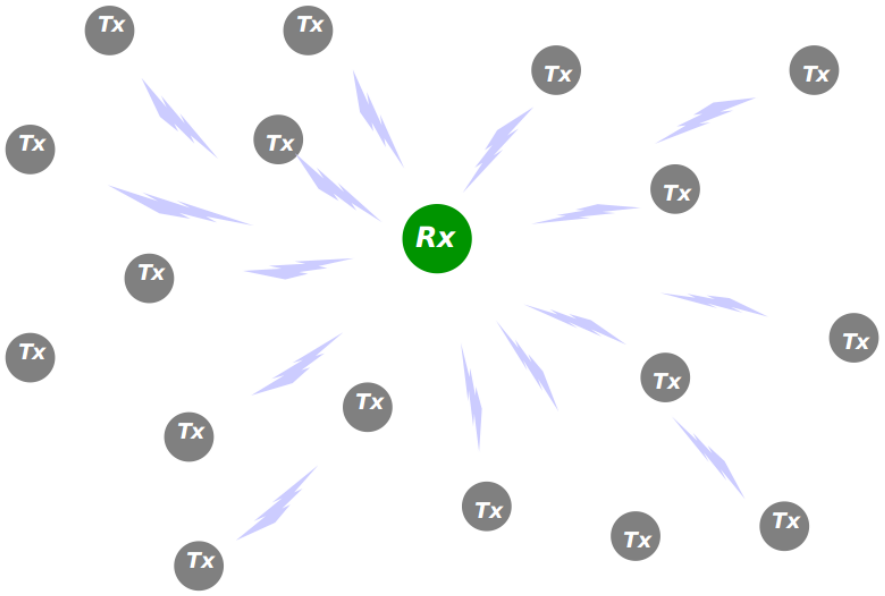


Figure 8 Experiment setup

The setup consists of up to 28 wireless nodes. The basic experiment consists of a variable number of sender nodes and one receiving node, i.e. sink node. To facilitate the experimental control and data collection from the nodes a WiSHFUL UPI global control program has been implemented. The global control program sets up the MAC and application level parameters (e.g. MAC protocol, packet size…) on all nodes and controls the duration for monitoring the measurements. During the experiment the density and network load has been controlled by configuring opportunistically the wireless nodes involved into the experiment, while interference has been introduced by the synthetic interferer as explained in previous section. The sink node collects statistics for each received packet and forwards them to the global controller for storage (using the event-based monitoring WiSHFUL functionality).

Data collection

To simplify experiment control and data collection we created a global control program on the global controller using the Unified Programming Interfaces (UPIs) developed by the WiSHFUL project [6]. We used the UPIs to *collect* measurements from wireless nodes, *configure* radio and network parameters and on-the-fly *adapt* node configuration on nodes that run a WiSHFUL agent program.

The global control program sets up the MAC and application level parameters (e.g. MAC protocol, packet size…) on all nodes and controls the monitoring duration. During the experiment the density and application load has been controlled, while interference has been introduced by the synthetic interferer.

We run several experiments with the aforementioned setups and measured several aspects of the MAC-level performance, while the system was operating (~18h). We captured raw data with per-packet statistics using events available for the IEEE 802.15.4 connector module. The raw data consists of: Sender ID, 802.15.4 Sequencing Number, Topology (Node density), Interference Indication, Traffic Load, packet loss.

Feature space design

We extracted the most relevant features for predicting the MAC performance from consecutive observation intervals of the raw data. Those are: number of detected nodes (*d*), inter-packet-interval (*IPI*), number of received packets (*rP*) and the number of erroneous packets/frames (*errP*). Then we formed the following feature vectors:

$$x^{(i)} = [d,\ IPI,\ rP, errP]^T,$$

and the corresponding communication reliability in terms of packet loss rate,

$$y^{(i)} = plr.$$

The density, d, is a good indicator about the number of contending nodes and potential intra-technology interference. The IPI is a good feature for reasoning about the current application demand and traffic load. Finally, last two features, rP and $errP$, are representative for inferring about the interference level and congestion in the network.

Offline machine learning

Pairs of ($x^{(i)}$, $y^{(i)}$) were used to train three machine learning algorithms: regression trees, linear regression and neural networks.

Linear regression

Linear regression is a technique for modelling the relationship between the input (x) and output variable (y) so that the output is a linear combination of the input variables (dependent variable).

$$y(x) = \theta_0 + \theta_1 x_1 + \cdots + \theta_n x_n = \theta_0 + \sum_{i=1}^{n} \theta_i x_i$$

We used the linear regression implementation available in Weka [7].

Regression trees

A regression tree is a tree-based learning algorithm that is used to predict a variable that takes continuous or ordered values. Regression trees is a supervised learning algorithm that creates a tree-like graph or model that represents the possible outcomes or consequences of using certain input values. The tree consists of one root node, internal nodes called decision nodes, which test its input against a learned expression, and leaf nodes which correspond to a final class or decision. The learning tree can be used to derive simple decision rules that can be used for decision problems by starting at the root node and moving through the tree until a leaf node is reached where a prediction outcome is assigned.

We used the M5P regression tree algorithm implementation available in Weka.

Neural networks

Neural Networks (NN) or artificial neural networks (ANN) is a supervised learning algorithm inspired on the working of the brain, that is typically used to derive complex, non-linear decision boundaries for building a classification model, but are also suitable for training regression models when the goal is to predict real-valued outputs. Neural networks are known for their ability to identify complex trends and detect complex non-linear relationships among the input variables at the cost of higher computational burden. A neural network model consists of one input, a number of hidden layers and one output layer. The input layer corresponds to the input data variables. Each hidden layer consists of a number of processing elements called neurons that process its inputs (the data from the previous layer) using an activation or transfer function that translates the input signals to an output signal.

We used the multilayer perceptron implementation available in Weka to train the neural network.

The algorithms were trained over several observation intervals, and validated with a 10-fold cross-validation algorithm [4]. We used the root mean squared error (RMSE):

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\bar{y}^i - y^i)^2}$$

where, y^i are the real outcome, while $\bar{y}^i$ the estimations, and the correlation coefficient:

$$\rho_{y\bar{y}} = \frac{cov(y,\bar{y})}{\sigma_y \sigma_{\bar{y}}}$$

as performance metrics to assess the performance of the algorithms. Figure 9 illustrates the high-level processes for building the machine learning models.

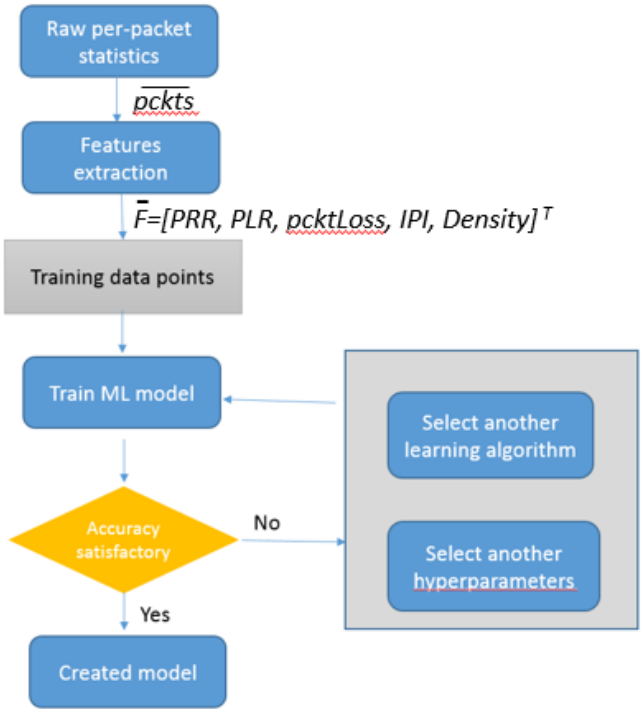


Figure 9 Offline machine learning training process

3.1.2 WiSHFUL functionalities and showcase phases

Figure 10 presents the conceptual system architecture to accomplish a cognitive loop for cognitive MAC selection [3]. There are two main components, the:

- **Sensor network** - a set of wireless nodes that generate information and are capable of reconfiguring its transmission parameters at runtime.
- **Global controller** - the central entity that collects and uses information from the wireless nodes to predict the MAC-level performance. Based on the predictions, it dynamically decides how to configure the MAC layer so as to improve the overall network performance (e.g. to cope with cross-technology interference it may decide to configure a more interference robust MAC protocol, e.g. TSCH). Finally, it disseminates the new configuration to the nodes.

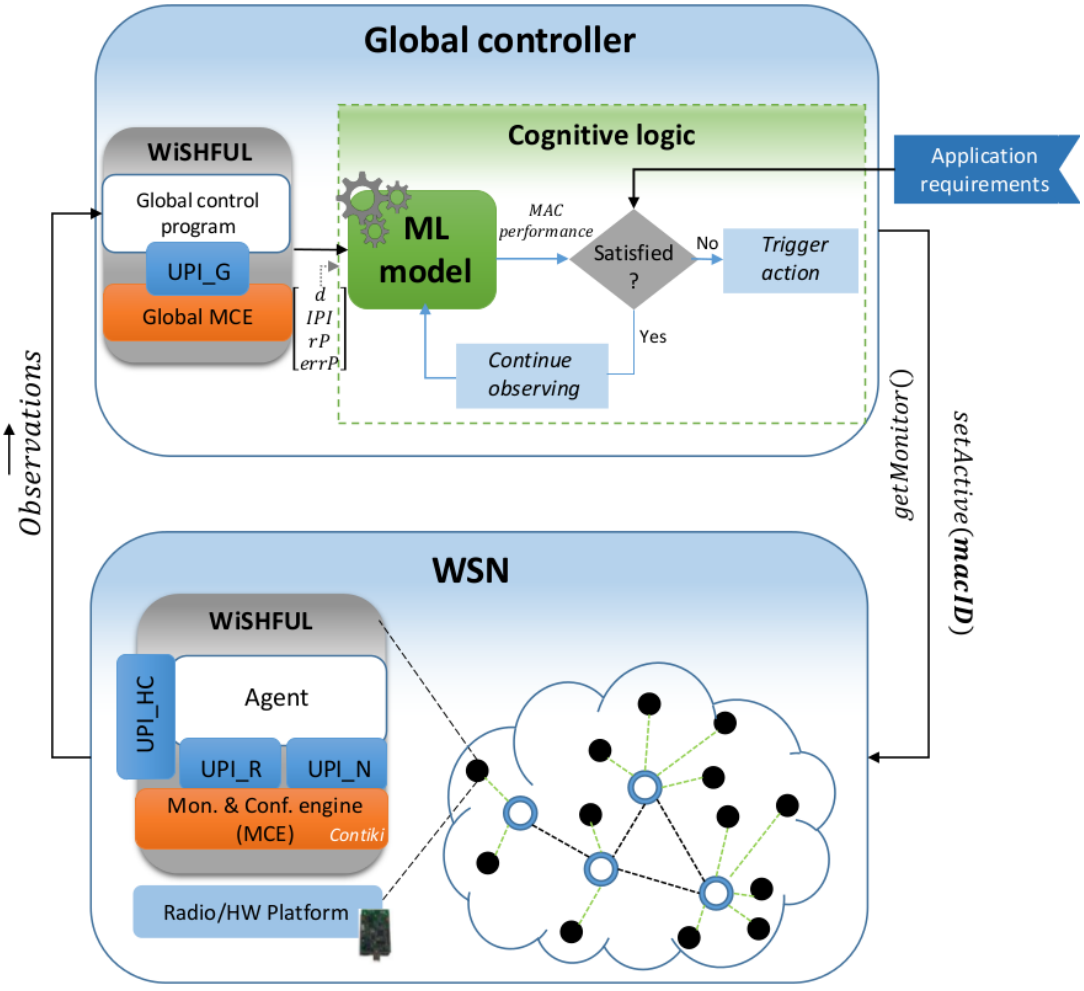


Figure 10 Cognitive MAC architecture

At the heart of the global controller is a machine learning (ML) model that learns the environmental properties and uses the knowledge to predict the future MAC performance. These performance predictions are compared with the application requirements to decide which MAC protocol would be the most appropriate to use.

For instance, a CSMA/CA MAC protocol can achieve low latency in low data rate applications, while in high-data rate applications and under high interference it might significantly underperform due to too unsuccessful channel contentions. On the other hand, a time-slotted channel hopping (TSCH)

MAC can deliver high throughput in high data-rate applications even under interfering transmission from other technologies by avoiding channel contentions and changing the central frequency of the operating channel. However, the channel hopping mechanism that is performed in each subsequent time slot may downgrade the performance in another dimension.

Next steps:

The current results presents a machine learning based MAC performance predictor, which can be used to design a new intelligent MAC layer that can detect a poorly behaving MAC protocol (e.g. CSMA) by predicting its performance in the future (e.g. switch from a CSMA/CA to an alternative more robust MAC like TSCH). This is the basis for our future work, to implement the model in production.

3.1.3 New intelligence modules

The proposed intelligent algorithm for MAC performance estimation can be used as a standalone module and integral part of a smart WSN. However, the final goal of this showcase is to demonstrate that it can be used for intelligent MAC selection.

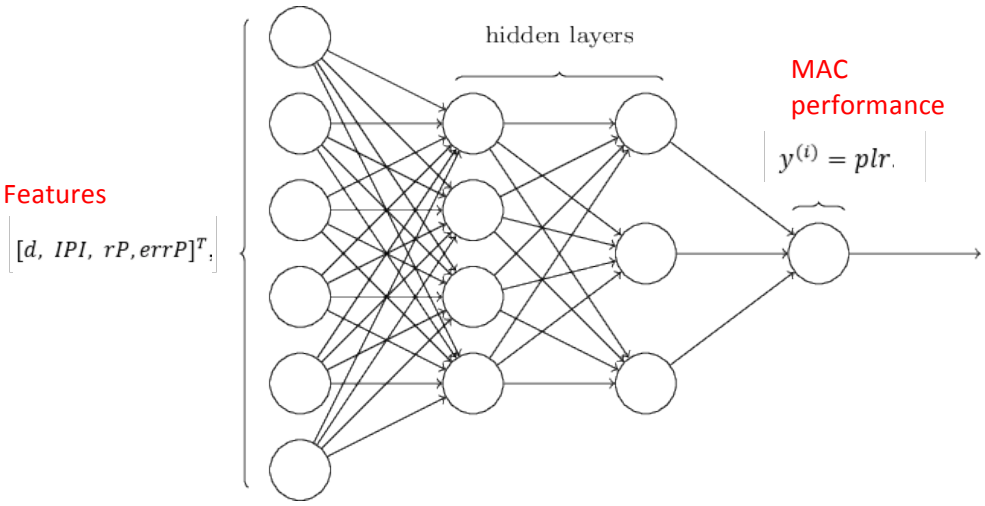


Figure 11 Conceptual neural networks performance predictor

The intelligence module is a trained neural network (Figure 11) MAC performance predictor and can be easily exported from standard machine learning toolboxes/libraries (e.g. Weka [8], scikit-learn [9]).

3.1.4 Results

As the distribution of the process that has generated the data sample is *a priori* unknown, we tested both linear and non-linear regression algorithms. Figure 12 and Figure 13 present the validation performance scores in terms of correlation coefficient and root mean squared error (RMSE). The performance was evaluated with regard to the feature extraction time window.

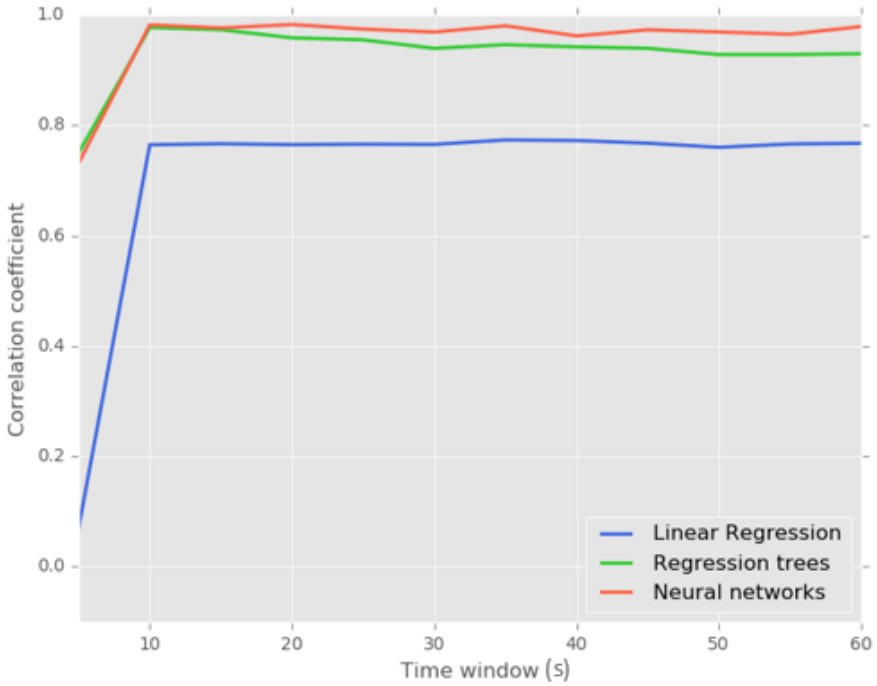


Figure 12 Prediction performance for linear regression, regression trees and neural networks (correlation coefficient)

Figure 12 and Figure 13 show that a neural network with 10 hidden layers (HL), 2000 training iterations, and a learning rate of a = 0:1, captured best the underlying distribution of the data. We selected the neural network model for an observation interval of 30s as an appropriate time window for making accurate future performance prediction.

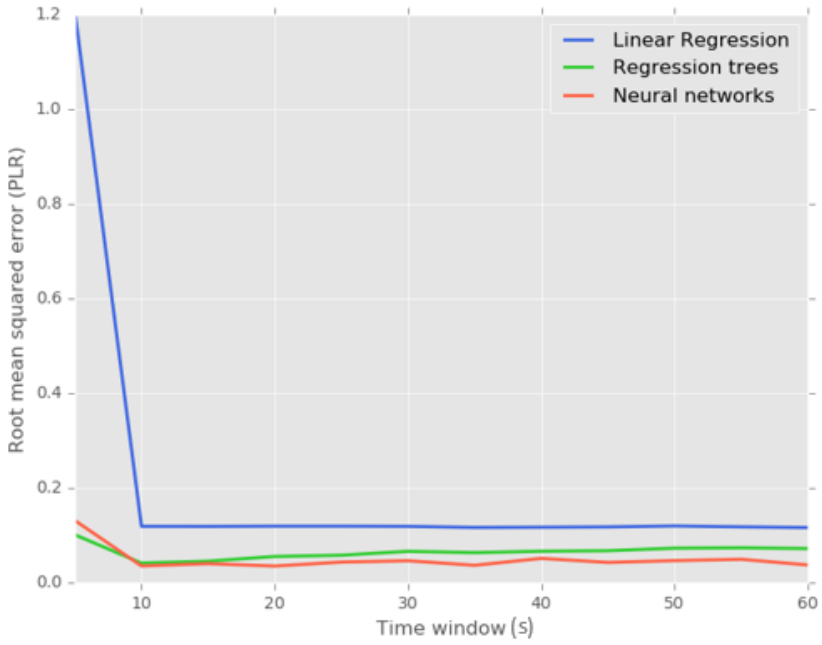


Figure 13 Validation error for linear regression, regression trees and neural networks (RMSE)

In order to validate how well the trained neural network model generalizes a separate set of experiments has been performed with a shorter duration of monitoring (~8h). The new measurements have been used to derive a validation set. The goal of this dataset is to find an estimate of the prediction error of the model in the future.

Figure 14 illustrates how well the model predicts plr on instances from the second set of experiments (the validation set). It can be seen that the predictions are good under changing environmental properties: d, IPI, and interference scenarios.

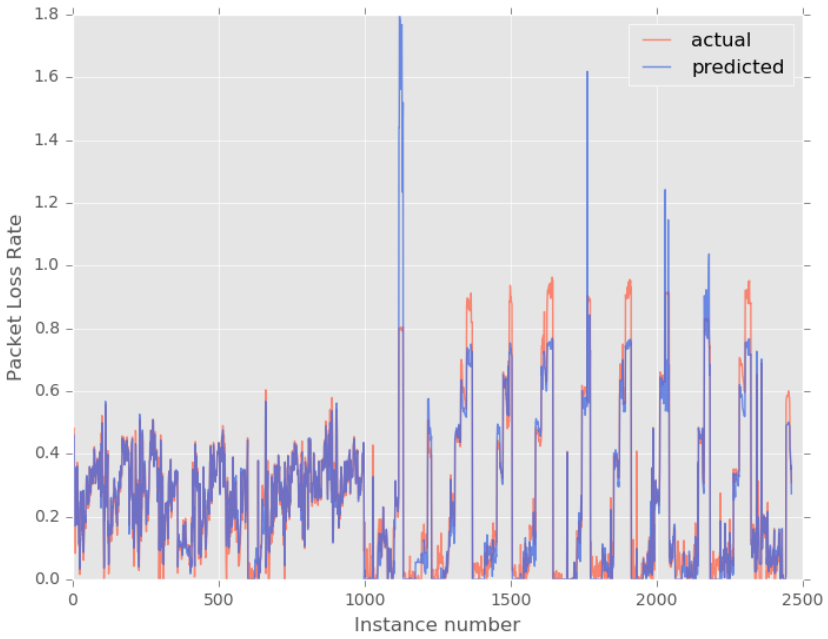


Figure 14 Performance of neural networks predictions vs. actual values on the test data

3.2 MAC optimizations in high-density scenarios (HIDE)

In this showcase we describe the HIDE intelligent module (Intelligent Module for HIgh DEnsity networks). The HIDE module recognizes the presence of hidden and exposed nodes in high-density scenarios. In facts, it is well known that in case of high-density deployments, WiFi networks suffer from serious performance impairments due to hidden and exposed nodes, belonging to overlapping BSSs (OBSSs) that interfere each other [10]. In dense environments, pathological topologies with hidden and exposed nodes are more probable to occur than in sparse wireless networks.

3.2.1 General overview

This showcase uses the HIDE intelligent module. It offers several capabilities including *advanced detection* of specific inter-BSS interference conditions, *context recognition*, focused on hidden nodes detection, and appropriate *actions* about MAC adaptation. Currently the detection phase, the creation of the classifier and its training are run offline, as a Matlab script trained with simulation data. Then, this provides decisions that are taken on nodes in the testbed.

Recognizing hidden nodes

To validate hidden node creation and recognition we use an archetypal tunable topology composed by four nodes: two APs and two STAs.

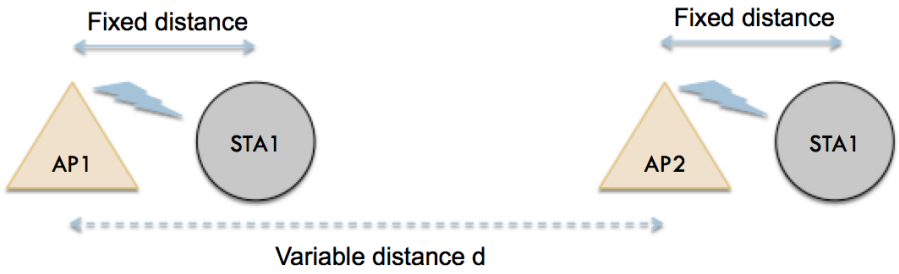


Figure 15 - Archetypal topology for the hidden node problem

Despite the simplicity of the used topology with hidden nodes, this is difficult to obtain because we require for tuning the distance between these two BSSs: AP1-STA1 and AP2-STA2, in order to show the desired phenomena.

Our topology differs from the typical three-nodes topology that is usually used for explaining the hidden node problem. This is because we aim to demonstrate that the hidden node phenomenon has different behaviour depending on some topological parameters when more BSSs overlap in dense networks. To explain the hidden node phenomenon arising within dense networks, we refer to the topology depicted in Figure 15.

There are two BSSs both with downlink traffic, therefore frames go from APs to STAs. The distance between the STA and its AP is fixed and they have an high-SNR link, while the distance between the two BSSs changes. By tuning the inter-BSS distance we change the network density and it has been demonstrated that different levels of interference conditions occur: hard, soft and zero interference [10]. These are respectively indicated with letters H, S, and Z in Figure 19, where distance spans from 0 to 200m. We exploit such interference classification for designing the advanced detection system, which works also in generic topologies and is therefore able to classify interference patterns in multi-hop wireless networks

First, to understand the effects of interference phenomena, we run testbed experiments organized into three phases lasting 120s each, as reported in Figure 17.

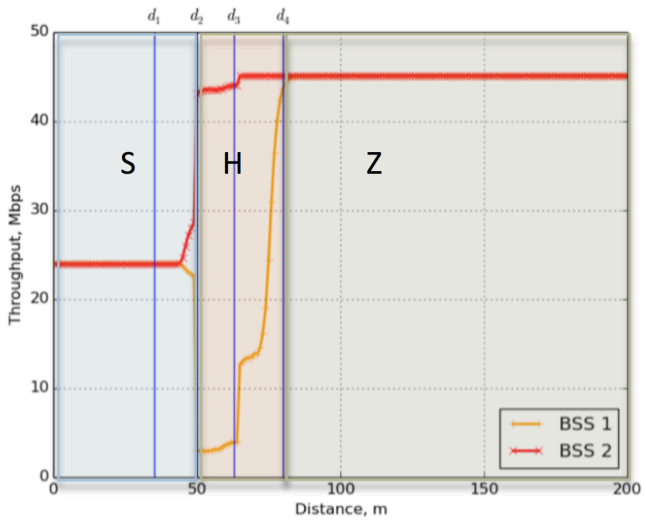
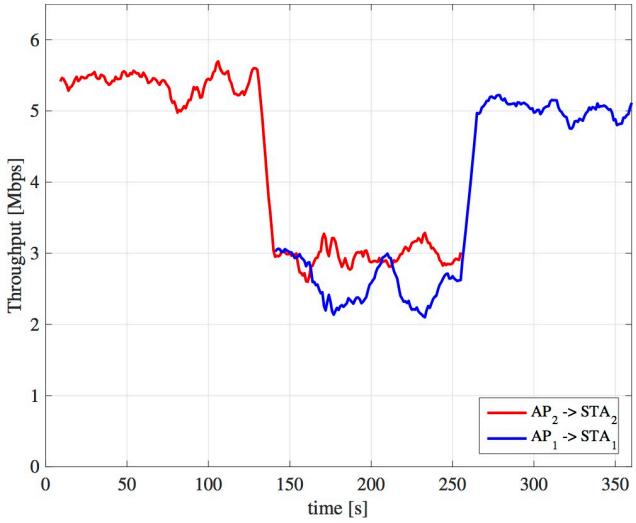
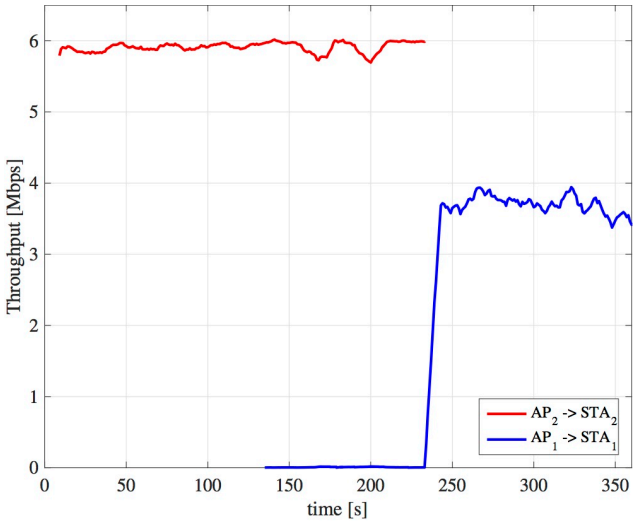


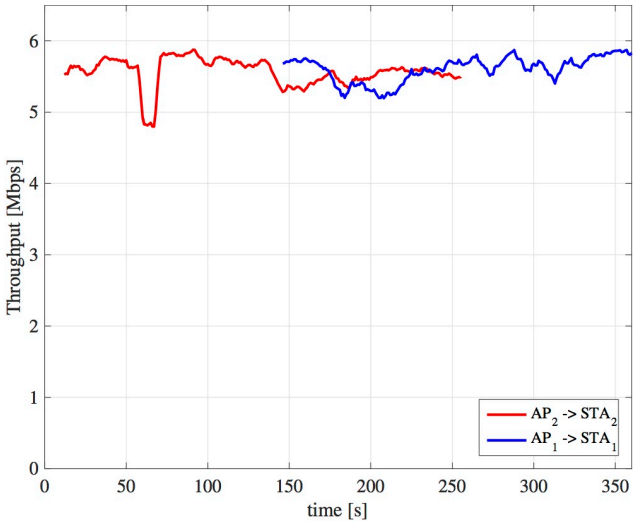
Figure 16- Interference classification depending on distance between the two overlapping BSSs



(a)



(b)



(c)

Figure 17 - Throughput in downlink of the two BSSs under different inter-BSS interference conditions: soft (a), hard (b), and zero interference (c).

These phases permit to evaluate effects due to reciprocal interference between the two BSSs. During the first phase, only one UDP flow from AP2 is active, then the two APs run their flows in parallel, finally only AP1 flow remains active. We tuned the transmission power and used attenuators in order to demonstrate what happens at variable distances: in Figure 17a the APs transmit at the maximum transmission power of 20dBm and can perfectly hear each other; in these conditions, they basically share the channel capacity and get one half of the total throughput each, as expected in the soft interference area. The phenomenon gets worse in Figure 17b, corresponding to the hard interference area when the transmission power is reduced to 1dBm and the two APs consistently fail in sensing each other and the hidden node phenomenon completely ruins performance of the first BSS. Finally, in Figure 17c we report results corresponding to the no interference area, obtained using 6dB attenuators for further reducing the inter-BSS interference and we observe that the two BSSs work as in isolated conditions, because they do not interfere with each other. These results have been obtained without RTS/CTS and for a data rate set to 11 Mbps. The effects of using RTS/CTS have been verified in simulation [10], demonstrating that this doesn't provide much help.

Distributed and centralized intelligence

This showcase uses a twofold approach for recognizing performance impairments due to interference, which have their specific data collection modules: the centralized and the distributed approaches, whose only difference resides in the used features for the interference classification. In the distributed case we feed the classifier with busy/idle data while in the centralized case we provide topological information.

In fact, the advanced detection capability is implemented as a classifier both in the distributed and in the centralized cases. The classifier is trained with supervised learning and is able to predict interference conditions experienced by wireless nodes.

To have comparable results we operate normalization and consider in output the throughput over the offered traffic.

$$y^{(i)} = \frac{\text{throughput}}{\text{offered traffic}}$$

This normalization works also for nodes that have heterogeneous traffic requirements, despite in our simplified scenario we assume all nodes with saturated traffic.

The advanced detection module predicts node interference conditions, which we classified as hard interference (H), soft interference (S) and zero interference (Z). H, S and Z interference zones correspond, respectively to normalized throughput in [0 ; 0.25], in]0.25 ; 0.75], and in]0.75 ; 1].

Data collection module for the distributed approach

In the distributed approach the classifier uses low-level observations given by **busy and idle** traces, taken using WiSHFUL UPIs. These traces are taken locally on the nodes, then values taken in a specific time window are summarized in numerical busy-idle features as follows.

First, the distributed data collection module computes the cdf (cumulative distribution function) of buys/idle durations, then it reads the cdf in specific points corresponding to pre-defined percentiles, as shown in Figure 18. In our showcase we sample the cdf at the 50th, 90th and 100th percentile.

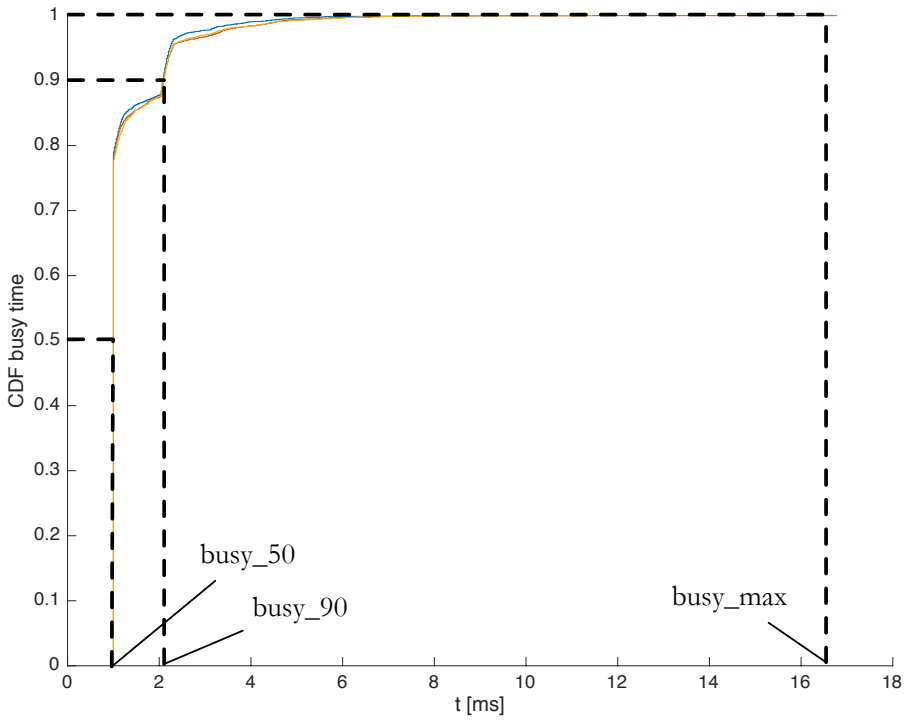


Figure 18– Typical cdf for duration of busy intervals and how busy_50, busy_90 and busy_100 are obtained

Feature space design for the distributed approach

For the distributed approach we used raw data formatted as follows:

$$x^{(i)} = [busy_{50}, busy_{90}, busy_{100}, idle_{50}, idle_{90}, idle_{100}]$$

For the sake of simplification, we assume that nodes send frames with the same size, whose transmission time is 1ms. Despite it is unrealistic to have homogeneous-sized frames, it is not difficult to extend our results to frames of any length. In absence of exogenous sources of interference, durations of busy time are due to transmissions, therefore their length is exactly long as the duration of frame transmission. Busy intervals become longer than the transmission duration in case of collisions. In fact, when one node receives more than one frame concurrently, it results in a collision and in general the colliding frames are not aligned in case of hidden nodes. Moreover, several collisions can occur in a row, therefore busy intervals become longer and longer. In the case shown in Figure 18 the busy time is till 16 times longer than the frame duration. Busy durations that are one hundred times longer than the frame duration are possible. This explains the importance of this feature in predicting node performance due to interference. Analogous considerations are valid for idle durations.

Nodes are able to distinguish interference conditions due to high-density from weak signals with low SNR, and take the corresponding action. This distinction is made possible because hidden nodes and flow in the middle have specific recognizable 'fingerprinting' in busy-idle traces.

Feature space design for the centralized approach

For the centralized approach we used raw data formatted as $x^{(i)} = [\#neigh_{1hop}, \#neigh_{2hops}]$.

In facts, interference depends on network topology and we used the number of transmitting nodes at one hop and at two hops as relevant feature to model interference. In this case, the numbers of nodes at one hop, as well as their unique identifiers are sent to the global controller, reporting also

the MAC addresses of heard nodes. The HIDE intelligent module assembles these contributions and computes the whole network topology by obtaining the adjacency matrix A. Given the set V of wireless nodes, which are the vertices of the graph, the adjacency matrix is a square $|V| \times |V|$ matrix whose generic element a_{ij} is one when node i listens (and is listened from) j, and zero otherwise. The diagonal elements of the matrix are all zero, since nodes have not self-loops.

Given the adjacency matrix the HIDE module computes the number of two-hops neighbours for each node. This is easily obtained from $G^2 - G$, where G^2 is the square of the graph G. These two observations, namely **the numbers of one-hop and two-hops neighbours** permit to run a predictive analysis about performance, under the assumption of all nodes offer saturated traffic.

HIDE protocol selection and inter-BSS synchronization

HIDE implements an if-then-else logic for protocol selection based on predictions provided by the advanced detection model. The MAC protocol is chosen between CSMA and TDMA and in case of TDMA the slots are dynamically assigned per BSS rather then per station. TDMA frames are assigned per BSS. In order to align times slots of adjacent BSSs we use the time synchronization function (TSF), which is natively provided by IEEE 802.11 standard and that we extended in a non-standard way. More into detail, we consider a synchronizing AP that provides its TSF to AP1 and AP2, which in turn adjust their internal TSF. This extra AP (which is not reported in Figure 15) is heard by both transmitting APs, its presence is guaranteed by the high-density assumption. The comparison between obtained performance with/without inter-BSS synchronization is schematically reported in Figure 19. This shows performance degradation due to intra-BSS TDM when no synchronization is employed with overlapping BSSs.

In this showcase we used the WMP platform, already discussed in detail in previous deliverables. We started from the state machine implementing an intra-BSS TDMA protocol and extended it by adding the inter-BSS synchronization mechanism. This had the following impact on WiSHFUL UPIs:

- Added a new action sync_by_ap, parameterized with the MAC address of the synchronizing AP. The synchronized AP adjusts its local TSF upon the reception the new information about channel reservations.
- (future work) From the adjacency matrix, APs in specific positions are elected and their beacons are used for inter-BSS over-the-air synchronization. This common reference signal is then used by the TDMA state machine for setting the slots in the BSS reserved interval in which the STAs are allowed to access the channel.
- Assign the interval within the frame to the two BSSs, as a configuration parameter (e.g. odd slots are assigned to AP1 and even slots to AP2).
- Tune the interval size, which has been set to the minimum interval required for transmitting a frame of 1500 bytes at 11 Mbps, in order to test our system under tight synchronization requirements.

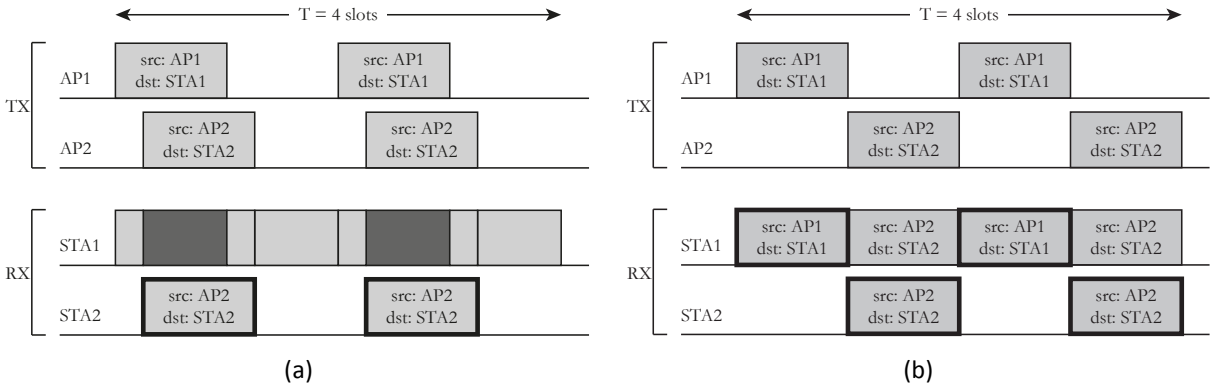


Figure 19 – When the two BSS use unsynchronized TDMAs, STA1 receives no frames and STA2 receive two frames (a). Using synchronized TDMAs, STAs receive two frames each (b).

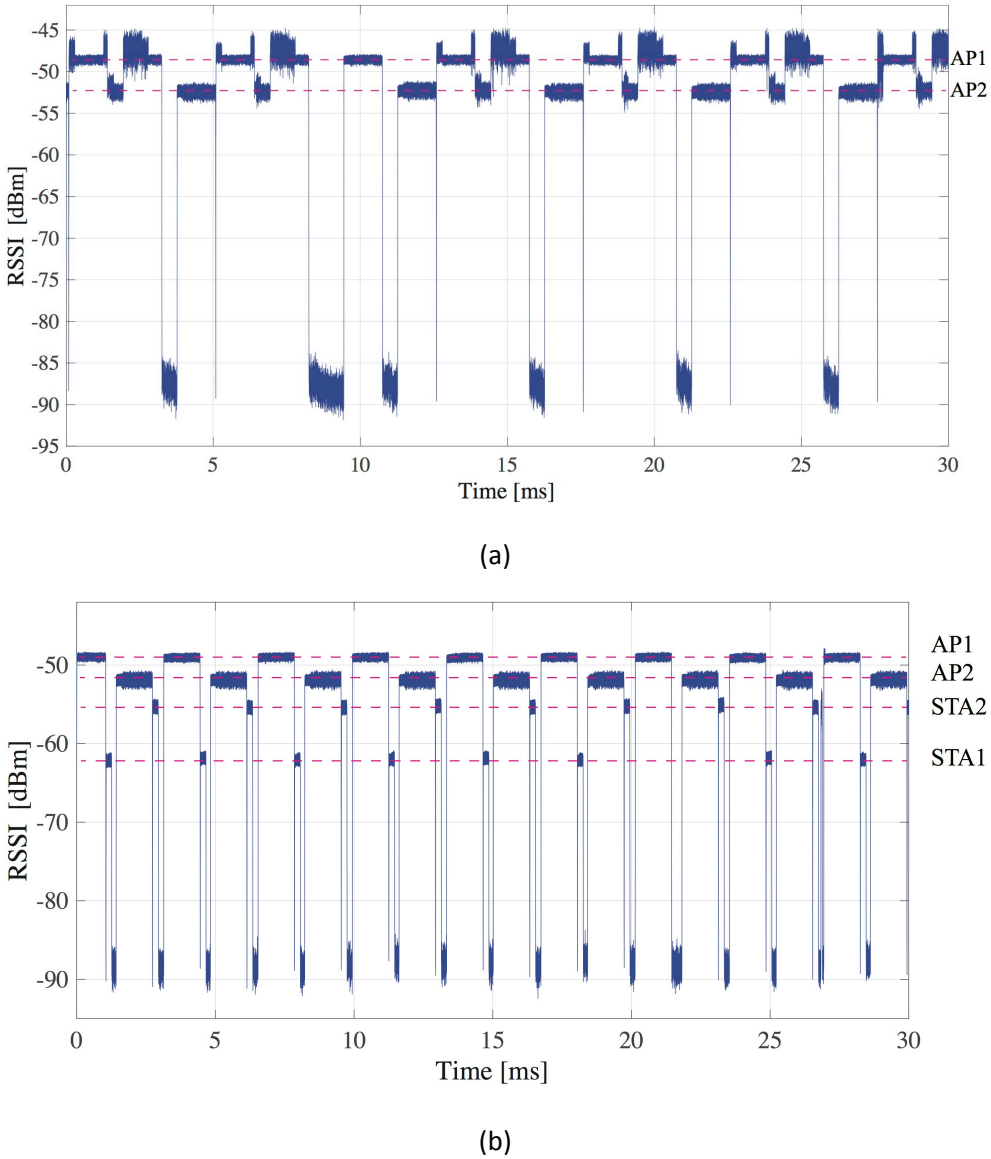


Figure 20 – Real trace obtained from BSSs with un-synchronized (a) and synchronized TDMAs (b)

Introducing inter-BSSs synchronization is beneficial for network performance, as demonstrated in Figure 20, which reports the channel occupancy with (b) /without (a) inter-BSS synchronization. The

power envelope grabbed with an USRP used as a PHY sniffer demonstrates that without synchronization several collisions occur, represented by the overlapped power masks. In this showcase we use downlink traffic, therefore long data frames are sent by APs and short ACKs are sent by STAs, as it is shown in figure by different power levels and durations.

3.2.2 WiSHFUL functionalities and showcase phases

In this paragraph we explain the use of *the WISHFUL framework and UPIs to implement the HIDE intelligent module.*

Specifically we exploit the following main functionalities supported by the WiSHFUL UPI:

- Run local control program that tune and switch between TDMA and CSMA protocols;
- Collect low level measurements on the busy-idle channel utilization (the busy-idle trace is composed by a sequence of durations for the idle and busy times);
- Compute the adjacency matrix;
- Receive the TSF from a synchronizing AP;
- Adjust the local TSF;
- Select the radio program and its parameters for each BSS;
- Synchronize over multiple BSSs by the mean of one beaconing signal taken as common reference;
- Select radio program parameters (the number of TDMA slots assigned to each BSS).

The showcase runs on three main phases: *detection*, based on measurements and statistics, *decision* and *action*.

3.2.3 New intelligence modules

The HIDE intelligent module for advanced detection and interference classification exploits an ensemble classifier and in particular the bootstrap-aggregated (bagged) trees algorithm [11]. Individual decision trees tend to overfit whereas bagged trees are more robust because they combine results of many decision trees. Bagged trees improve generalization in comparison to individual decision trees because they use bootstrap samples of the data and select a random subset of predictors to be used at each decision split.

3.2.4 Results

Results from this showcase aim to evaluate the **efficacy of recognition capabilities, both in centralized and distributed cases, as well as the efficacy of applied MAC countermeasures.**

To present the performance evaluation of interference classification we use confusion matrices. These report the number of elements and percentages of nodes that belong to the true classes H, S, and Z in rows (hard, soft, and zero interference) and fall within the corresponding predicted classes in columns. For example, in Figure 21a it results that 1951 observations were in total affected by true hard interference. The classification algorithm was able to rightly predict the right class H in 92.2% of cases (the true positive rate TPR), while in 7.8% it was wrong (the false negative rate FNR). In facts, 143 observations were classified as S (7% of cases) and 10 as Z (0.1% of cases), even if they belonged to the H class. This means that the mistaken classification of hard interference to zero interference is quite improbable, while there is a residual but significant probability to be confused with soft interference. The same considerations hold for the S and Z rows.

Results in Figure 21a and b are related to decentralized intelligence that observes busy/idle traces. This approach brings to 93.1% of accuracy. In Figure 21c and d we use centralized topological information exploited by the global controller and resulting in 77.9% of accuracy. Poorer accuracy in the centralized approach may appear counterintuitive, but can be explained by the limited information provided, composed only on topological aspects. For the sake of completeness, despite not relevant for this showcase, we also trained the centralized classifier with *all* the described features, obtaining 96% of accuracy.

In Figure 21b-d there are shown the receiver operating characteristics (ROCs). These diagrams consider the performance of a binary classifier as its discrimination threshold is varied. In our case the classifier is not binary, therefore we assimilated it to binary by considering two classes: the hard interference class and another class containing soft and zero interference. These curves are obtained by plotting the true positive rate (TPR, or the probability of detection) against the false positive rate (FPR, or probability of false alarm) at various threshold settings.

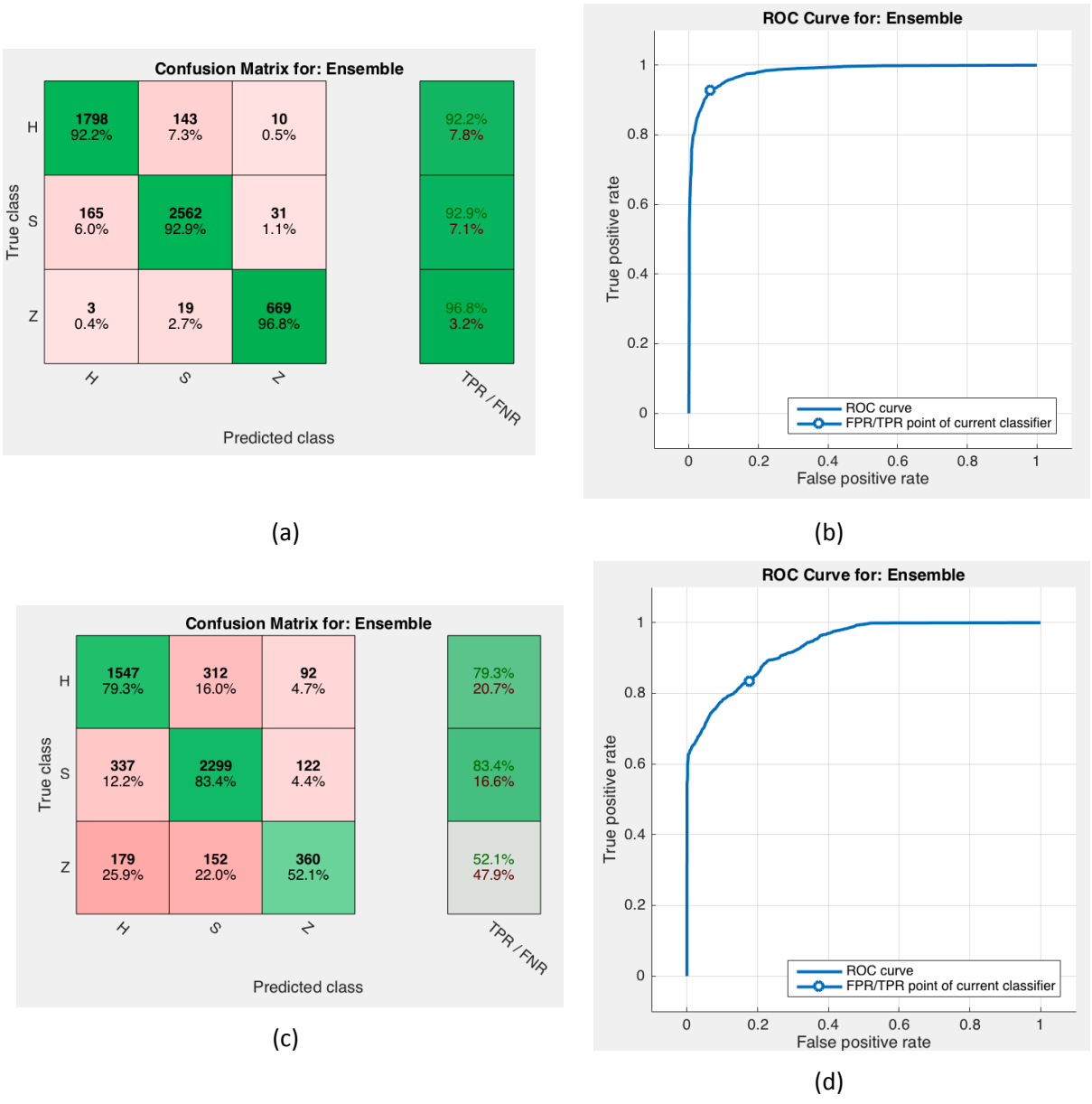
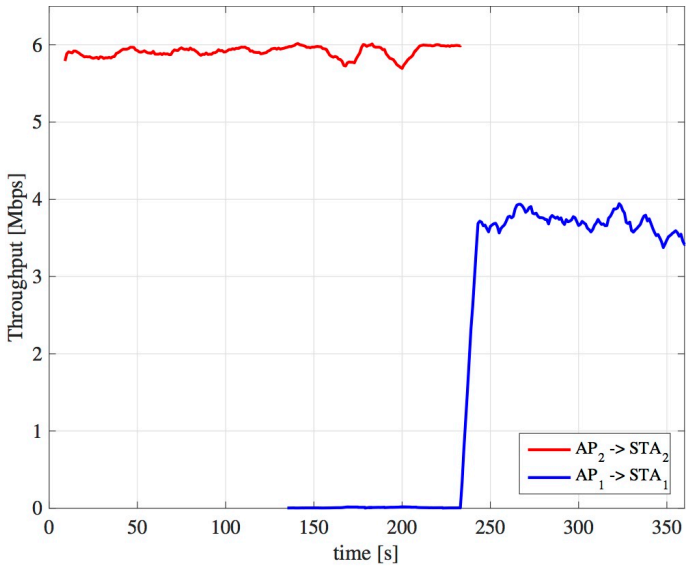


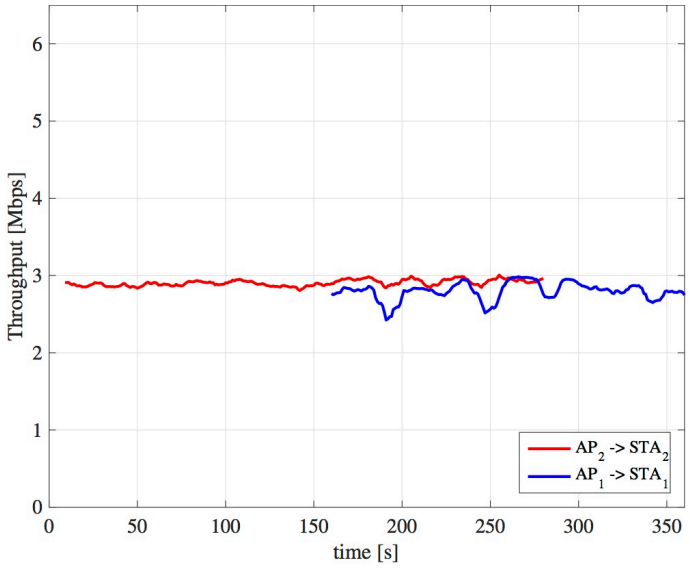
Figure 21 – Classification results for interference using radio features in the distributed approach (three features for busy and three for idle) (a,b) and using only topological features in the centralized approach (neighbours at one hop and at two hops) (c,d)

Finally, we show the impact of *MAC adaptation action* by evaluating throughput in case of CSMA, static inter-BSS TDMA and dynamically assigned inter-BSS TDMA. In static assignment, if one BSS has no traffic to transmit, its assigned interval is wasted. With dynamic assignment, this slot can be reassigned to the remaining BSS.

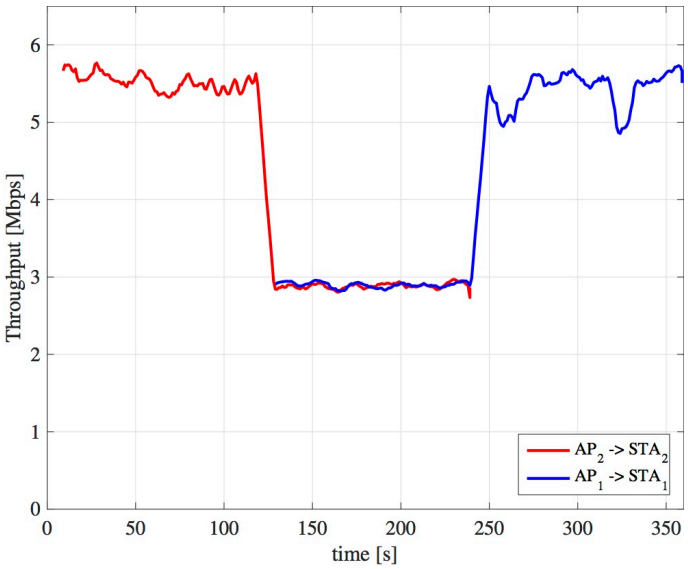
As we have only two overlapping BSSs, we consider a periodic frame of two intervals. We assign odd intervals to BSS1 and the even ones to BSS2. Obviously, this static assignment is not optimal when only one traffic flow exists (see Figure 22b). In the dynamic scenario, we assume that the assignments of reserved intervals can vary over time, according to the load experienced in each BSS. When only one BSS is active, both reserved intervals of the frame are assigned to the active BSS; when both the BSSs are active, each BSS receives only one assignment according to the usual odd/even schedule. Fig. 7 shows the throughput results obtained under inter-BSS TDMA, in the static and dynamic case. Results are valid with all interference classes because the adoption of time-division makes the two BSSs completely orthogonal.



(a)



(b)



(c)

Figure 22 - Throughput for CSMA in case of hard interference (a), TDMA with statically assigned slots in all interference conditions (b) and TDMA with dynamically assigned slots in all interference conditions (c)

3.2.5 Next steps

This showcase presents three possible directions for future improvements. First, we foresee to look for different low-level observations for better interference classification. Then, we propose to demand to the intelligent module the dynamic selection of one or more beaconing APs for the inter-BSS synchronization, through the analysis of the network adjacency matrix. Finally, the recognition phase can be included online.

4 Intelligence showcases based on on-line learning

4.1 Distributed intelligent selection of MAC protocol: Meta-MAC

4.1.1 General overview of Meta-MAC

In this showcase we demonstrate how learning mechanisms can be implemented in each node, based on channel observations gathered by means of UPI functions. In particular, the general idea of the showcase is defining a medium access logic, which is not based on a pre-defined protocol, but rather is given by a combination of elementary protocols, which contribute to a final medium access decision by weighting opportunistically the decisions of each one.

The basic idea of *MAC learning* has been introduced some years ago, in the so called Meta-MAC protocol [12]. Meta-MAC introduces a method for systematically and automatically combining any set of existing protocols into a single MAC protocol. Each protocol runs in parallel and takes a decision at each channel slot; the final medium access decision is obtained by combining the decision of each protocol. At the end of the slot, protocols decisions are classified as or not correct in case they lead to successful transmissions, collisions or wasted channel time and their weights are updated accordingly. For implementing such a scheme in a real network, where protocols cannot be executed in parallel, we exploit the **formal definition of simple MAC protocol components** and **platform-independent representation of channel events** gathered from the wireless node. The idea is emulating the behavior of Meta-MAC by virtually executing multiple protocol components, learning about the performance of protocol components not running in the node, and dynamically reconfiguring the wireless nodes by loading the best expected protocol.

MAC components can be different protocols or protocols with different parameters. For example, slotted **p-persistence** protocol with dynamically adjusted retransmission probabilities p may be regarded as one. In a slotted p-persistent protocol, whenever there is a packet queued, the probability of transmission in a slot is a constant p, independently for each slot. If we dynamically change the value of p, then we effectively combine p-persistent protocols that differ in their p values. Thus, in each slot we decide to use one of these component protocols, namely the one with the appropriate p. Conversely, **TDMA** protocols are characterized by the frame length and by the slot allocated to each node. For a given frame size, different allocations are mapped into different protocol components.

In this showcase we consider two type of slotted protocols, **TDMA** and **p-persistence** protocol; we also use channel feedback at the end of each slot, for updating the weight of each protocol as a function of its right/wrong decision. While for protocol simulation the channel feedback has to specify many different events (such as the reception of a specific frame), for estimating the correctness of the protocol decisions is generally enough to consider a ternary feedback (transmission occurred, collision occurred, or idle slot).

As Figure 23 shows, M MAC protocols $P_1,\ldots,P_M$ are combined at a given node; here M is not related to the number of nodes in the network, N. In order to simplify the presentation, time is divided into slots. Each protocol P_i runs locally and in each slot t produces a decision $D_{i,t}$, $1 < i < M$, where $D_{i,t} = 1$ is interpreted as P_i transmits in slot t and $D_{i,t} = 0$ is interpreted as P_i does not transmit in slot t. A value $0 < D_{i,t} < 1$ is interpreted as a probability with which P_i transmits. The meta-MAC protocol is an algorithm that runs locally at each node and combines the local decisions $D_{i,t}$, $1 < i < M$, to produce a combined result D_t with the same interpretation as $D_{i,t}$. The final binary decision $\epsilon \{0,1\}$ is derived from D_t by drawing a random binary value that takes the value 1 (transmit) with probability D_t and the value 0 (do not transmit) with probability $1\text{-}D_t$.

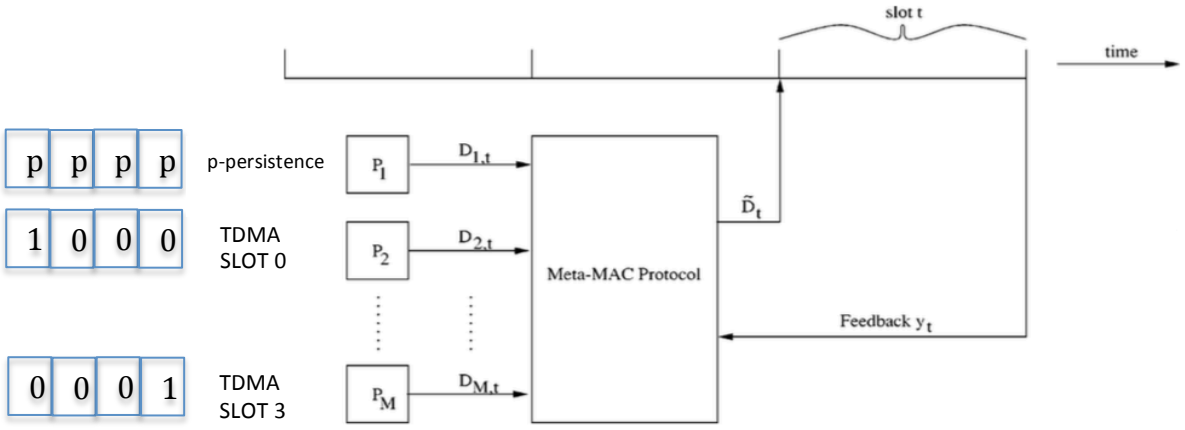


Figure 23 - Operation of the meta-MAC protocol

The combined decision D_t is computed as a function of the weighted average of the $D_{i,t}$ values:

$$D_t = F\left(\frac{\sum_{i=1}^{M} w_{i,t} D_{1,t}}{\sum_{i=1}^{M} w_{i,t}}\right)$$

F is a function that grows linearly from 0 to 1 in an interval *[1/2-c, 1/2+c]* and is truncated to 0 and 1 before and after the interval *c,* which depends on another parameter *η* that controls how the weights are updated. The meta-MAC protocol maintains the weights of each protocol P_i at time t as $w_{i,t}$. At the end of each slot the weights are updated using the channel feedback. In a *ternary* feedback model, a node can determine whether a successful transmission occurred, a collision occurred, or the channel remained idle in a slot. We require information to be available for the meta-MAC protocol to conclude *at the end of the slot* whether the decision for the slot was correct.

For example, from the ternary feedback we can conclude the *correctness feedback*: If we decided to transmit and the transmission was successful then the decision was correct. However, if a collision occurred then the decision was incorrect. If there is a packet queued for transmission but we decide not to transmit there are two possibilities. If the channel was idle the decision was incorrect because the slot was wasted. However, if the channel was not idle then it was correct not to transmit as the channel was used by another node. If the queue was empty, then refraining from transmission was correct. Given such correctness feedback, the weights are updated as follows. Let *y_t* denote the correctness feedback:

$$y_t = \begin{cases} 1 & \text{if the decision in slot t was correct} \\ 0 & \text{if the decision in slot t was incorrect} \end{cases}$$

Then the correct decision for slot *t* is *z_t = ~{D}_ty_t + (1-~{D}_t)(1-y_t)*. But we cannot set the decision for slot *t* to *z_t* because *z_t* only becomes known at the end of the slot. Using *z_t*, the weights are updated as :

$$w_{i,t+1} = w_{i,t} \cdot e^{-\eta |D_{i,t} - z_t|}$$

The constant *η > 0* controls how fast the weights change. This update rule can be interpreted as reflecting the ``correctness history'' of the component protocols.

4.1.2 WiSHFUL functionalities and showcase phases

In this showcase we want to demonstrate how the *WISHFUL framework and UPIs can be exploited for implementing the meta-MAC algorithm.* In the WiSHFUL framework, elementary MAC components are implemented in terms of radio programs, but only one radio program can be executed at run-time. Therefore, the Meta-MAC implementation is based on the possibility of virtually executing the

protocols on the basis of their formal representation and channel feedback. To this purpose, we exploit the following main functionalities provided by the WiSHFUL UPI:

- Run a local control program that implements the meta-MAC algorithm in each node, updates the weights of the protocol components and loads the best protocol on each node;
- Collect low level channel measurements, in order to get a feedback at end of each time slot and classify protocol decisions are correct or incorrect;
- Tune radio program parameters;
- Switch from a radio program to another.

Although the showcase is based on a distributed intelligence, developed within the local controllers, we also use a global control program for the set-up of the network, the loading of the intelligence modules and the activation of the traffic streams.

Network scenario. We consider a wireless network with 5 active nodes under the same contention domain (where all the nodes are in radio visibility): 1 Access Point node (AP) and 4 Station nodes (STA1…STA4) associated to the same AP. A wired Ethernet network is available as a control network (Figure 24). Each STA node runs the meta-MAC logic loaded by the global control program for tuning the current protocol as a function of the network condition. Our implementation supports 4 different versions of a TDMA protocol with a frame size equal to 4 slots (each version corresponding to a different slot assignment) and one slotted ALOHA protocol with a tunable persistence probability p.

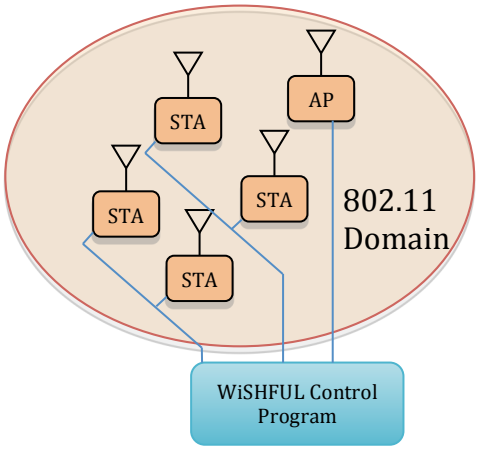


Figure 24 - Network topology used in the showcase

Storyline. For demonstrating the effectiveness of the general Meta-MAC implementation, the showcase generates a configurable channel occupancy pattern and shows how a Meta-MAC node is able to find automatically the best protocol that improves the network performance in dynamic traffic conditions.

Phase 1: at the beginning of the experiment, the network works under sporadic traffic. In this case, the meta-MAC logic selects the slotted ALOHA protocol with a high p-persistence value, in order to optimize the time delivery of the frames.

Phase 2: as long as the traffic rate increases and the network load reaches saturation conditions, the meta-MAC moves from the slotted ALOHA protocol to the TDMA; moreover, it is able to find a non-conflicting scheduled (in a distributed way) among all the contending nodes..

Summarizing, after the set-up of the wireless network, the showcase works by executing the following steps performed by the experiment controller:

1) Sending of the local control program from the global MCE to the local MCEs;

2) Performing a low network traffic (300kbps) between STAs and AP ***(phase 1)***;
3) Performing a saturation network traffic between STAs and AP ***(phase 2)***;

Implementation details. For running this experiment, we use two different python scripts: the first script is responsible of network configuration, AP activation and station associations (**metamac_testbed_controller**); the second script is the most relevant one and implements the Global Control Program on top of the WiSHFUL global MCE (**metamac_experiment_controller**), which includes the **metamac_local_control_program** that is sent to the local controller of each node**.**

The showcase code is available on Github in [13].

Figure 25 shows the overall software architecture of the control modules involved in this showcase. The WiSHFUL control framework provides the basic functionalities for developing different control programs, such as the possibility to discover the nodes, their capabilities and the network topology. These functionalities are natively provided by the WiSHFUL framework in the modules **DISCOVERY Nodes** and **GET Nodes Capabilities,** that are included in Global MCE. The meta-MAC Local Control Program provides two different main modules implemented in two separated threads: i) **Meta-MAC LOGIC**, that implements the meta-MAC algorithm, and ii) **COLLECT MEASURE** responsible of gathering the channel feedback consumed by the Meta-MAC LOGIC. The Meta-MAC LOGIC module includes the definition of the following functions: i) updating the weights of the protocol components, ii) deciding about protocol switching, iii) tuning protocol parameters.

Note that the meta-MAC local control program is defined in the global control program (as indicated by the dashed lines in the figure), sent to the local MCEs available in each network node, and executed locally. The figure shows the flows of global control messages in the green arrows, the actions of the local MCE in the red arrows and the data flows in the blue arrows.

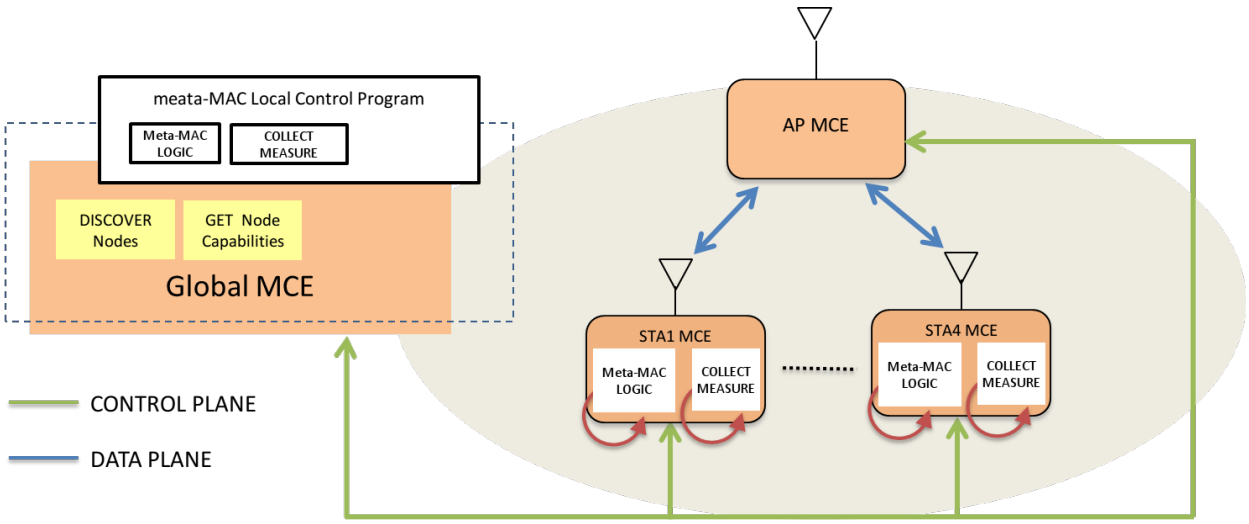


Figure 25 – Module involved in the global and local controllers in this showcase

The feedback variables retrieved by the COLLECT MEASURE module and their semantics are described in Table 2. In addition, the local control program uses a 3-bit slot index counter, representing the current slot number modulo 8, and a timestamp (with a precision in the order of microseconds) that are used to determine the number of channel slots elapsed between consecutive readings of the channel measurements.

Variable	Semantics
packet_queued	One or more packets queued for transmission.
Transmitted	Transmission attempted.
transmit_success	ACK received for successful transmission.
transmit_other	Any data frame or ACK received.
bad_reception	Invalid data frame received.
busy_slot	More than 500us of channel activity in slot.

Table 2 - Feedback variables and semantics

In Figure 26 we show the software architecture used in the showcase, by enlightening the separation between the data processing level (MAC protocol level) and the control layer (Monitoring and Configuration Engine level). The COLLECT MEASURE module use UPI to get time slot feedback and keep available this information to the Meta-MAC LOGIC module, responsible of finding the best protocol component. As shown in Figure 26, the channel feedback records are also stored in a csv file on the nodes, in order to perform other type of post processing. In particular, at the end of each slot, the COLLECT MEASURE module saves six variables with the binary feedback described in Table 2.

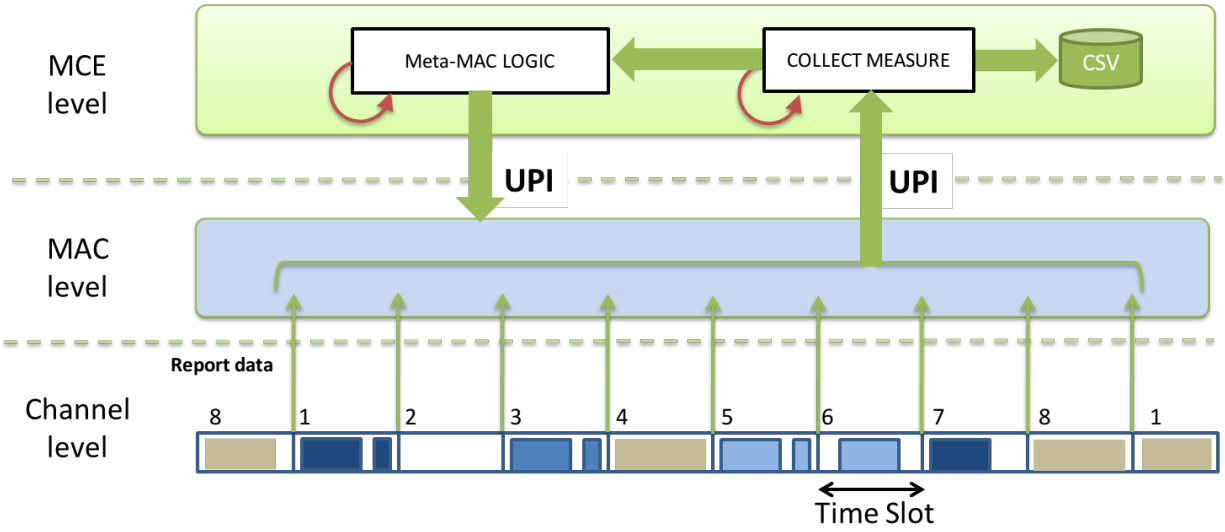


Figure 26 - meta-MAC implementation architecture

On the basis of the previous description, it is possible to easily read an extracted code of the global control program reported below, which is responsible of injecting the local control logic. The local logic is implemented in the *metamac_local_control_program()* function, while *mytestbed* is the set of discovered nodes, and *controller* is the WiSHFUL global MCE. After the local control program is sent to the nodes, the global controller keeps receiving messages from the local controllers by means of the *collect_remote_message* function, for recording the statistics about the throughput results and the protocol choices of each network node. This information is used for visualizing the experiment results.

```
…
''' set default radio program on STA nodes '''
for node in mytestbed.wmp_nodes:
     active_ALOHA_radio_program(node, log, controller, nodes_platform_info[0])

''' start local control program on STA nodes '''
lcpDescriptor_wmp_nodes = []
reading_thread = []
for ii in range(len(mytestbed.wmp_nodes)):
       lcpDescriptor_wmp_nodes.append(controller.node(mytestbed.wmp_nodes[ii]).hc.st
art_local_control_program(program=metamac_local_control_program))
    #start thread for collect measurements from nodes(THR and Active protocol)
    reading_thread.append(threading.Thread(target=collect_remote_messages,
args=(lcpDescriptor_wmp_nodes[ii], socket_visualizer,)))
    reading_thread[ii].start()
…
```

The code below shows the main loop of the global control program. The showcase phase 1 starts when the first traffic stream (300Kbps) is activated on nodes; after 30 seconds, the control program moves to phase 2 with a saturated traffic stream for each nodes.

```
…
#after 30 seconds move from showcase phase 1 to phase 2
CHANGE_TRAFFIC = 30

#SHOWCASE PHASE 1
#start server traffic
controller.delay(2).nodes(mytestbed.ap_node).net.create_packetflow_sink(port='1234'
)
for node in mytestbed.wmp_nodes:
      #start client traffic
       controller.delay(4).nodes(node).net.start_packetflow(mytestbed.ap_node.ip,
            '1234', '500', '0.3M')
while do_run:
    log.warning('waiting for ... (%d sec / %d)' % (dt, EXPERIMENT_DURATION) )
    # 1 second sleep
    gevent.sleep(1)
    dt += 1
    if dt == CHANGE_TRAFFIC:
       #ACTIVE SHOWCASE PHASE 2
       for node in mytestbed.wmp_nodes:
          #start client traffic
          controller.delay(4).nodes(node).net.start_packetflow(mytestbed.ap_node.ip,
              '1234', '500', 'SATURATION')
…
```

4.1.3 Intelligence composition module: meta-MAC logic

As already discussed in the general description of the showcase implementation, the meta-MAC algorithm is performed independently and locally by each node, by executing the custom function metamac_local_control_program(). This function implements the local control program and uses two different threads, which are implemented in the two modules showed in Figure 26. The main loop of the COLLECT MEASURE module is implemented in the following code, where the UPI get_measurements_periodic(parameters) is used to get the feedback of the last 8 time slots (reporting_period/collect_period=16ms/2ms=8 slots). In detail, we use the following UPI parameters:

1. measurement_key_list:
    - **Packet queue; Transmitted; Transmitted Success; Transmit Other; Bad Reception; Busy slot;**

2. collect_period = 2ms
3. report_period = 16ms
4. num_iterations = 1

afterwards the feedbacks are separated slot by slot and stored in the story_channel array list, where the slots information are organized in metamac_slot()structure. The module uses this array list to push the time slots feedback to Meta-MAC LOGIC and to CSV file.

```
collect_period = 2 #ms
report_period = 16 #ms
num_iterations = 1


while getattr(reading_thread, "do_run", True):

      last_tsf = tsf
      last_slot_index = slot_index

      UPI_myargs = {'interface' : 'wlan0', 'measurements' : [UPI_R.TSF,
              UPI_R.COUNT_SLOT, UPI_R.PACKET_TO_TRANSMIT, UPI_R.MY_TRANSMISSION,
              UPI_R.SUCCES_TRANSMISSION, UPI_R.OTHER_TRANSMISSION,
UPI_R.BAD_RECEPTION,
              UPI_R.BUSY_SLOT, UPI_R.COUNT_SLOT] }
      node_measures = controller.radio.get_measurements_periodic(UPI_myargs,
              collect_period, report_period, num_iterations, None )

      tsf =node_measures[0]
      slot_index = node_measures[1]& 0x7
      packet_queued = node_measures[2]
      transmitted = node_measures[3]
      transmit_success = node_measures[4]
      transmit_other = node_measures[5]
      bad_reception = node_measures[6]
      busy_slot = node_measures[7]


slots = [ metamac_slot() for i in range(8)]
ai = 0
while slot_offset > 0 :
      slot_offset-=1
      si = slot_index - slot_offset #(int)
      if  si < 0 :
            si = si + 8

      slot_num+=1
      slots[ai].slot_num = slot_num
      slots[ai].read_num = read_num
      slots[ai].tsf_time = tsf
      slots[ai].slot_index = slot_index
      slots[ai].slots_passed = slots_passed
      slots[ai].filler = 0
      slots[ai].packet_queued = (packet_queued >> si) & 1
      slots[ai].transmitted = (transmitted >> si) & 1
      slots[ai].transmit_success = (transmit_success >> si) & 1
      slots[ai].transmit_other = (transmit_other >> si) & 1
      slots[ai].bad_reception = (bad_reception >> si) & 1
      slots[ai].busy_slot = (busy_slot >> si) & 1
      ai+=1

for i in range(ai):
      story_channel.append(slots[i])
```

```
read_num+=1
```

After collecting the time slots feedback, the module Meta-MAC LOGIC is responsible of retrieving and processing the feedback provided by the story_channel array list, and selecting the MAC component protocol to be activated. The module code is reported below and works slot by slot. Once the feedback is retrieved, the module evaluates the decision of each protocol component for the next slot. This is critical because only one component protocol is actually running, while the decisions of all component protocols are required in order to generate the correctness feedback for the meta-MAC protocol. Thus, a software representation of each protocol, embedded in the local control program, must be able to emulate the protocol's decision for each slot as a function of the feedback provided.

```
…
#Performs the computation for emulating the suite of protocols
#for a single slot, and adjusting the weights.
def update_weights(suite, current_slot, ai):
      z = 0.0
      uu = 0.0
      d = 0.0

      #If there is no packet queued for this slot, consider all protocols to be
correct
      #and thus the weights will not change
      if (current_slot.packet_queued) :
            #z represents the correct decision for this slot - transmit if the
channel
            #is idle (1.0) or defer if it is busy (0.0)
            # // transmission AND success: GOOD
            # // no trasmission AND slot busy - GOOD
            # // trasmission AND NOT success - WRONG
            # // trasmission  AND slot empty - WRONG

            if (suite.protocols[suite.active_protocol].emulator == b'tdma') :
                  if (not current_slot.channel_busy):
                        z = 1.0

            if (suite.protocols[suite.active_protocol].emulator == b'aloha'):
                  p_curr =
suite.protocols[suite.active_protocol].parameter.persistence

            if (not current_slot.channel_busy):
                        z = p_curr
                  else :
                        z = 1-p_curr

            #evaluate protocols weights
            for p in range(suite.num_protocols) :
                  # d is the decision of this component protocol - between 0 and 1
                  if (suite.protocols[p].emulator == b'tdma') :
                              #Protocol emulation for TDMA
                        d = tdma_emulate(suite.protocols[p].parameter,
current_slot.slot_num,
                                    suite.slot_offset)
                  else :
                              #Protocol emulation for ALOHA slotted
                        d = aloha_emulate(suite.protocols[p].parameter,
current_slot.slot_num,
                               suite.slot_offset)


                  #evaluate weight
                  exponent = suite.eta * math.fabs(d - z)
```

```
                suite.weights[p] *= math.exp(-exponent)

                if suite.weights[p]<0.01:
                     suite.weights[p]=0.01

           # Normalize the weights
           s = 0
           for p in range(suite.num_protocols):
                s += suite.weights[p]
           for p in range(suite.num_protocols):
                suite.weights[p] /= s
…
```

Using the feedback, the module maintains weights for each of the component protocols according to the model described in Figure 23. The meta-MAC LOGIC module then actives the radio program and modifies the radio program parameters as necessary to execute the highest weighted component. Finally, the extracted code below shows how UPI functions are used to enforce protocol switching when the protocol with the highest weight is different from the one under execution. The radio program is activated by calling the UPI function activate_radio_program(), and the protocol configuration is specified by the UPI function set_parameters(UPIargs) (responsible of configuring radio program parameters).

```
…
if (protocol == suite.slots[active]) :
     #This protocol is already running.
     pass

elif (protocol.type == suite.slots[active].type) :
     #Protocol active match with the type of best, but is not the same protocol
     #Write the parameters for this protocol.
       UPIargs = { 'interface' : 'wlan0', 'params':
suite.protocols[protocol].params }
       rvalue = controller.radio.set_parameters(UPIargs)
       if rvalue[0] == SUCCESS:
            log.warning('Parameter writing successful')
       else :
            log.warning('Error in parameter writing')
     suite.slots[active] = protocol

elif (protocol != suite.slots[active]):
     #Switch to other Protocol.
     if active == 'TDMA':
          protocol = 'ALOHA'
     else:
          protocol = 'TDMA'

     UPIargs = {'Protocol' : protocol, 'interface' : 'wlan0' }
     rvalue = controller.radio.activate_radio_program(UPIargs)
     if rvalue == SUCCESS:
          log.warning('Radio program activation successful')
     else :
          log.warning('Error in radio program activation')
     suite.slots[suite.active_slot] = protocol
…
```

4.1.4 Results

To evaluate the capabilities of our meta-MAC implementation, we run an experiment in a simple topology of 4 nodes connected to the same AP. Each of these participants runs the meta-MAC protocol with four variants of TDMA and one slotted-ALOHA protocol. Each TDMA variant works with a frame size of four slots but with different assignments in slot 0, 1, 2 o 3, respectively. At the beginning of the experiment, all protocol weights are equal and the TDMA slot 0 protocol is loaded

and activated on each of the participant node. All the nodes transmit at 6 Mbps frames of 1500 bytes in a slot of 2.2ms. The duration of the experiment is 120 seconds. The results of the experiment are shown in Figure 27 and Figure 28.

Figure 27 shows the protocol selected by each node at run time in terms of protocol label. After two second, the traffic is activated in the low-rate conditions (i.e. 300Kbps) and the meta-MAC LOGIC selects the slotted ALOHA protocol with 0.9 persistent probability as the best protocol. After 50 seconds, the experiment controller changes the traffic conditions of the network, by activating greedy traffic sources in each node and by saturating all the transmissions queues. In these conditions, all the nodes switch to TDMA protocols and find a non-conflicting assignment after a transient phase of a few seconds. Figure 28 shows the throughput results achieved by each node under the time-varying protocol execution.

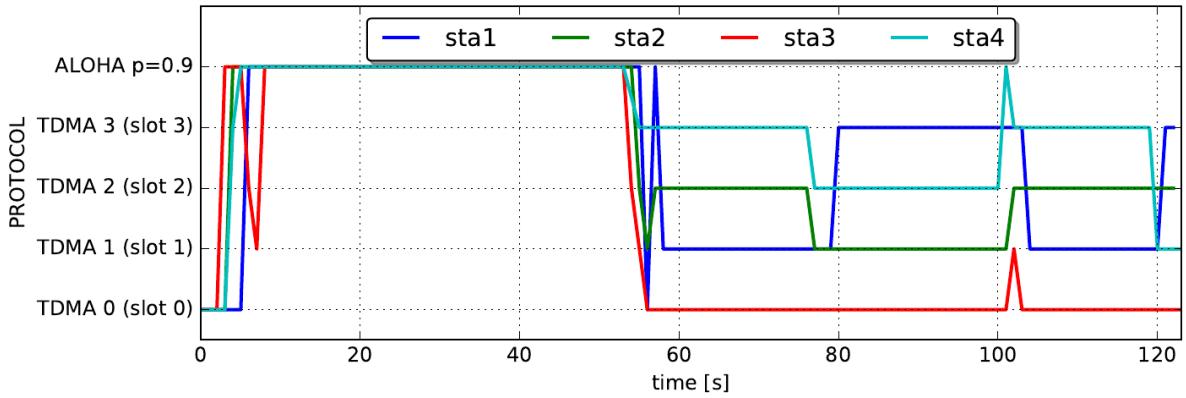


Figure 27 - Selected protocol by each nodes

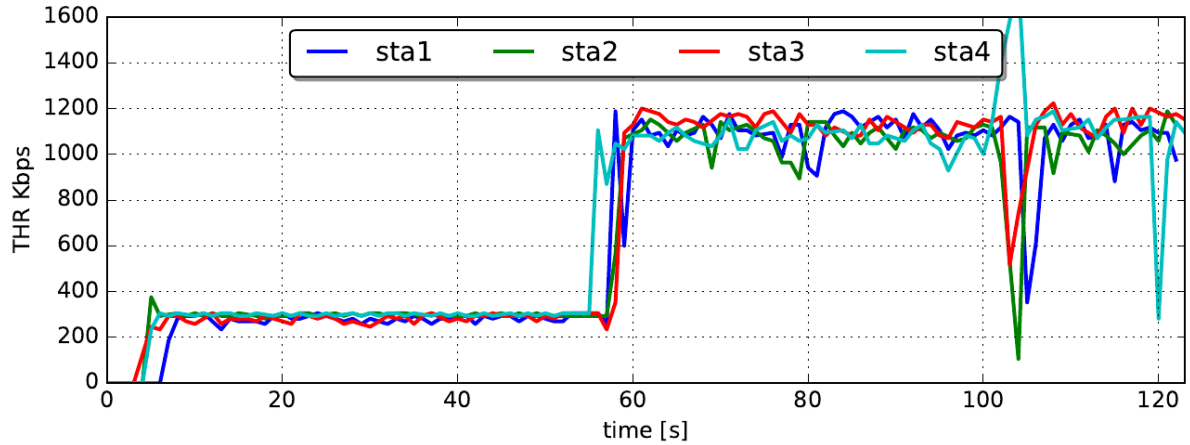


Figure 28 - Throughput performance of 4 wireless nodes executing meta-MAC logic

Finally, Figure 29 shows a channel activity trace acquired by a USRP monitoring node, in terms of received RSSI samples, in which we can easily recognize data transmissions and acknowledgements performed by different nodes (with different RSSI values; the best one, with the shorted duration, is the ACK sent by the AP), according to a TDMA scheme with four slots.

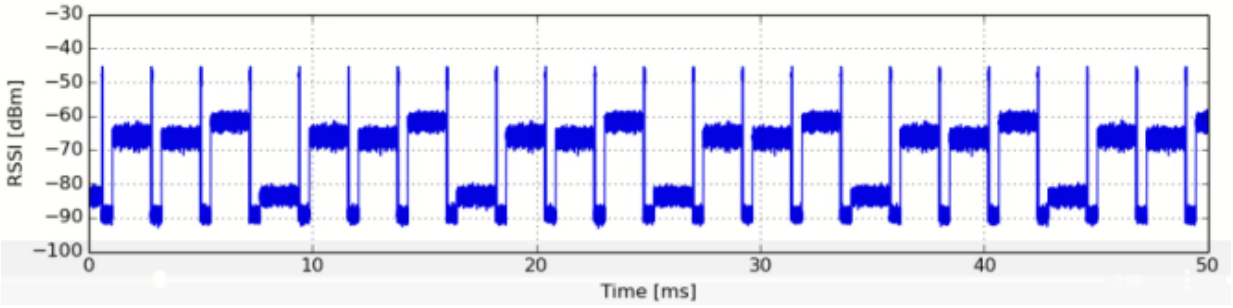


Figure 29 – Channel acquiring when the saturated traffic is present in the network

Figure 30 shows an extract of the CSV file, in which we can read the slots feedback for all the slots between the 36580 and 36596 slot number.

```
slot_num, read_num, host_time, tsf_time,
slot_index, slots_passed, filler,
packet_queued, transmitted, transmit_success, transmit_other,
bad_reception, busy_slot, channel_busy
.........
36580, 7342, 90921618, 936155573, 6, 5, 0, 1, 1, 1, 0, 0, 1, 0
36581, 7342, 90921618, 936155573, 6, 5, 0, 1, 0, 0, 1, 0, 1, 1
36582, 7342, 90921618, 936155573, 6, 5, 0, 1, 0, 0, 1, 0, 1, 1
36583, 7342, 90921618, 936155573, 6, 5, 0, 1, 0, 0, 0, 0, 0, 0
36584, 7343, 90933656, 936167621, 4, 6, 0, 1, 1, 1, 0, 0, 1, 0
36585, 7343, 90933656, 936167621, 4, 6, 0, 1, 0, 0, 1, 0, 1, 1
36586, 7343, 90933656, 936167621, 4, 6, 0, 1, 0, 0, 1, 0, 1, 1
36587, 7343, 90933656, 936167621, 4, 6, 0, 1, 0, 0, 1, 0, 1, 1
36588, 7343, 90933656, 936167621, 4, 6, 0, 1, 1, 1, 0, 0, 1, 0
36589, 7343, 90933656, 936167621, 4, 6, 0, 1, 0, 0, 1, 0, 1, 1
36590, 7344, 90946775, 936180838, 2, 6, 0, 1, 0, 0, 1, 0, 1, 1
36591, 7344, 90946775, 936180838, 2, 6, 0, 1, 0, 0, 1, 0, 1, 1
36592, 7344, 90946775, 936180838, 2, 6, 0, 1, 1, 1, 0, 0, 1, 0
36593, 7344, 90946775, 936180838, 2, 6, 0, 1, 0, 0, 1, 0, 1, 1
36594, 7344, 90946775, 936180838, 2, 6, 0, 1, 0, 0, 1, 0, 1, 1
36595, 7344, 90946775, 936180838, 2, 6, 0, 1, 0, 0, 1, 0, 1, 1
36596, 7346, 90970814, 936204740, 5, 3, 0, 1, 1, 1, 0, 0, 1, 0
.........
```

slot_num

TX TX TX TX TX

Figure 30 - Store channel information in a local CSV file

4.1.5 Next Steps

In the future, it may be worth investigating how the meta-MAC model can be generalized to non-slotted time. We also intend to evaluate the effect of the learning parameter η on the convergence rate and stability of the meta-MAC. A completely general meta-MAC capable of combining any set of protocols would be a decisive leap forward in dynamic adaptation of MAC protocols.

4.2 Monitoring, Reasoning and Decision using Markov Chains

4.2.1 General Overview

In spectrum sharing scenarios, secondary users (SUs) avoid causing interference to primary users (PUs) through the configuration of multiple operating parameters, such as transmit power, frequency, waveforms, scheduling, coding, and antenna patterns. This adaptation to the environment can be performed based on information gathered and decisions taken locally at each node, or can be managed through spectrum-aware network-wide control entities, such as an Spectrum Access System (SAS), geo-location databases, or radio environment maps (REM). The first

approach, followed in the traditional cognitive radio literature, raises many implementation challenges, namely in the design of power efficient local sensing mechanisms that can provide a wideband, long term statistical view of spectrum occupancy, without draining SUs' limited battery resources. As a way to circumvent the limitations of local sensing, regulators and industry bodies have generally shifted their focus towards the use of network-wide spectrum management systems (e.g. SAS in US, and LSA in Europe) that hold a more global and long term view of spectrum utilisation across space and frequency, and can be used to manage SUs' access to the spectrum. On the other hand, there are still several questions and challenges regarding the use of these systems in radio environments where the PUs display a highly dynamic channel access behavior.

In this showcase, we focus on the study of learning and agile channel access/decision making techniques that SUs can employ to tackle the aforementioned issue of PU's dynamics, while being supervised and informed regarding their radio environment by a centralized intelligence spectrum management entity. Our vision is to assess the adequacy of representational statistical models, dimensionality reduction and learning algorithms whose output results efficiently characterize and reliably predict PUs' behavior, are easily transferable/shared across different network elements (e.g. spectrum management system, sensors and SUs), and which SUs can leverage for effective decision making. Examples of such representational models may include tables of observed PU-specific relevant features, and state-transition models, such as Markov Chains (MCs). These features and models can be extracted by a dedicated sensor network or SUs with less restrictive power consumption limits (e.g. base stations), and shared with spectrum management systems that use them as input during intelligence decisions to configure SUs' operational parameters and channel access strategies. The several stages associated to the setup and utilization of this framework are displayed in Figure 31.

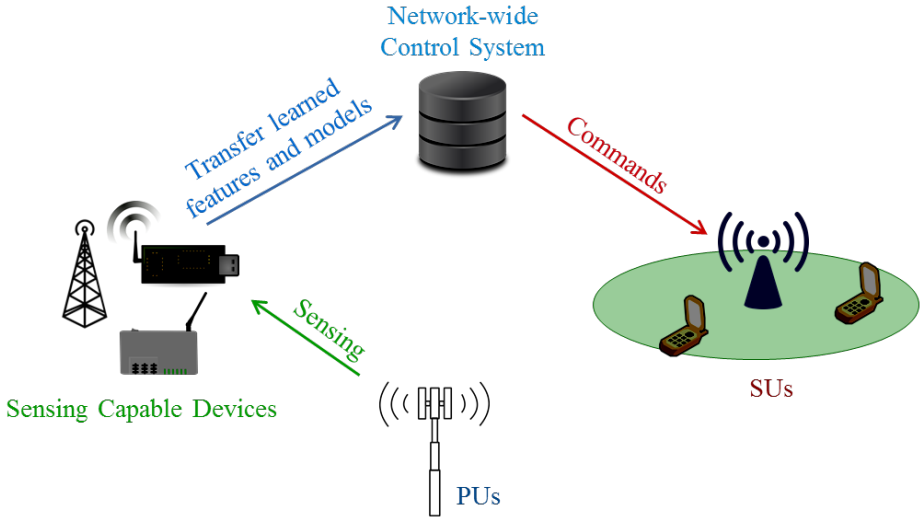


Figure 31 – Transfer of learned features and model across different network elements

For a PU's behaviour to be efficiently reduced to a small set of features or parameters, it has to follow some specific channel access pattern. Some examples of such patterns may include channel hopping sequences, back-off period protocols, and received power variations for the case of radar systems with mechanically rotating antennas. In this work, we focus on the particular scenario of PUs with non-arbitrary channel hopping sequences. In particular, we implemented in SDR several algorithms and functionalities that can support SUs in their decision making and that allow them to reduce interference by avoiding minimizing the probability of accessing the same channels of the frequency agile PU. In our approach, SUs do not need accurate synchronization with the instants of PU's channel transition (t_k). Whenever SUs detect the PU's current channel "i", either through out-

band or in-band sensing, they will hop to the channel with highest availability expected time j provided by a network-wide control system. Thus, overall interference will decrease, but will not be fully avoided.

In this framework, sensing-capable devices **collect** raw wideband sensing samples from the radio environment, derive channel availability, and estimate several PU parameters and a MC channel transition probability matrix based on the PU hopping pattern. The obtained parameters and MC matrix are shared with the network-wide management system, which will **analyse** the adequacy of the Markov-chain assumption, and compute a list of **actions**/commands the nodes in the SU network should follow for each different event. More specifically, when aware of the current PU channel of operation (e.g. when the PU moves to the current SU channel), the SU will consult the list of commands provided by the management system, which will tell the optimal channel to hop to, so that the probability of interference in the future is minimized.

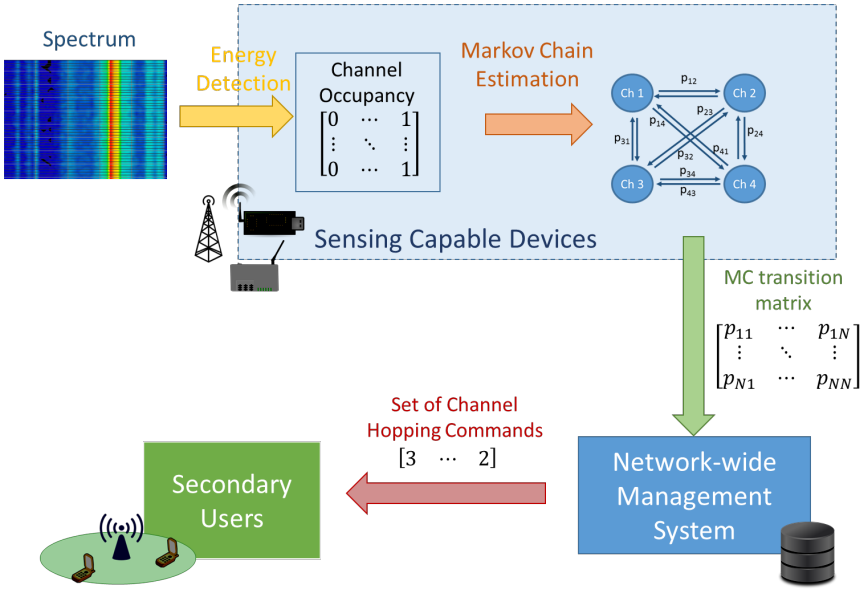


Figure 32 – Markov Chain-based Channel Hopping Framework. The set of channel hopping commands are only derived from the MC transition matrix. Hence, the management system does not need information regarding the current PU's channel in a timely manner.

4.2.2 Markov chain transition matrix estimation

Assume a PU with instantaneous bandwidth of BW that hops across N_{ch} contiguous channels. The decision to hop or stay in the same channel, which we denote as dwell time, happens every T_D seconds. Sensing capable devices, unaware of both the BW, T_D, and hopping transition probabilities, try to estimate these parameters through their collected samples. To achieve these tasks, the following components were designed:

- Energy Detector – The raw IQ samples received by sensing devices are converted to frequency domain, magnitude squared, and grouped into N_{ch} values, each corresponding to the average power of each channel. These values are further smoothed out along the time axis using a moving average, and compared to an adaptive threshold to assess the availability of their respective channels. The energy detector will output every T_{ED} seconds a binary array of size N_{ch}, where channel busy and free correspond to the values 1 and 0, respectively.
- Dwell Time Estimator – Before being able to record the PU channel transitions, the sensing device has to estimate the dwell time of the PU T_D. This procedure can be carried out through observation of the time differences between PU channel transitions. In a noise-less

environment, the period per channel transition will be $\Delta t = mT_D, m \in \mathbb{N}$, where $m > 1$ when the PU stays in the same channel for multiple dwell times. The sensing device can find the fundamental dwell time by finding the highest peak in a histogram of registered time differences Δt.

- Transition Matrix Estimator – The sensor registers the PU activity through a transition matrix M, where each entry $M(i,j)$ corresponds to the number of times the PU transitioned from channel/state i to $j \in \{1, \dots, N_{ch}\}$. The division of $M(i,j)$ by the total number of transitions N_T observed will converge assymptotically to the PU channel transition probabilities p_{ij}. Once N_T surpasses a stop criterion, the sensing device interrupts its transition matrix estimation procedure, and sends the obtained p_{ij}, in combination with other PU-specific parameters such as BW and T_D to the network-wide control entity.

4.2.3 Generation of SUs' channel access commands

From the received transition probability matrix p_{ij}, the intelligence network-wide control system will compute the set of actions/commands an SU should take to minimize the chances of interference to the PU. In this work, we propose that SUs, when they are aware of the current PU channel, move to the channel with the highest expected availability time; that is, given a PU operating in channel $x_0 = i$, and the channel availability time operator $T_{j|i} = \min\{n \geq 1 : x_n = j \mid x_0 = i\}$ for channel j, the SU should move to the channel $j_{\max}|i = \max_{j \neq i} \mathbb{E}\{T_{j|i}\}$. Under our framework, the network-wide control system will compute $j_{\max}|i$ for all possible $i = 1, \dots, N_{ch}$, store them in an array and forward it to the SUs to configure their hopping patterns.

Unfortunately, the value $\mathbb{E}\{T_{j|i}\}$ is not trivial to compute, unless the Markov Chain has some particular properties. We circumvented this issue by relying on approximate solutions $T_{j|i}^{(m)} = \min\{m \geq n \geq 1 : x_n = j \mid x_0 = i\}$, to obtain the solution set $\{j_{\max}|i \; \forall i = 1, \dots, N_{ch}\}$.

4.2.4 General Demonstration Details

Our demonstration setup comprises five radio nodes with different roles, which are illustrated in Figure 33. The used PU-Tx and PU-Rx were ZigBee-like dongle radios that transmit/receive over an instantaneous bandwidth of BW=5 MHz, and hop between N_{ch}=4 separate channels. The SU-Tx transmits OFDM packets to the SU-Rx with an instantaneous bandwidth of 5 MHz, and simultaneously senses the environment to detect whether it is currently causing interference to the PU network. Every time the PU-Tx hops to its current channel, the SU-Tx needs to move to a different channel, whose index $j_{\max}|i$ is derived from the Markov Chain according to Section 4.2.3. Finally, the sensor is responsible for estimating the PU's parameters, namely BW and dwell time, and computing the Markov Chain of the PU's channel transitions. These parameters can then be exposed to other SU network elements through, for instance, the WiSHFUL UPI.

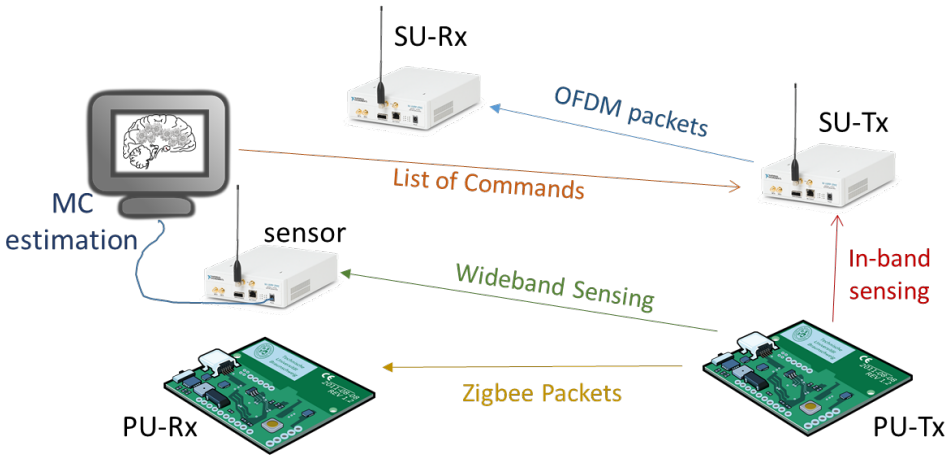


Figure 33 – Markov Chain-based Hopping Demonstration Setup

We developed the several functionalities of the SU-Tx, SU-Rx, and sensor in C++, so they can be easily integrated in both Iris and GNU Radio frameworks, both of which are supported by WiSHFUL UPIs. In the repository made available, all of these network elements can be run through the simple "markov_radio" script. To select which of the network elements to run (SU-Tx, SU-Rx or sensor), the user of the demo can set the flags "has_learning" and "tx_opt" over the command line. When the flag "has_learning" is set on, the SDR node will estimate the PU's parameters and Markov channel transition matrix. If the "tx_opt" flag is on, the SDR node will act as SU-Tx and will start the transmission of packets over the air. The channel hopping strategy of the SU-Tx is undefined until the channel transition commands ($j_{\max}|i$) derived from the Markov Chain are available to it. Finally, in case of no "has_learning" and no "tx_opt", the SDR node will behave as a SU-Rx and waits for incoming packets from the SU-Tx.

To provide the sufficient separation and modularity of the SU-Tx, SU-Rx, and sensor functionalities, the implemented algorithms and techniques, namely the energy detector, dwell time estimator, and MC generator were separated into independent C++ classes that the experimenter can use independently.

4.2.5 Next Steps

As a next step, we plan to further integrate the described demo with the WiSHFUL framework and UPIs. In particular, we intend to provide the functionality of configuring the sensor, SU-Tx, and SU-Rx's bandwidth and centre frequencies remotely, using the already made available Iris/GNU Radio-WiSHFUL controller interface. Additionally, we plan to extend the WiSHFUL UPI to support configuration file exchange between Iris/GNU Radio nodes. This functionality is required so that the exchange of the Markov Chain and list of action/commands can take place via the WiSHFUL UPI.

We also plan to improve the current intelligence capabilities of the sensor node and SU-Tx. In its current state, our implementation does not support PUs that alter their patterns/MCs over time. To overcome this limitation, we will increase the number of features the sensor and SU-Tx can estimate from the radio environment, and implement a hidden Markov model-based inference algorithm at the SU-Tx that can detect the current pattern used by the PU-Tx. Finally, we plan to improve the sensor's capabilities to discriminate between SU-Tx and PU-Tx transmissions. This can be achieved either through matched filter, feature detection algorithms, or more advanced classification/learning techniques..

5 Conclusions

The design and the implementation of the intelligence showcases, considered during Y2, allowed to **validate the effectiveness of the approaches and tools** envisioned for supporting intelligence in WiSHFUL, and to **populate the WiSHFUL intelligence repository** with an initial set of components, available to other experimenters for facilitating the definition of new intelligent control programs.

We consider two main contributions for other experimenters, in terms of **components for data collections and aggregation**, which allow to collect basic network measurements from independent nodes and build global network views, and in terms of **components for estimating the consequence of adaptation actions** on performance metrics. In turns, this second group of components include data-driven models and on-line learning engines, that have been designed and used for solving specific network problems, but can be potentially re-used in completely different contexts. In particular, we developed:

- a **WSN performance model**: this data-driven model is devised to estimate performance of WSNs under different MAC protocols, as a function of node density and interference conditions;
- a **WiFi performance model for multi-cell scenarios**: this data-driven model is devised to classify interference conditions experienced in different cells as hard, soft or zero interference and identify blocked cells;
- a **Meta-MAC engine**, for dynamically learning about the best medium access decision among a set of elementary protocol components, in WiFi fully-connected networks under varying traffic conditions;
- a markov-chain based **decision engine**, for inferring about the primary user channel with will remain available for longer time intervals in cognitive networks, in case of predictable behaviours.

These modules and the exemplary composition of these modules considered in Y2 showcases provide an initial entry in the catalogue of the intelligence utilities of the WiSHFUL framework.

6 References

[1] Peng W. et al. Cogmac+: A decentralized mac protocol for opportunistic spectrum access in cognitive wireless networks. Elsevier Journal of Computer Communications, 2015.

[2] Jerry Z. et al. Understanding packet delivery performance in dense wireless sensor networks. In Proceedings of the 1st international conference on Embedded networked sensor systems, pages 1–13. ACM, 2003.

[3] Kulin, Merima, et al. "Towards a cognitive MAC layer: Predicting the MAC-level performance in Dynamic WSN using Machine learning", (2016) arXiv:1612.03932

[4] Kulin, Merima, et al. "Data-Driven Design of Intelligent Wireless Networks: An Overview and Tutorial." Sensors 16.6 (2016): 790.

[5] Bart Jooris, Jan Bauwens, Peter Ruckebusch, Peter De Valck, Christophe Van Praet, Ingrid Moerman, Eli De Poorter, "TAISC: a cross-platform MAC protocol compiler and execution engine", Computer Networks, Volume 107, Part 2, 9 October 2016

[6] Peter R. et al. A unified radio control architecture for prototyping adaptive wireless protocols. In Networks and Communications (EuCNC), 2016 European Conference on, pages 58–63. IEEE, 2016.

[7] Eibe Frank, Mark A. Hall, and Ian H. Witten (2016). The WEKA Workbench. Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques", Morgan Kaufmann, Fourth Edition, 2016.

[8] http://www.cs.waikato.ac.nz/ml/weka/

[9] http://scikit-learn.org/stable/

[10] E. Khorov, A. Kiryanov, A. Krotov, P. Gallo, D. Garlisi, I. Tinnirello, “Joint Usage of Dynamic Sensitivity Control and Time Division Multiple Access in Dense 802.11ax Networks”, Multiple Access Communications MACOM 2016

[11] Caruana, R. and A. Niculescu-Mizil (2006). An empirical comparison of supervised learning algorithms. In Proceedings of the 23rd international conference on Machine learning, pp. 161–168. ACM.

[12] Faragó, A. D. Myers, V. R. Syrotiuk, and G. Záruba, “Meta-MAC protocols: Automatic combination of MAC protocols to optimize performance for unknown conditions,” IEEE Journal on Selected Areas in Communications, vol. 18, no. 9, pp. 1670–1681, September 2000.

[13] Meta-MAC - https://github.com/wishful-project/examples/tree/master/wmp/wmp_metamac