DIWINE

Contract No. CNET-ICT-318177

SLS platform and algorithm transfer methodology
D5.31

Contractual date:	M12
Actual date:	M12
Authors:	Kostas RAMANTAS, David BOIXADE, Jan SÝKORA, Tomáš HYNEK, Pavel PROCHÁZKA, Umberto SPAGNOLINI, Maria Antonieta ALVAREZ VILLANUEVA, David HALLS, Eduard A. JORSWIECK, Alister G. BURR, Roger TORNE, Cheng CHEN, Justin P. COON
Participants:	IQU, CTU, PdM, TREL, TUD, UoY
Work package:	WP5
Security:	Public
Nature:	Report
Version:	1.0
Number of pages:	35

Abstract

This is a report on the methodology for the transfer of proposed algorithms and scenarios to be converted into the system level simulation and hardware demonstration platforms. The report covers the algorithm transfer methodology and interfaces, as well as the system level simulation platform.

Keywords

Algorithm transfer, software-hardware interfaces, system level simulation.

Executive summary

This report details the methodology to transfer key DIWINE algorithms from concept to the software (SW) and hardware (HW) platforms. Firstly, a high-level description of a subset of algorithms detailed in WP3 and WP4, which were deemed most suitable for demonstrator implementation, is provided. Then, the strategy to be followed within WP5 to ensure a smooth transfer from MATLAB functions to C++ modules for both the system level simulator (SLS) and the HW demonstrator is detailed. Finally, the SLS is presented and the new DIWINE-specific modules which were implemented as part of the DIWINE system level simulation platform are detailed.

The algorithm transfer to the HW demonstrator consists of a three-stage process. At each stage, the algorithm becomes more embedded and autonomous within the HW demonstrator. The first stage involves exchanging samples from WP3 and WP4 to WP5 for off-line processing. The data will be transferred in the frequency domain (FD) at the constellation level. A backward transfer from WP5 to WP3 and WP4 will also take place. A proof of concept has already been shown using CTU's cloud initialisation procedure (CIP). The next stage is where small fragments of MATLAB code can be transferred between partners, which will allow the offline processing of results captured using GNU Radio. Once this has been achieved, the MATLAB code can be compiled as a shared library and run within a C++ block in GNU Radio. Proof of concept has also already been achieved in the CIP case where the MATLAB code has been compiled and embedded in a new GNU Radio Companion (GRC) block. The final core stage is to replace the call to compiled MATLAB code with C++ code, which is native to the GNU Radio software and universal software radio peripheral (USRP) hardware.

The transfer of DIWINE algorithms to the SLS can be achieved with a three-step process, which essentially converts MATLAB functions to C++ classes. The first step comprises the transfer of channel and error models to the SLS, which can be jointly defined by partners and incorporated into the SLS. A cross-layer API has already been implemented, which allows MATLAB vectors with the channel realisations to be loaded to the simulator in the form of MATLAB (*.mat) files. The same approach can be followed for the packet error function. The next step is the implementation of new packet-level statistics, defined by the academic partners for the evaluation of proposed algorithms. Finally, being a flexible tool, the SLS can be easily extended with the new DIWINE algorithms and protocols, which can be implemented in the form of finite state machines (FSM). A proof of concept of this procedure, transferring the interference neutralisation algorithm from TUD to the SLS, has already been implemented.

The role of the SLS in the DIWINE project is to serve as a platform for evaluating advanced algorithms in complex topologies, which would be unrealistic to implement in the hardware demonstrator. It also allows benchmarking and verification of algorithms before submitting to the hardware platform for implementation. In this report all DIWINE-specific modifications are detailed, as well as the set of new modules that had to be implemented.

Contents

Abbreviations

2D	2-Dimensional
2S2R2D	Two-Source Two-Relay Two-Destination
BER	Bit Error Rate
BPSK	Binary Phase Shift Keying
CFO	Carrier Frequency Offset
CIP	Cloud Initialisation Procedure
CSMA	Carrier Sense Multiple Access
CZ	CAZAC – Constant Amplitude Zero Autocorrelation (sequence)
FD	Frequency Domain
FIFO	First-In First-Out
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
GRC	GNU Radio Companion
GUI	Graphical User Interface
HDF	Hierarchical Decode-and-Forward
HI	Hierarchical Information
HNC	Hierarchical Network Code
H-SI	Hierarchical-Side Information
HW	Hardware
ID	Identifier
IN	Interference Neutralisation
JDF	Joint Decode-and-Forward
LDPC	Low Density Parity Check
MAC	Medium Access Control
MIMO	Multiple-Input Multiple-Output
MISO	Multiple-Input Single-Output
OFDM	Orthogonal Frequency Division Multiplexing
OOK	On-Off Keying
OSI	Open Systems Interconnection (model)
PEP	Packet Error Probability

PHY	Physical Layer
PiHrc	Hierarchical Pilot
QoS	Quality of Service
QPSK	Quadrature Phase Shift Keying
SINR	Signal-to-Interference-plus-Noise Ratio
SLS	System Level Simulator
SMN	Smart Metering Network
SNR	Signal-to-Noise Ratio
SW	Software
SWIG	Simplified Wrapper and Interface Generator
TD	Time Domain
USRP	Universal Software Radio Peripheral
UT	Utility Target
WNC	Wireless Network Coding
WP	Work Package
XOR	eXclusive OR

2 Algorithm transfer methodology

2.1 Introduction

In order to ensure smooth transfer of the algorithms from the WP3/WP4 partners to the WP5 SLS and HW platforms is it necessary to agree, and adhere to, a predefined methodology. This section details firstly the proposed algorithms that will be transferred to the SLS and HW demonstrators, representing a subset of those detailed in WP3 and WP4 which are deemed most suitable for demonstrator implementation – possibly affected by necessary simplifications. A subset of these will be implemented in the smart metering network (SMN), with a focus on wireless network coding (WNC), due the risks associated with implementing new theoretical techniques and time constraints. A high-level description and some initial evaluation of these algorithms are provided. The proposed algorithm transfer interfaces for the SMN demonstrator platform is then outlined, followed by those for the SLS. The algorithms addressed in this report are:

- Block-structured layered design of network coded modulation (CTU);
- Distributed learning process for hierarchical network code (HNC) selection (CTU);
- Interference neutralisation on interference relay channel (TUD);
- Network synchronisation (PdM).

2.2 High-level description and evaluation of proposed algorithms

2.2.1 Block-structured layered design of network coded modulation

This algorithm, based on the block structured layered design, allows the conventional capacity achieving technique, low density parity check (LDPC) codes, to robustly take advantage of the wireless network coding paradigm, i.e. it enables wireless extension of the network coding paradigm. It describes how to properly design particular channel encoders and corresponding HNC matrices within the network such that the relay consistently decodes a proper hierarchical stream (a desired function of the source data streams). The main advantage of the considered algorithm (see [1] for full description) is its robustness, allowing the incorporation of the hierarchical stream with arbitrary information content [2]. Moreover, the channel code rates in sources can be adjusted arbitrarily.

It should be stressed that the relay is supposed to receive a (signal-space) noisy super-position of the signals from sources (no orthogonal resource slicing is allowed). The conventional solution used in this situation is joint decoding of the particular streams from the superimposed constellation. When error-less joint decoding in the relay is insisted upon, the limit is the second order cut-set bound even if the decoding is followed by conventional network coding applied on the jointly decoded individual data streams. The basic idea enabling performance potentially beyond the higher order cut-set bound is direct decoding of the hierarchical stream from the super-imposed constellation, which avoids a joint decision about the individual data streams. Instead, a soft metric is inserted into a block deciding upon the hierarchical data stream containing less information than the joint data streams. The current state-of-the-art recognises mainly eXclusive OR (XOR) as the hierarchical function fixing an equal information content from the sources. Figure 1 shows the superimposed modulations (the simplest example, using binary phase shift keying (BPSK) modulation) and two possible techniques at the relay:

1) Joint decode-and-forward (JDF) that provides a decision about the pair b_A, b_B (reliable decision making is restricted by the classical multiple access channel region);
2) Hierarchical decode-and-forward (HDF) provides a decision about a function, e.g. bitwise XOR, of b_A, b_B, i.e. $b_{AB}=f(b_A, b_B)$.

Since b_{AB} contains less information than the pair b_A, b_B, one can reliably operate on rates above the classical multiple access channel region. Such rates, i.e. potential HDF gain, are emphasised by the red shaded area.

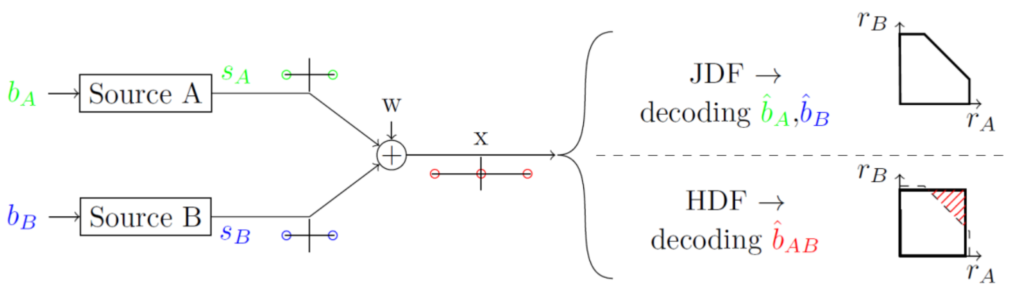


Figure 1: JDF vs. HDF strategy bounds in relay

The hardware verification of the algorithm will be performed in a simple butterfly network (as shown in Figure 2), where the information stream through the relay must contain at least a mixture of both streams, e.g. the XOR function, since the direct links are not considered and only one possible path from both sources to the desired particular destinations is through the relay. The degree of freedom consists of the quality of Hierarchical Side Information (H-SI) links. The H-SI is shown by the dashed green links in Figure 2, and they must deliver a specified complement of the hierarchical stream to the particular destination.

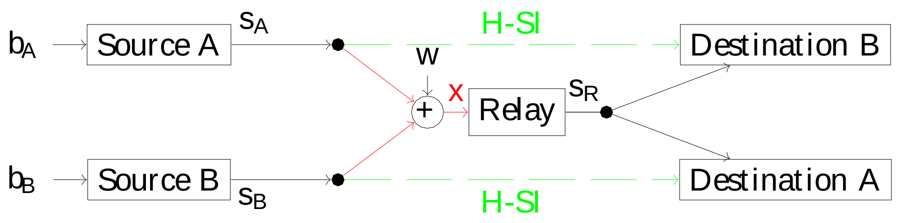


Figure 2: Information flows in butterfly network

If these links are capable of carrying full information from the sources, then it is sufficient for the minimal mapping, e.g. XOR hierarchical function, to be applied in the relay. On the other hand, if there are no H-SI links, the relay must perform JDF and the system is multiple access channel limited. In this project the case in between is investigated, i.e. the H-SI is neither empty nor full, by means of the block structured layered design. A smooth overall bit error rate (BER) at the destinations is expected, depending on the quality of the side links, with a given fixed mean energy per symbol in all nodes.

2.2.2 Distributed learning process for HNC selection

This algorithm equips each cloud internal node (relay) with an internal intelligence and ability to self-adapt its private HNC. Although the node HNC adaptation is done in a selfish way – each node wants to maximise its own utility (or equivalently minimise its costs) – the global goal of the cloud has to be fulfilled, the cloud has to provide the reliable source-destination communication. The algorithm concerns distributed operation and the internal intelligence of the cloud rather than signal processing of particular transmissions.

Since the selfish selection of the HNCs at cloud nodes may lead to a non-invertible observation at the final destinations, a conflict of interest among the relay nodes arises. In accordance with the DIWINE concept this conflict of interest has to be resolved without any intervention by a central authority – it rather has to be resolved in a distributed way.

The proposed distributed learning process is a game theory based algorithm. It is described in depth in [3]. Each cloud node, being informed of the HNCs of the other relays is given a chance to adapt its own HNC in order to improve its (local) utility. The only thing that it has to respect is the invertibility of global HNCs at each individual destination. The local utility/cost function may be an arbitrary function. Although one starts with functions that are simple to evaluate such as relay output cardinality (which is related to energy efficiency of the node) or minimal hierarchical distance (which correspond to error rate performance) any of the utility targets (UT) proposed in [4] can be used.

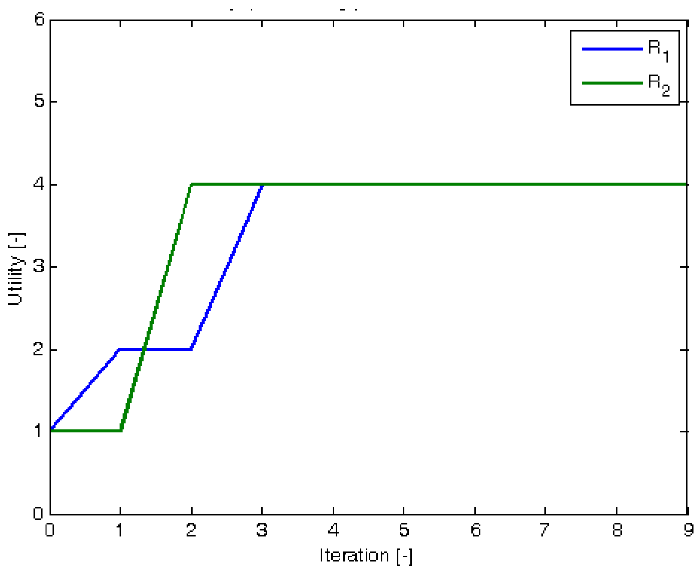


Figure 3: Myopic learning process– 2S2R2D

An analysis in [5] shows that the learning process (based on relay output cardinality) leads to promising and reasonable (in game theory sense) results in the Nash equilibria of the game, that are all efficient. It also shows that the process converges in several iterations for two-source two-relay two-destination (2S2R2D) cloud network. An example of utility progress during the learning process is depicted in Figure 3. Figure 4 shows an example of minimal hierarchical distance utility function as a function of channel parameterisation.

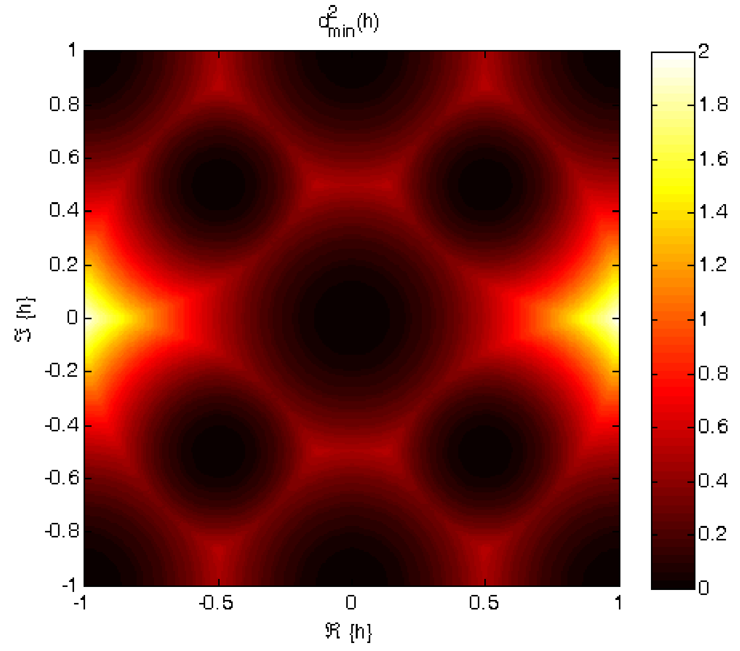


Figure 4: Minimal hierarchical distance utility function as a function of channel parameterisation

To be able to infer the information about the opponent HNCs (in game theory terminology their ‘actions’), one must receive some kind of feedback information. This feedback can be provided by hierarchical pilots (PiHrc) included in the backward frame of the DIWINE air interface (see [4] for detailed information).

Currently the algorithm is implemented through software simulation using MATLAB. To implement it, the algorithm is further decomposed into several functions that solve particular problems. Each of these functions is a MATLAB m-file that will be transferred to industrial partners for practical implementation through a process described in the next sections. Particularly, the following subtasks exist:

- A general simulation framework, which will be replaced by a real world scenario;
- A function to generate PiHrcs;
- A function to infer the information from backward PiHrcs;
- A function to check the invertibility of local HNC with opponents’ HNCs;
- A function to compute a local utility/cost function;
- A function to select a new local HNC;
- Support functions to compute ranks of matrices, to update PiHrc, etc.

Since the algorithm serves to improve the performance of WNC it assumes that the WNC in the cloud is already running (although sub-optimally with high probability). Thus, it assumes that wireless connections among the nodes are established – nodes are time and frequency synchronised and each node has the information about the beginnings of frames and superframes and knows its transmission and reception time slots. These tasks are necessary to begin any WNC communication and thus have to be fulfilled prior to the commencement of the WNC part of communication.

It is a matter of correct HW design and implementation as well as implementation of preceding algorithms (for distributed synchronisation, initial WNC, etc.) to achieve those conditions.

2.2.3 Interference neutralisation on interference relay channel

This section describes a simple example for the demonstration of interference neutralisation (IN). A simple cloud network with two source nodes, two destination nodes and three relay nodes is assumed, as shown in Figure 5. Coordination and cooperation among the two relay nodes takes place within the wireless cloud, whereas the source and destination nodes enjoy a plug-and-play service.

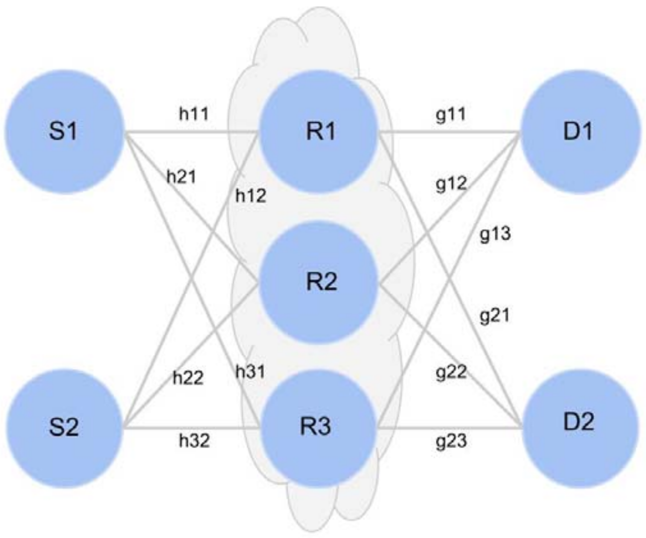


Figure 5: The system model of the cloud network

Every node in the system is assumed to have a single antenna. The relay nodes are assumed to operate in a half-duplex mode. Denote the data symbols sent by the transmitters S_1, S_2 by x_1, x_2 respectively. The symbols x_i, i = 1,2, are chosen from a predefined codebook and are statistically independent with a predefined power P_i. The received signals at the relay nodes R_1, R_2 are given by:

$$\begin{aligned} y_{r1} &= h_{11}x_1 + h_{12}x_2 + n_{r1} \\ y_{r2} &= h_{21}x_1 + h_{22}x_2 + n_{r2} \end{aligned} \qquad (1)$$

The background noise at the relay nodes n_{r1}, n_{r2} and at the destination nodes in the following are assumed to be complex white Gaussian noise with zero mean and unit variance. The relays are assumed to employ an amplify-and-forward strategy and multiply the received signal with a complex scalar α_1, α_2 and α_3 respectively. The received signal at the destination nodes D_1, D_2 are given by:

$$\begin{aligned} y_{d1} &= g_{11}\alpha_1 y_{r1} + g_{12}\alpha_2 y_{r2} + g_{13}\alpha_3 y_{r3} + n_{d1} \\ y_{d2} &= g_{21}\alpha_1 y_{r1} + g_{22}\alpha_2 y_{r2} + g_{23}\alpha_3 y_{r3} + n_{d2} \end{aligned} \tag{2}$$

Plugging in (1) to (2), we can derive the received signal at D_1, D_2, i.e. y_{d1}, y_{d2}, as a function of x_i, g_i, and h_i. We observe that each message reaches each destination node in three paths. By choosing the relay strategies intelligently, the three paths of the interference messages can algebraically add up to zero. In particular, the following two *interference neutralisation constraints* have to be satisfied simultaneously:

$$\begin{cases} g_{11}\alpha_1 h_{12} + g_{12}\alpha_2 h_{22} + g_{13}\alpha_3 h_{32} = 0 \\ g_{21}\alpha_1 h_{11} + g_{22}\alpha_2 h_{21} + g_{23}\alpha_3 h_{31} = 0. \end{cases} \tag{4}$$

To facilitate the problem formulation, the following matrices are defined. Denote the equivalent channel (through three relays) from S_i to D_j to be:

$$f_{ji} = [g_{j1}h_{1i}, g_{j2}h_{2i}, g_{j3}h_{3i}]^T. \tag{5}$$

The relay amplification factors in a column vector are written as:

$$\alpha = [\alpha_1, \alpha_2, \alpha_3]^T. \tag{6}$$

Let $F = \left[f_{12}^{\mathrm{T}}; f_{21}^{\mathrm{T}} \right]$. Any $\boldsymbol{\alpha}$ that satisfies the IN constraints must be in the null space of **F** as the interference neutralisation constraints (4) can be written as:

$$F \cdot \alpha = 0_{2\times 1} \tag{7}$$

Note that the null space of **F**, which is a 2-by-3 matrix, is only one dimension. Hence, the null space vector of **F** can be denoted by **v** and $\boldsymbol{\alpha}$ can be written as:

$$\alpha = c \cdot v \tag{8}$$

for some complex constant c. The channel from the relays to D_j is defined to be:

$$D_j = diag\left(|g_{j1}|^2, |g_{j2}|^2, |g_{j3}|^2 \right) \tag{9}$$

The achievable signal to noise ratio of x_i, $i = 1,2$, is given by:

$$\gamma_i = \frac{|f_{ii}^{\mathrm{T}}\alpha|^2 P_i}{\alpha^T D_i \alpha + 1} \tag{10}$$

The transmit power of relay *k, for* $w_k = |h_{k1}|^2 P_1 + |h_{k2}|^2 P_2 + 1$, is computed as:

$$p_{rk} = |\alpha_k y_{rk}|^2 = |\alpha_k|^2 \left(|h_{k1}|^2 P_1 + |h_{k2}|^2 P_2 + 1\right) = w_k |\alpha_k|^2, \tag{11}$$

Given a pair of quality of service (QoS) constraints γ_1, γ_2, the achievable γ_i must be larger than or equal to the QoS. It is possible to formulate the selection of relay forwarding factors $\boldsymbol{\alpha}$ that minimise the relay transmit power, while satisfying the QoS and interference neutralisation constraints, as an optimisation problem [3], where the optimal solution is given by:

$$c = \min\left\{\frac{\overline{\gamma}_1}{|f_{11}^{\mathrm{T}}v|^2 P_1 - \overline{\gamma}_1 v^{\mathrm{H}} D_1 v}, \frac{\overline{\gamma}_2}{|f_{22}^{\mathrm{T}}v|^2 P_2 - \overline{\gamma}_2 v^{\mathrm{H}} D_2 v}\right\}, \tag{12}$$

In the following subsections, a set of interference neutralisation algorithms are demonstrated and compared in terms of their performance.

2.2.3.1 Suboptimal interference neutralisation

Any relay vector $\boldsymbol{\alpha}$ satisfying (8) completely neutralises interference at D_1 and D_2. A simple algorithm is to construct different $\boldsymbol{\alpha}$ by plugging in simple solution such as $c = 1$. However, this does not minimise the relay transmit power nor guarantee the QoS. The algorithm is described in pseudo-code below.

Algorithm 1: Suboptimal interference neutralisation

```
1. Input: Channel realisations, source power $P_1, P_2$.
2. Output: relay transmit power $p_{r1}, p_{r2}, p_{r3}$, achievable SINR values $\gamma_1, \gamma_2$.
3. Transmitters $S_1, S_2$ send signals $x_1, x_2$.
4. At the first hop, relay nodes receive and amplify the signal with amplification gains $\alpha = v$ in
   (8). Transmit power of relay $k$ is given by $p_{rk} = w_k |v|^2$ in (11).
5. At the second hop, there is no interference. Achievable SINR at $D_i$ is $\gamma_i = \frac{|f_{ii}^{\mathrm{T}} v|^2 P_i}{v^{\mathrm{H}} D_i v + 1}$
```

2.2.3.2 Interference neutralisation with QoS

For this algorithm, the optimal relay vector $\boldsymbol{\alpha}$ is computed such that QoS is guaranteed, interference is neutralised and the relay power is at the same time minimised. The closed-form solution is given by (12). The algorithm is described in pseudo-code below.

Algorithm 2: Interference neutralisation with QoS

1. Input: Channel realisations, SINR constraints $\bar{\gamma}_1, \bar{\gamma}_2$, source power P_1, P_2.
2. Output: relay transmit power p_{r1}, p_{r2}, p_{r3}, achievable SINR γ_1, γ_2.
3. Transmitters S_1, S_2 send signals x_1, x_2.
4. At the first hop, relay nodes receive and amplify the signal with amplification gains $\alpha = c \cdot v$ in (8) and c is given by (12). Transmit power of relay node k is given by (11), $p_{rk} = w_k |\alpha_k|^2$, $k = 1, 2, 3$.
5. At the second hop, there is no interference. Achievable SINR at D_i is equal to the given QoS constraints $\gamma_i = \bar{\gamma}_i$.

2.2.3.3 Layer-1 relay system

This is a baseline algorithm in which the relay vectors α are chosen such that the relay transmit power constraint is satisfied, mimicking a wireless network connected by layer-1 relays (amplifiers).

Algorithm 3: Layer-1 relay

1. Input: Channel realisations, relay node power constraints P_{r1}, P_{r2}, P_{r3}, source power P_1, P_2.
2. Output: achievable SINR γ_1, γ_2.
3. Transmitters S_1, S_2 send signals x_1, x_2.
4. At the first hop, relay nodes receive and amplify the signal with full power. Amplification gains is given by $\alpha_i^2 = \dfrac{P_{ri}}{|h_{i1}|^2 P_1 + |h_{i2}|^2 P_2 + 1}$, where α_i a real number.
5. At the second hop, there is interference. Achievable SINR at D_i is equal to the given QoS constraints $\gamma_i = \dfrac{|f_{ii}^{\mathrm{T}} \alpha|^2 P_i}{|f_{ij}^{\mathrm{T}} \alpha|^2 P_j + \alpha^{\mathrm{T}} D_i \alpha + 1}$, where $i, j = 1, 2, j \neq i$.

2.2.4 Network synchronisation

In this section a description of the DIWINE network synchronisation algorithm is provided [7]. Each DIWINE superframe consists of a synchronisation frame and a payload (data symbols) according to the duplexing scheduling. The synchronisation frame (Figure 6) is a set of orthogonal cyclic sequences, each with a length of 64 samples or duration of *T*, as agreed due to hardware constraints. These are referred as CAZAC (CZ) sequences. Each node generates frames according to its own timing depending of some phase and frequency misaligned error.

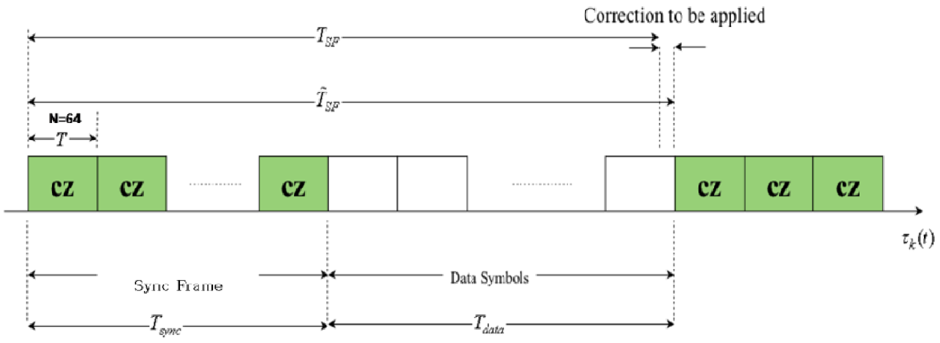


Figure 6: CZ sequence in frequency frame

For the algorithm design, each node generates a unique CZ sequence that is modulated with a local oscillator that has frequency error. The received signal at an individual node is the sum of the transmitted signals by all the others nodes of the network. Each node alternates receive or transmit states. When transmitting, the node distributes its frequency and time to all the other nodes to let them synchronise; when the node is in the receive state, it corrects its local frequency and time to synchronise to the others. The receiver node first applies the demodulation to the received signal – this signal is the sum of all the transmitted signals from the other nodes after the multipath channel ($\boldsymbol{h}_{ik}(t)$) – then this is correlated with each CZ sequence (Figure 7) The output from the bank of correlators consists of large amplitudes (peaks) around the delay of the specific CZ sequence with respect to the timing of the receiver. Peaks repeat periodically depending on the number of CZ sequences that are repeated within the synchronisation frame (the number of the CZ sequences is variable, which depends on the state of the network). The real and imaginary parts from multiple peaks are used to estimate the timing drifts and frequency errors.

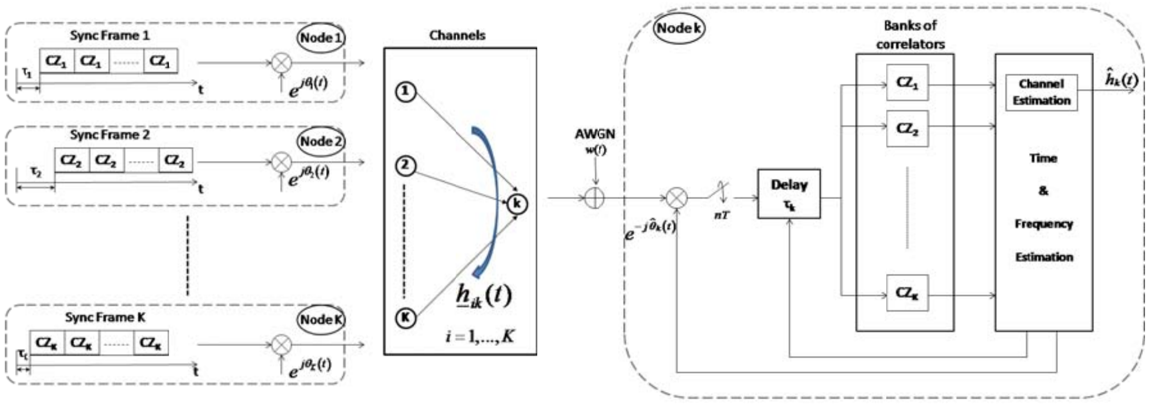


Figure 7: Synchronisation algorithm scheme. $\boldsymbol{h}_{ik}(t)=\left[h_{1k}(t),h_{2k}(t),\ldots,h_{Nk}(t)\right]^T$ is the MISO vector according to the Kx1 channels from all the K nodes to the k^{th} receiving node (here it is accounted the loopback k→k)

For clock synchronisation, it is assumed that the period of both sync frame and superframe is approximately constant. Thus, the simplified node k's clock model is adopted as:

$$\tau_k(t) = t + \beta_k \qquad k = 1, \ldots, K, \tag{13}$$

where the timing offset β_k depends on the mutual misalignments. In Figure 8, the outputs of the bank of correlators in a receiver node can be seen, which are correlated with the CZ sequences of a set of four transmitter nodes. Figure 8 illustrates the misalignments of nodes 1, 3, 4 with respect to node 2 which is used as the reference. The synchronisation frame signal is affected by the channel due to the length of the CZ being 64. Compensation requires the use of the periodicity property of CZ sequences to recover the starting point of the synchronisation frame.

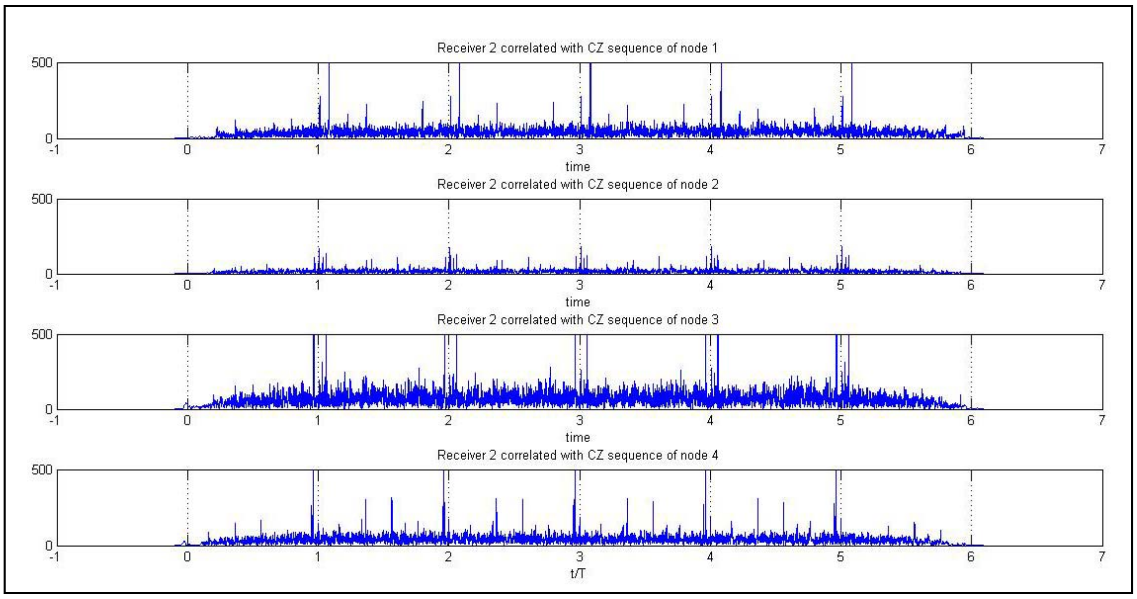


Figure 8: CZ sequence at receiver node 2 from a set of K=4 nodes, 1→2, 2→2 (loopback), 3→2, 4→2

The instantaneous phase of the carrier of k^{th} node is:

$$\theta_k(t) = \varphi_k + 2\pi \times CFO_k(t) \qquad k = 1, \ldots, N, \tag{14}$$

which is going to affect the CZ sequences in the modulating and demodulating procedure. Figure 9 shows the estimations of the CFO at node 2. The slope is equal to the difference between the transmitter and receiver nodes ($\theta_i - \theta_k \quad i = 1, \ldots, K$). The synchronisation algorithm is an iterative correction method that corrects locally the CFO and timing based on the consensus of measurements from all transmitting nodes using the following correction rules:

$$CFO_k(t+1) = CFO_k(t) + \varepsilon_{CFO} \times CFO_{error}(t), \tag{15}$$

$$\tau_k(t+1) = \tau_k(t) + \varepsilon_{timing} \times \Delta\tau_k(t), \tag{16}$$

where ε_{CFO} and ε_{timing} are properly chosen scaling terms to guarantee the global convergence.

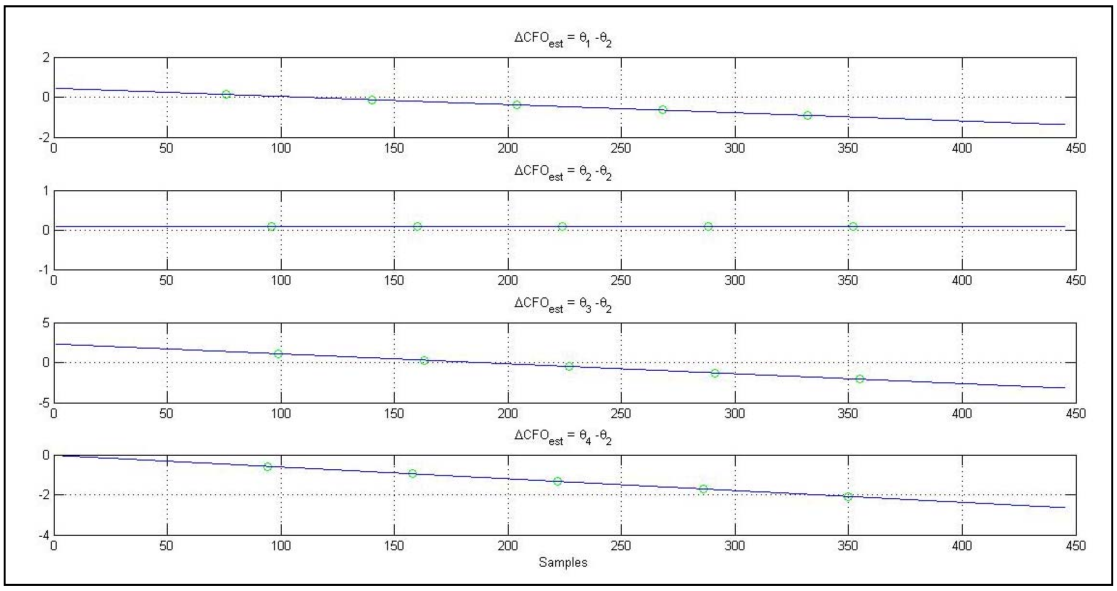


Figure 9: Frequency estimation at node 2 from the set of K=4 nodes (same settings as in Figure 8)

2.3 Algorithm transfer interfaces to the HW platform

The algorithm transfer will comprise a three-stage process. At each stage, the algorithm becomes more embedded and autonomous within the HW demonstrator. This will allow for a robust, iterative approach. At each of the three stages of the algorithm exchange process, the initial debugging will take place without using a USRP in the testing. This is achieved by storing the transmitted waveform in a file (rather than transmitting over the USRP), and then loading the received waveform from the same file (rather than receiving from the USRP). This additional debugging stage will ensure that all of the baseband digital signal processing is functioning as expected. Passing the time domain (TD) waveforms via a file removes all of the imperfections added by the USRP, as well as the radio channel. Once this is working correctly, it will then be possible to add, in software, a known channel response (with given idealised properties) and receiver noise, as a transition towards the full USRP testing scenario.

2.3.1 Stage 1: Exchange of samples for off-line processing

In the first stage, for the forward transfer from WP3 and WP4 to WP5, the data will be transferred in the frequency domain (FD) at the constellation level. This will be for each individual orthogonal frequency division multiplexing (OFDM) subcarrier including all of the constituent symbols that make up the preamble, pilots, header and payload.

The backward transfer from WP5 to WP3 and WP4 will take place in two different ways, depending upon the circumstances. Where full external synchronisation of the hardware is available and ensured, the transfer will be made in the FD at the constellation level for each subcarrier, in the same manner as the forward transfer. Where this cannot be ensured, transfer will take place at the sample level, i.e. time domain samples of the full OFDM signal.

Proof of concept of this has already been shown using CTU's CIP. This algorithm requires only random on-off keying (OOK) constellations at the FD constellation level input, so these were able to be generated by the WP5 partner. a multiple-input single-output (MISO) OFDM transmission was then performed using two transmitters and one receiver, firstly transmitting through a file, and secondly via USRPs connected using a three-way power-divider. The resulting pre-equalised superimposed FD constellations were then input to CTU's CIP MATLAB script. This script takes constellation points as inputs and outputs the estimated number of sources (transmitters), and the estimate of the best centroids. The results in Figure 10 and Figure 11 show the correctly estimated best centroids results from the CIP algorithm in red for two runs, where there are two sources in the MISO system.

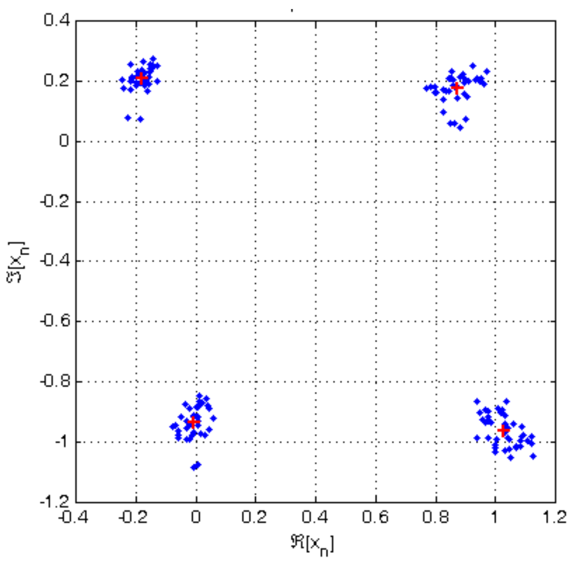


Figure 10: Cloud initialisation procedure with OFDM data, first run

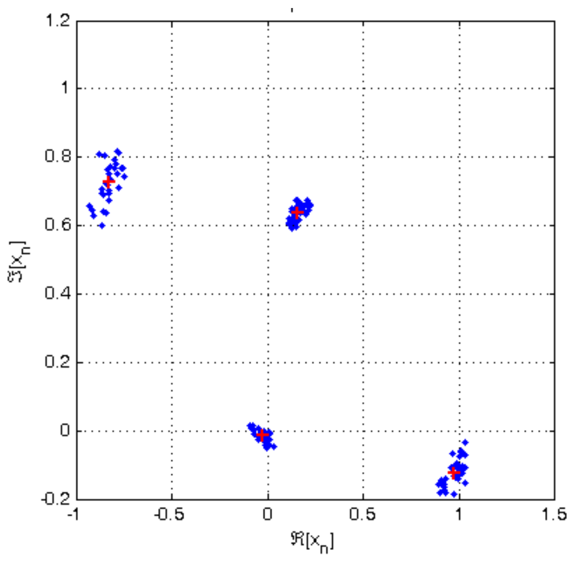
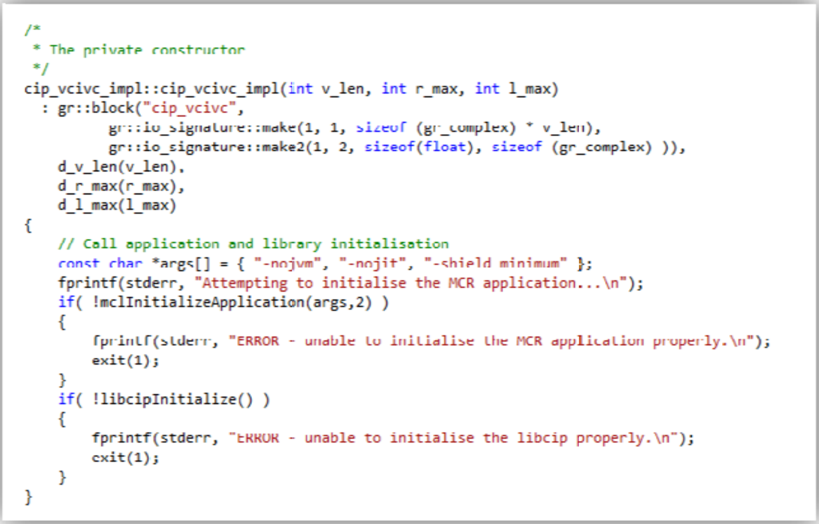


Figure 11: Cloud initialisation procedure with OFDM data, second run

2.3.2 Stage 2: Exchange of MATLAB processing fragments

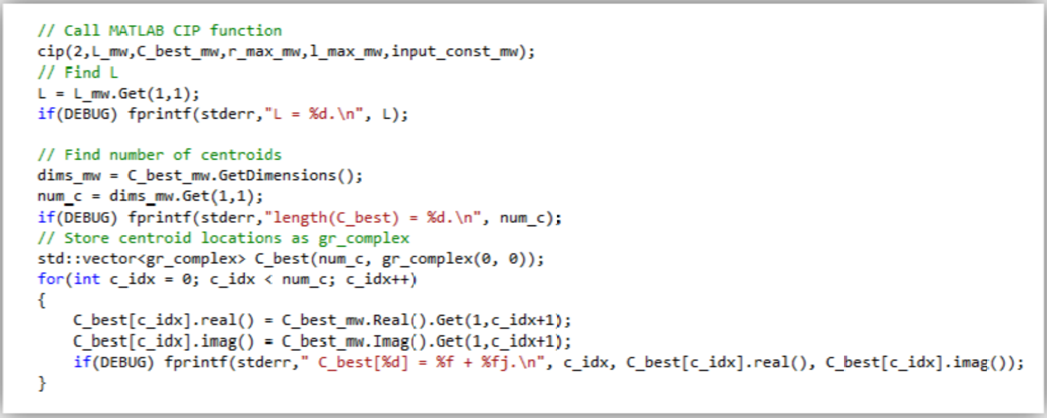
The next stage is where small fragments of MATLAB code can be transferred between partners. Initially, as a stepping-stone, this MATLAB code can be run offline by the WP5 partner. This will allow the offline processing of results captured using GNU Radio (firstly without, and then with the USRPs in-the-loop). This can be used to ensure that the results match expectations. This will help to reduce the number of iterations required of data passed between partners by allowing for the WP5 partners to perform some superficial analysis of results 'in-house'.

Once this has been achieved, the MATLAB code can be compiled as a shared library and run within a C++ block in GNU Radio. Proof of concept has also already been achieved in the CIP case where the MATLAB code has been compiled and embedded in a new GRC block (GNU Radio's graphical user interface (GUI)). Code snippets are shown in Figure 12 and Figure 13.



```
/*
 * The private constructor
 */
cip_vcivc_impl::cip_vcivc_impl(int v_len, int r_max, int l_max)
  : gr::block("cip_vcivc",
          gr::io_signature::make(1, 1, sizeof (gr_complex) * v_len),
          gr::io_signature::make2(1, 2, sizeof(float), sizeof (gr_complex) )),
    d_v_len(v_len),
    d_r_max(r_max),
    d_l_max(l_max)
{
    // Call application and library initialisation
    const char *args[] = { "-nojvm", "-nojit", "-shield minimum" };
    fprintf(stderr, "Attempting to initialise the MCR application...\n");
    if( !mclInitializeApplication(args,2) )
    {
        fprintf(stderr, "ERROR - unable to initialise the MCR application properly.\n");
        exit(1);
    }
    if( !libcipInitialize() )
    {
        fprintf(stderr, "ERROR - unable to initialise the libcip properly.\n");
        exit(1);
    }
}
```

Figure 12: Initialisation of MATLAB compiler runtime and CIP shared library in GNU Radio C++ block

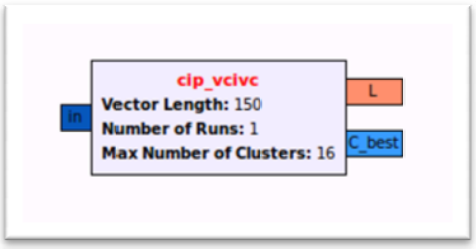


```
// Call MATLAB CIP function
cip(2,L_mw,C_best_mw,r_max_mw,l_max_mw,input_const_mw);
// Find L
L = L_mw.Get(1,1);
if(DEBUG) fprintf(stderr,"L = %d.\n", L);

// Find number of centroids
dims_mw = C_best_mw.GetDimensions();
num_c = dims_mw.Get(1,1);
if(DEBUG) fprintf(stderr,"length(C_best) = %d.\n", num_c);
// Store centroid locations as gr_complex
std::vector<gr_complex> C_best(num_c, gr_complex(0, 0));
for(int c_idx = 0; c_idx < num_c; c_idx++)
{
    C_best[c_idx].real() = C_best_mw.Real().Get(1,c_idx+1);
    C_best[c_idx].imag() = C_best_mw.Imag().Get(1,c_idx+1);
    if(DEBUG) fprintf(stderr," C_best[%d] = %f + %fj.\n", c_idx, C_best[c_idx].real(), C_best[c_idx].imag());
}
```

Figure 13: Calling of CIP shared library in GNU Radio C++ block

The block is called ‘CIP’ and it takes in the received superimposed constellations, and outputs the estimate of the number of sources and the estimate of the best centroids. Both can be calculated and displayed, on-the-fly, in GRC by attaching to the receiver flow graph. Figure 14 shows the block in the GRC GUI.



cip_vcivc
Vector Length: 150
Number of Runs: 1
Max Number of Clusters: 16

Figure 14: CIP GRC block

2.3.3 Stage 3: Replace MATLAB code with C++

The final core stage is to replace the call to compiled MATLAB code that is embedded within the GNU Radio blocks with C++ code. The C++ language is native to the GNU Radio software, and USRP hardware, with the flow graphs created using Python which acts as a wrapper connecting all of the C++ blocks using the simplified wrapper and interface generator (SWIG). As a result the porting of the MATLAB code to C++ will provide significant speed enhancements. The risk associated with this is that manual porting of the code is time consuming. A possible stepping-stone would be for the WP3 and WP4 partners to optimise their MATLAB code by coding high-complexity sections using Mex. This is where sections of C-code are called from MATLAB, and this can implemented and easily tested by the WP3/WP4 partners within the MATLAB environment. This would reduce the amount of manual porting that has to be performed by the WP5 partner.

2.3.4 Advanced stages

If the previous stages are achieved within the time-frame of the project and further speed-up, or precise real-time performance is required, it may be possible to implement some functionality on the field programmable gate array (FPGA) of the USRP. The tool chain required is still to be investigated and therefore represents some risk.

2.4 Algorithm transfer interfaces to the SLS

The role of the SLS in the DIWINE project is two-fold. Firstly, it serves as a platform for evaluating and demonstrating algorithms whose complexity exceeds the capabilities of the hardware platforms. Then, it serves as a bridge between theoretical algorithm development and system integration. Key DIWINE innovations proposed in WP3 and WP4 will be transferred to the SLS simulation platform for implementing a proof-of-concept to prove their feasibility and obtain preliminary results. In what follows, the methodology for the transfer of algorithms from high-level MATLAB functions to C++ routines and incorporating to the SLS is presented. Due to being a flexible, high-level tool, the SLS allows rapid prototyping of algorithms and simulation scenarios.

2.4.1 Algorithm transfer steps

The transfer of DIWINE algorithms to the SLS can be achieved with a three-step procedure, which essentially converts MATLAB functions to C++ classes.

2.4.1.1 Transfer of channel and error models

New channel and error models can be jointly defined by partners and incorporated into the SLS. A cross-layer API has been implemented between the underlying channel, implemented with MATLAB code by the academic partners, and the SLS. The goal was to devise a generic methodology, which would allow the academic partners to provide the channel model in a format appropriate for incorporating in the SLS. The approach taken was to modify the SLS so that it can accept MATLAB vectors with the channel realisations, which can be loaded to the simulator in the form of MATLAB (*.mat) files. The same approach can be followed for the packet error function, in case the internal error model, which is based on signal-to-noise ratio (SNR) measurements, does not suffice. The function can be approximated with a look-up table versus the signal transmit powers and the noise levels.

2.4.1.2 New packet-level statistics

The SLS calculates a wide range of packet statistics, at both node-level and network-level. Specifically, it keeps track of data traffic generated and received throughout the network, as well as energy consumption and energy efficiency. The SLS also measures packet delays, packet reception errors and traffic lost due to collisions or buffer overflows. Due to being a flexible tool, the SLS allows new metrics to be jointly defined by partners for the evaluation of proposed algorithms. All statistics gathered by the SLS are stored in trace files for further statistical analysis.

2.4.1.3 DIWINE algorithms implementation

The SLS can be easily extended with new algorithms and protocols, which can be implemented in the form of FSMs. Protocol FSMs typically change states based on measured signal power and signalling carried at the packet headers. This approach enables the DIWINE nodes to make opportunistic scheduling decisions, without the need for a medium access control (MAC) protocol. A proof of concept of this procedure, transferring the interference neutralisation algorithm from TUD to the SLS, has already been implemented. A massive interference scenario was implemented, with four cloud access nodes within one-hop distance, and a set of relays that cancel the interference (see Figure 15). This allows two sets of source–destination pairs to communicate simultaneously. TUD contributes the signal-to-interference-plus-noise ratio (SINR) values for each wireless link that interconnects two DIWINE nodes, obtained with MATLAB offline simulations. These values are then fed to the SLS error model, which emulates packet losses. After each packet transmission, source and destination nodes are randomly permuted, so that different source-destination pairs are selected for the next round of packet transmissions. Relay amplification gains for neutralising interference are accordingly selected by the relays based on the set of transmitting and receiving nodes. This is made possible by including the transmitting node ID at the packet superframe headers, which the relays have to scan.

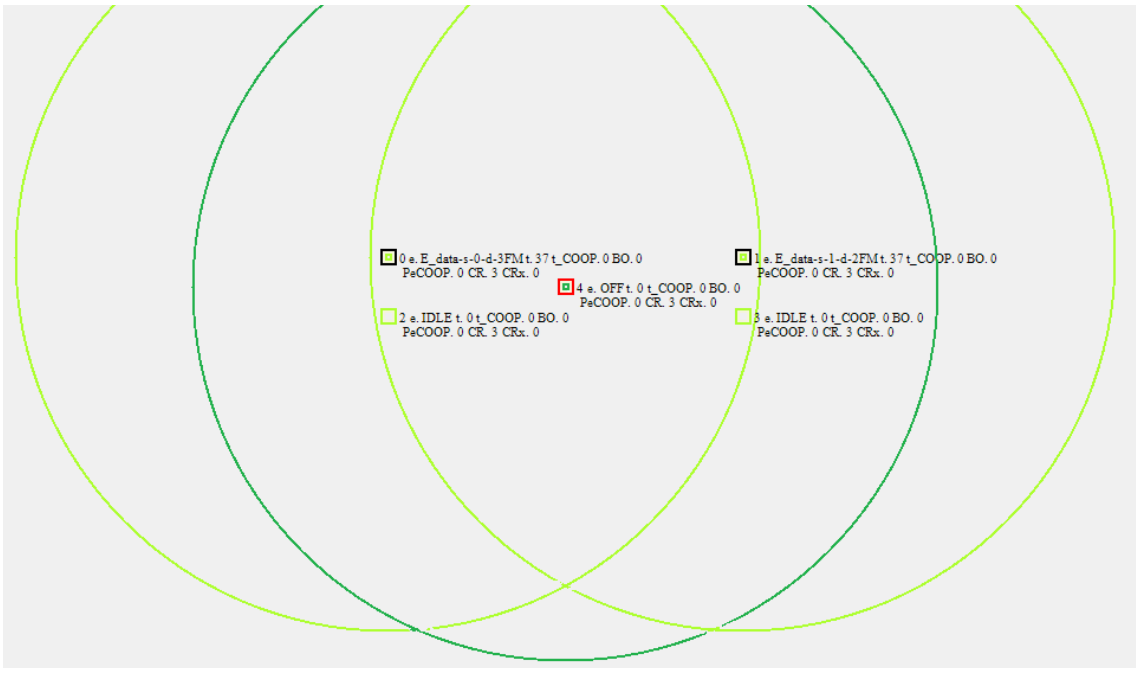

0 e. E_data-s-0-d-3FM t. 37 t_COOP. 0 BO. 0
PeCOOP. 0 CR. 3 CRx. 0
1 e. E_data-s-1-d-2FM t. 37 t_COOP. 0 BO. 0
PeCOOP. 0 CR. 3 CRx. 0
4 e. OFF t. 0 t_COOP. 0 BO. 0
PeCOOP. 0 CR. 3 CRx. 0
2 e. IDLE t. 0 t_COOP. 0 BO. 0
PeCOOP. 0 CR. 3 CRx. 0
3 e. IDLE t. 0 t_COOP. 0 BO. 0
PeCOOP. 0 CR. 3 CRx. 0

Figure 15: Interference neutralisation in the SLS

2.4.2 System-level simulation of hierarchical decode-and-forward

The primary approach to be used in DIWINE at nodes within the cloud, for forwarding data to further nodes or to destination terminals, is HDF. In this approach, hierarchical information (HI), derived from the transmitted information from sources or other cloud nodes, is obtained from the combined signal received from those nodes. A network-coded symbol, which is a network-coded combination of the symbols at the sources, is demodulated directly from the signal at the relay, without necessarily demodulating the two original symbols first.

This is illustrated in Figure 16, in which two sources simultaneously transmit symbols s_A and s_B using BPSK modulation, resulting in the constellation diagrams shown. The modulated signals are received at a relay R_1, and in this case we assume that there is the same attenuation and phase shift on the two channels from S_A and S_B to R_1, so that the signals are received in phase, and superimposed to form the three-point constellation diagram shown. Rather than attempting to demodulate s_A and s_B separately (which in this case could not be done unambiguously), R_1 demodulates a symbol which is a network-coded combination of the two, $s_{AB} = C(s_A, s_B)$. In this case the network coding function, $C(s_A, s_B)$, is simply the XOR function, but in other cases more complex functions may be used.

This HI obtained at R_1 is then re-modulated and transmitted (in this case as BPSK symbols) to R_2. It is supposed that R_2 has access to s_A, perhaps through a direct link from S_A, and it can be observed that this allows it to reconstruct s_B, in this case by repeating the XOR function, this time between s_A and s_{AB}. The information from S_A is referred to as H-SI because although it does not contain information about s_B it nevertheless allows s_B to be reconstructed at R_2.

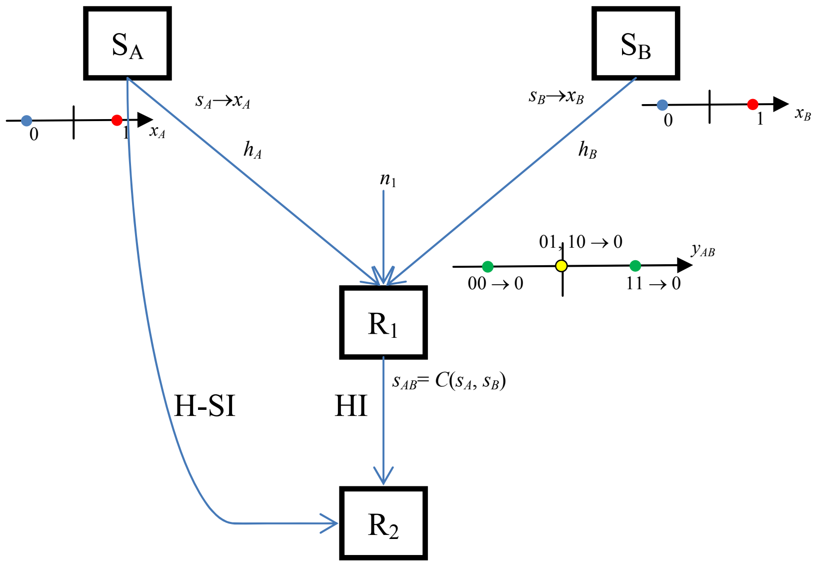


Figure 16: Hierarchical decode-and-forward for BPSK signalling

In general, the sources may generate symbols of greater radix than 2, transmitted using higher order modulation than BPSK, there may be more than two superimposed signals received at a relay, and the function $C(s_A, s_B)$, referred to as the network code *mapping function*, is likely to be more complex than XOR: however these are the basic functions performed at a relay in HDF.

The SLS, however, does not explicitly simulate the transmission of individual symbols: rather it accounts for correct delivery of packets at each node in the network. At a relay using HDF, however, the packet decoded is in general not the same as the packet from either source. It is nevertheless possible to define a packet error at a relay, as the event that the packet $\hat{\mathbf{s}}_{AB}$ decoded is not identical to the result of applying the network code mapping function to the original source packets $\mathbf{s}_A, \mathbf{s}_B$, i.e.:

$$E_P \square \left[\hat{\mathbf{s}}_{AB} \neq \mathbf{s}_{AB} = C\left(\mathbf{s}_A, \mathbf{s}_B\right)\right] \qquad (17)$$

Packet errors in such a system are random events, due to random variations in both the noise, n_1, and in the fading channels h_A, h_B. Moreover it is not possible to define a single SNR, or SINR, since the packet error probability P_{eP} depends jointly on the signal strength from the two (or more) sources as well as the noise. Unlike conventional networks, we cannot identify one source as the signal and the other as the interference. Hence, the packet error probability (PEP), in the case where two sources are involved, can be defined in terms of some function of the ratio of each received signal power to the noise at the relay:

$$P_{eP} = f\left(\frac{\overline{|h_A|^2}}{\overline{|n_1|^2}}, \frac{\overline{|h_B|^2}}{\overline{|n_1|^2}}\right) = f\left(\rho_A, \rho_B\right) \tag{18}$$

where *f* is some function to be determined. Note that we assume unit transmit power from each source.

The function *f* is to be determined by physical layer research, conducted within WP4 of DIWINE. It is possible that a closed form expression, exact or approximate, may be obtained, but if this is not feasible, the most straightforward approach will be to draw up a look-up table of P_{eP} versus ρ_A and ρ_B, obtained by simulation. An example of the result of such a simulation is shown in Figure 17. Here the two sources generate convolutionally coded quadrature phase shift keying (QPSK), and the relay employs a fixed network code mapping function, with hard decision decoding of the convolutional code. Note that in this example, the PEP is a symmetric function of the two channel-to-noise ratios, which would facilitate a reduction in implementation complexity when integrated into the SLS and ultimately allow for more complex scenarios to be considered.

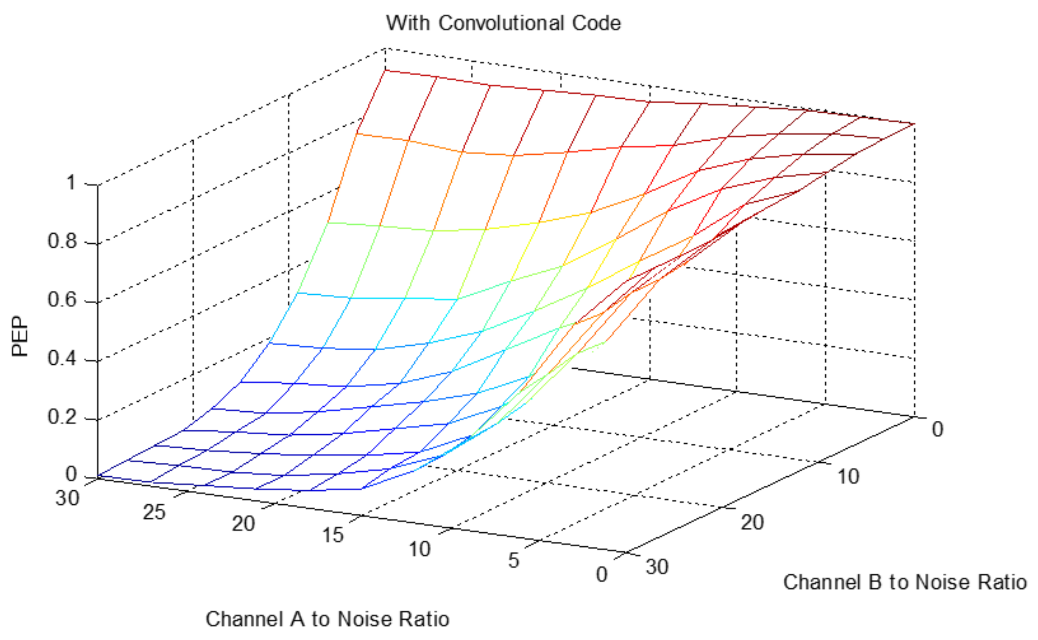


Figure 17: Simulated packet error probability versus SNR on source relay channels

In a run of the SLS, path loss calculations would be performed to determine ρ_A and ρ_B, and then P_{eP} would be calculated using the function *f*, obtained by physical layer simulation or otherwise, using the appropriate set of assumptions about modulation and network code mapping. A given packet would then be deleted at R_1 with probability P_{eP}.

Note that in (18) it is assumed that the PEP depends on the mean signal power on the two links, averaged over Rayleigh fading. In practice of course it will depend on the value of h_A and h_B encountered by each packet, and the approach implicitly assumes a block fading channel model, in which the channels remain constant for the duration of one packet but are uncorrelated from one packet to the next. However in the DIWINE scenario it is likely that the fading is quasi-stationary, varying only over time-scales significantly longer than a packet duration. However the block fading model can be used in cases where either the relay does not adapt at all to the source-relay channels, i.e. it uses a fixed network code mapping, or where the mapping is determined only by h_A and h_B, and is not controlled by other parts of the network, because there is then no dependency from one packet to the next. To investigate the effect of adaptation across the whole network, as when the destinations negotiate with the intermediate nodes to establish the mapping functions they will use, will require a more realistic fading model, and will require f to depend on the instantaneous amplitudes and phases of the channels.

3 System level simulation

3.1 Introduction

The SLS is a simulation platform for wireless networks. Its focus is in simulating Layer-2 protocols, but it also implements physical layer (PHY) functionalities, i.e. simulating the underlying wireless channel and the propagation of wireless signals. The SLS in the DIWINE project serves as a bridge between theory, i.e. algorithms, and practice, i.e. implementation in the hardware platforms. The SLS is a flexible software tool, which will allow rapid prototyping and validation of key DIWINE algorithms and scenarios. One of the key strengths of the SLS is the availability of a GUI. The SLS has been developed in the C++ programming language, using Microsoft .Net Framework. It thus requires a Windows operating system to operate.

During the design phase of the SLS, a lot of competing solutions were evaluated, which nevertheless did not exactly fulfil the requirements. One of the most well-known and versatile simulation tools available is OPNET Modeler [8], which supports most layers of the protocol stack. However, its expensive and time-limited license poses a risk, as its availability is dependent on the availability of funding. Open source simulation tools, most notably Network Simulator [9], are a viable alternative, but with a steep learning curve. Scenarios need to be programmed in the form of scripts as no GUI is available, and extending the simulator with new protocols is complex. The design goals of the SLS on the other hand were:

- Implementation of a tool that is easy to use and master. This can be achieved by implementing a GUI for all simulator operations, such as defining the scenario and setting the parameters.
- Packet-level simulation with a focus on Layer-2 protocols.
- Easy to extend with new protocols. This is achieved with a modular object-oriented architecture, which allows simulator components to be easily replaced.

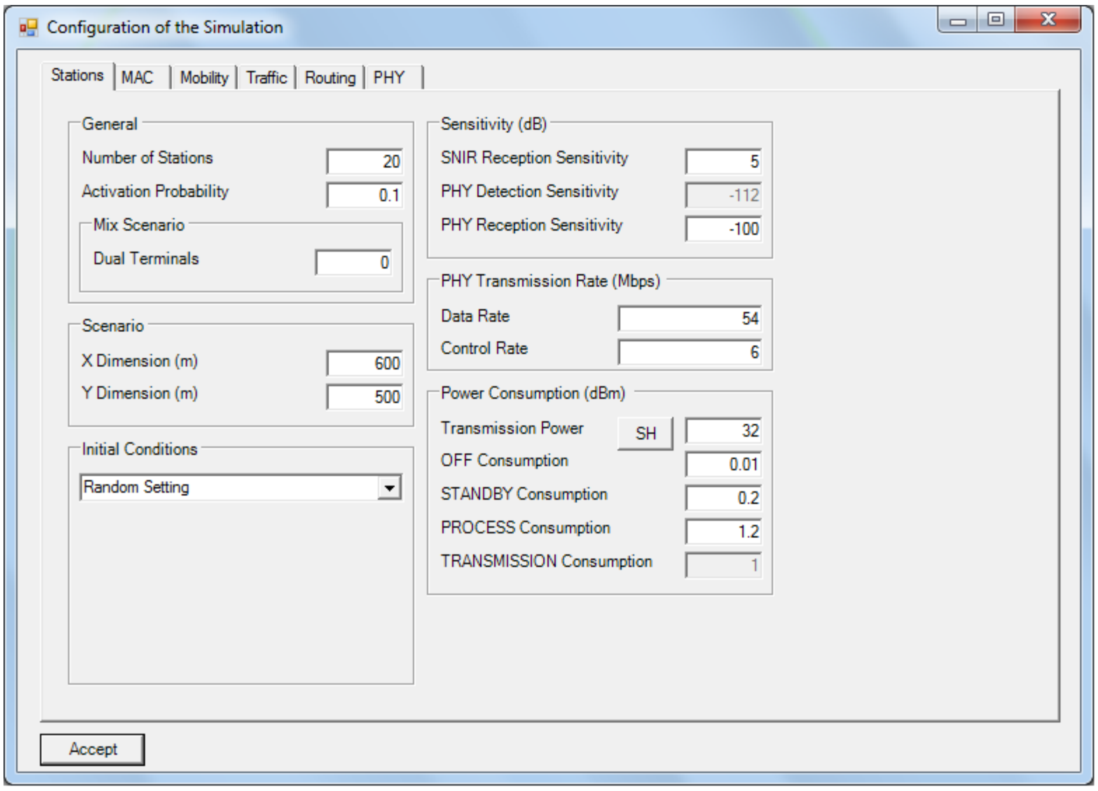
In what follows, the SLS architecture is presented and its supported operations are explained in more detail, as well as the modifications performed to prepare the DIWINE simulation platform.

3.2 SLS architecture and supported operations

One of the strengths of the SLS, and a key differentiator from open source solutions, is the availability of a GUI, which greatly simplifies its operation. It also visualises the operation of the network in real time, a very useful feature in evaluating and debugging new networking protocols. The GUI interoperates with the simulation engine which implements the networking protocols, as well as the simulator components, which are responsible for generating traffic, calculating traffic statistics, and writing trace files. The main components that comprise the SLS tool are further described in the following sections.

3.2.1 Graphical user interface

As mentioned earlier, the role of the GUI is two-fold. It is responsible for configuring the simulation parameters and defining the network scenario, and also for visualising the network operation. Configuring simulations at the SLS is a matter of filling text-boxes, selecting checkboxes and selecting items from drop-down lists (see Figure 18). Thus, this is a tool that is easy to master, as it does not use external script files, which typically require learning a new programming language, e.g. Python. An important feature of the SLS is the ability to pre-program sets of simulations, which automatically varies simulation parameters after each simulation cycle. This ability allows to schedule multiple sets of simulations that differ on a given parameter to run successively. The simulator typically works unsupervised at a high-end server, and automatically sends the simulation results as soon as they are ready via e-mail. Alternatively, the same simulation can be repeated with the same parameters but different initial conditions, e.g. initial seeds. This helps to ensure that the results are statistically independent, which is a very important requirement in wireless network simulators.



General
Number of Stations: 20
Activation Probability: 0.1
Mix Scenario
Dual Terminals: 0

Scenario
X Dimension (m): 600
Y Dimension (m): 500

Initial Conditions: Random Setting

Sensitivity (dB)
SNIR Reception Sensitivity: 5
PHY Detection Sensitivity: -112
PHY Reception Sensitivity: -100

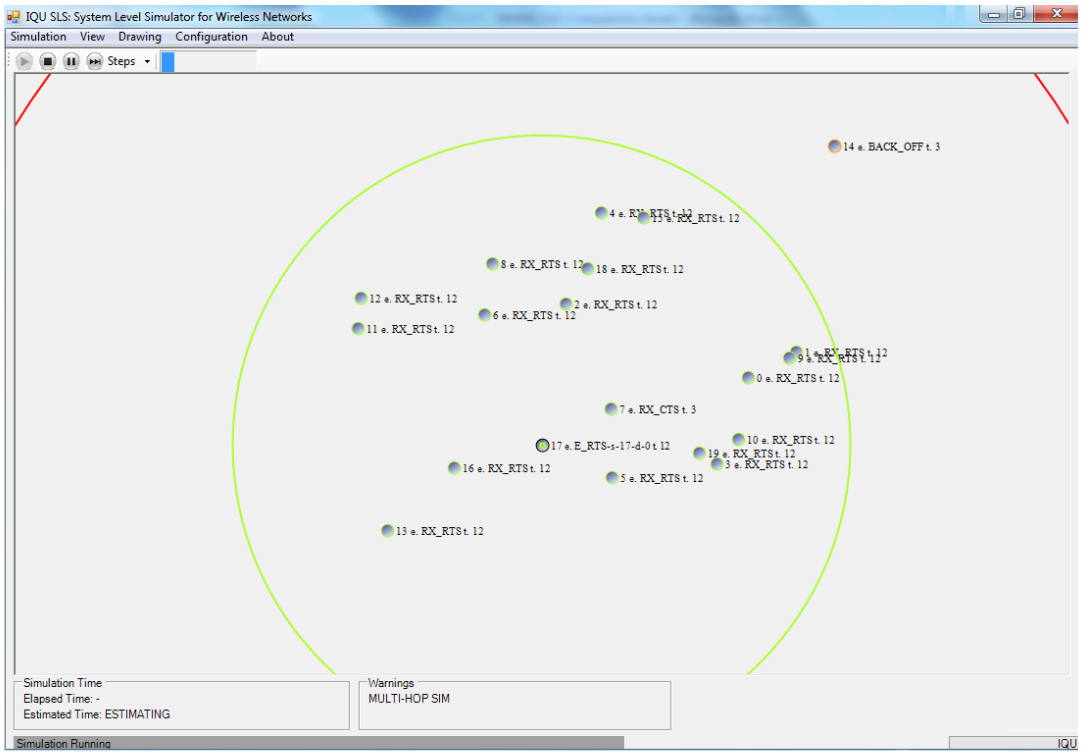
PHY Transmission Rate (Mbps)
Data Rate: 54
Control Rate: 6

Power Consumption (dBm)
Transmission Power SH: 32
OFF Consumption: 0.01
STANDBY Consumption: 0.2
PROCESS Consumption: 1.2
TRANSMISSION Consumption: 1

Figure 18: SLS Configuration screen

One of the most important features of the SLS GUI is its ability to animate the protocol operation, in a user-defined time reference and step intervals. A colour code is employed to symbolise the moving stations facilitating the interpretation of the results; different colours are used for the stations depending on their operating state, e.g. depending on whether the station is transmitting or receiving. The SLS GUI can help in validating the correct operation of networking protocols, facilitating protocol development. The SLS allows printing the values of important protocol variables alongside the network nodes. This feature supplements the common debugging process of using breakpoints and stack tracing, significantly helping the designer identify bugs in the protocol operation. The variables to be inspected at run-time are user configurable, and the SLS can be easily extended to include more.

After verifying the error-free operation of a new protocol, a macroscopic operation check is still useful, as the designer can gain an insight into how the protocol really works and make further improvements. Obviously, the right design for an efficient wireless MAC protocol requires theoretical analysis of the network operation and its associated characteristics. However, there are cases where various simplifications have to be made to make theoretical analysis tractable. Thus, system-level simulation remains a powerful brainstorming tool.



Simulation Time
Elapsed Time: -
Estimated Time: ESTIMATING

Warnings
MULTI-HOP SIM

Simulation Running

Figure 19: SLS GUI main screen

3.2.2 Simulation engine

As mentioned, the simulator consists of a modular class hierarchy, where individual components can be easily replaced. The SLS modules all have a common time representation, which is measured in (user-configurable) time step intervals. The main simulation loop advances time by a time interval per simulation step. In each simulation step all modules interact with each other to implement the network operations. It must be noted that there is a delay of at least one time interval in the interactions between stations. For example, if a packet transmission starts at time *t*, it cannot be received by the other stations before *t+1*. The main classes that implement the simulation engine in the SLS are:

- The *class_main*. This class implements the GUI of the SLS and also contains the source code of the main simulation loop. It instantiates the simulation classes of nodes and MAC protocols, and initiates the variable tracing procedure. An instance of *class_main* class is initiated when the simulator starts.
- The *scenario* class. The scenario class models the terrain where the simulation takes place. It addresses the size of the simulated scenario, buildings, type of terrain, environment peculiarities, and other details. Any 2D environment can be modelled, including obstacles which constrain the free movement of stations.
- The *wireless_channel* class models the wireless radio channel. This includes modelling the signal propagation in the terrain and simulating packet losses. At the beginning of each simulation step, Wireless stations are responsible for reporting to the channel class their transmission power (or 0 for not transmitting). This is used by the channel class to calculate an *N*x*N* power matrix with the received power from any active transmission, given the propagation losses and channel fading.
- The *mobile_station* class represents a mobile node, or equivalently a network client. It models the application layer of the OSI model, and generates data traffic which is split in data packets. This class also keeps track of the node position in the terrain, which is updated by the node mobility model.
- The *mobile_MAC* class implements common functionalities of MAC protocols. It models the wireless network interface of network clients, which implement basic functionalities like transmitting and receiving packets, and performing integrity control. This class is extended by classes that implement the actual MAC protocols operations. For example the *mobile_DCF* class implements the carrier sense multiple access (CSMA) algorithm which handles collision avoidance operations, so that multiple users can share the wireless medium.

3.2.3 SLS operation

During the simulation run-time, one instance of the scenario class and the channel class is generated, and *N* instances of the *mobile_station* class and the *mobile_MAC* class (where *N* the number of nodes). These classes, that comprise the simulation engine, interact in order to simulate the network operations. During the SLS operation it is possible for the user to select the time interval (or equivalently the simulation resolution), pause and continue the simulation, or advance the time step-by-step. In what follows, we present the SLS key operations.

3.2.3.1 Mobility model

Mobile stations keep track of their current speed and position in the *x* and *y* axis, which is updated by the node mobility model. The mobility models employed in the SLS can be found in [10]. These models belong to two broad categories, depending on whether nodes move independently or there is some form of correlation in their movement:

- Group mobility models category implements: column, nomadic, pursue, and reference point mobility models.

- Individual mobility models category implements: random walk, random waypoint, random direction, boundless simulation area, Gauss-Markov, and city section mobility models.

3.2.3.2 Data traffic generation

The SLS implements a traffic generator, which simulates data traffic generation in the application layer of the OSI model. Three data traffic arrival patterns were implemented in the SLS:

- Poisson traffic generation, i.e. exponentially distributed inter-arrival times and exponentially distributed holding times (or message lengths), derived from the traffic load parameter.
- Bernoulli process, i.e. generation of a message with a probability *p* in each simulation step, which depends on the traffic load.
- Saturation process, which ensures that each station always has as packet in the buffer ready for transmission.

Each message arrival corresponds to a (user-selectable) fixed or exponentially distributed message length. Messages are fragmented to a number of fixed-length data packets, with a user-selectable length. Data packets are stored in buffers, which implement the first-in first-out (FIFO) policy.

3.2.3.3 Collision models

In each simulation step, stations 'sense the channel' to determine whether it is busy or idle. The SLS uses two thresholds. The *carrier sensing threshold* is defined as the minimum signal strength that can be detected by the station, and characterise the channel as 'busy'. The *reception threshold* is the minimum signal strength required for the channel to attempt decoding the signal. In case the signal power is above the reception threshold and no collision is detected, then it is assumed that the data packet can be successfully decoded. The SLS uses two collision models:

- Binary model: A collision is registered if signals from two or more stations above the receiving threshold are sensed.
- Capture model: A successful packet reception is registered if the received signal strength is above the receiving threshold and the SINR ratio is above a (user-configurable) threshold.

The capture model is much more realistic and should be the default choice for simulations, but the binary model has also been used extensively in the literature, so it was also included in the SLS.

3.2.3.4 Statistics collection

The SLS calculates a wide range of packet statistics, both node-level and network-level, over the run-time of its operation. Specifically, it keeps track of data traffic metrics throughout the network such as delays, data throughput, energy consumption and energy efficiency, packet reception probabilities and collision probabilities, etc. All these statistics are stored in a trace file at the end of each simulation cycle, for further analysis or for generating plots and diagrams.

3.3 DIWINE simulation platform

The role of the SLS in the DIWINE project is to serve as a platform for evaluating advanced algorithms in complex topologies, which would be unrealistic to implement in the hardware demonstrator. It also allows benchmarking and verification of algorithms before submitting to the hardware platform for implementation. The GUI interoperates with the simulation engine which implements the networking protocols, as well as the simulator components which are responsible for generating traffic, calculating traffic statistics, and writing trace files. In order to prepare the DIWINE platform, a set of new DIWINE-specific modules had to be implemented. Specifically, support was added for the frame structure, i.e. ‘superframe’, as well as the DIWINE scenarios. The modular architecture makes it possible to reuse the SLS traffic generators, topology and mobility model classes.

3.3.1 Superframe structure

The DIWINE network uses a new frame structure (as shown in Figure 20), the main purpose of which is to enable scheduling in stages. Each superframe contains forward and backward frames for bidirectional communications, each divided in a number of slots. Each slot corresponds to the transmission of a codeword. The SLS previously supported 802.11 MAC frames, so the new frame structure had to be implemented.

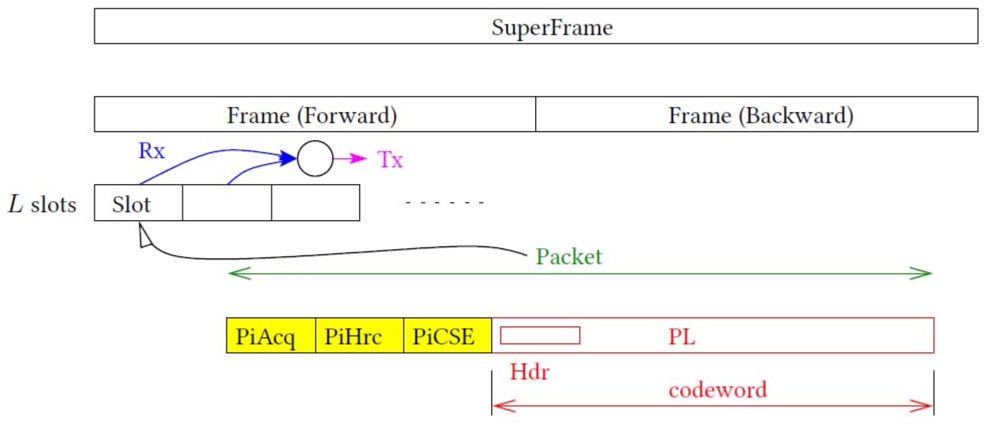
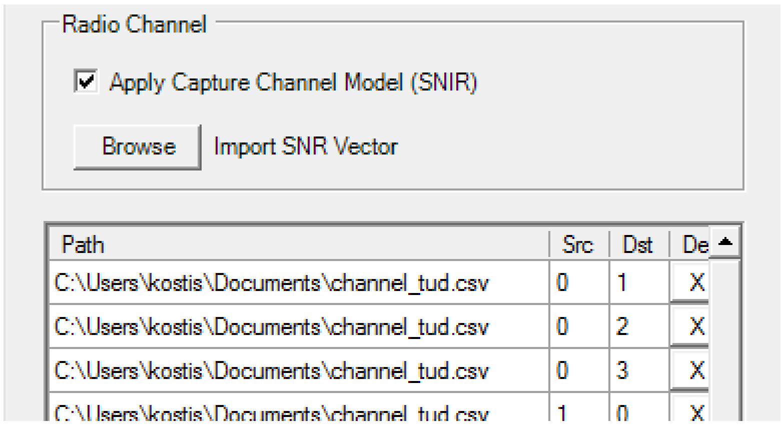


Figure 20: DIWINE 'superframe' structure

Firstly, the superframe pilot headers were implemented, which are responsible for carrying all the synchronisation and signalling information. These are separate from the codeword, as they must be accessible without the need of codeword decoding. Transmission of frames at the SLS is divided in a (user-configurable) number of simulation steps. These are divided in a number of steps dedicated to pilot header transmission, and the rest correspond to the codeword transmission. Finally, the SLS allows multiple frames to be combined and routed as a single entity inside the network, effectively emulating the superframe concept.

3.3.2 DIWINE channel model

In order to converge the SLS packet-level simulations with MATLAB physical layer simulations, we added support for user-defined channel and error models. This mode of operation deactivates the internal channel and error models that employ free-space path loss and Rayleigh fading and allows users to 'upload' their own model in the form of MATLAB vector (*.mat) files. One vector of SINR values is provided per wireless link, obtained with MATLAB physical layer simulations. Packet transmissions are annotated with an SINR value, which is drawn from the corresponding MATLAB vector. The internal vector of packet transmissions along with the distance matrix is used to compute an *N*x*N* matrix of SINR values received by each station from any transmitting node. These values will be fed into the error model to simulate errors during packet reception. It must be noted that contrary to existing collision models in the SLS, in the DIWINE network interference is not always registered as a collision**.** Interfering packets may still be decoded, due to interference cancelling and HDF techniques employed in DIWINE network.


Radio Channel

[✓] Apply Capture Channel Model (SNIR)

Browse Import SNR Vector

Path	Src	Dst	De
C:\Users\kostis\Documents\channel_tud.csv	0	1	X
C:\Users\kostis\Documents\channel_tud.csv	0	2	X
C:\Users\kostis\Documents\channel_tud.csv	0	3	X
C:\Users\kostis\Documents\channel_tud.csv	1	0	X

Figure 21: SLS GUI for submitting vectors with channel realisations

3.3.3 DIWINE network scenarios

The SLS scenario class was extended, to implement the DIWINE massive interference scenario. This scenario includes massively interacting nodes, whose number is much larger than required for full connectivity. The system utilises interference cancelling techniques to enable multiple pairs of nodes to communicate. Effectively the cloud acts as an active environment, which adapts to cancel the interference of direct links. The node model of the SLS was accordingly modified to include smart MIMO Layer-1 relays that implement the interference cancellation functionality. Simulating DIWINE scenarios is now as simple as selecting the scenario in the corresponding dropdown list of the GUI, and uploading the error model vectors. In each scenario, simulation parameters and performance evaluation metrics are jointly defined by partners.

4 Conclusions

This report covers the algorithm transfer methodology and interfaces, as well as the system level simulation platform. It has detailed the methodology for the transfer of key DIWINE algorithms and scenarios in both the SLS and the HW demonstrator platform.

4.1 Algorithm transfer methodology

In this report the algorithms that will be transferred to the SLS and HW demonstrators have been presented, representing a subset of those detailed in WP3 and WP4, which are deemed most suitable for demonstrator implementation. Firstly, the block-structured layered design algorithm was presented. The real-word (hardware) verification of the algorithm will be performed in a simple butterfly network where the information stream through the relay must contain a mixture of both streams, e.g. the XOR function. The second algorithm to be presented was the distributed learning process for HNC selection. This algorithm equips each cloud internal node (relay) with an internal intelligence and ability to self-adapt its private HNC. Next, a simple example for the demonstration of an interference neutralisation algorithm was presented, assuming a simple cloud network with two source nodes, two destination nodes and three relay nodes. Finally, the synchronisation algorithm, which employs CAZAC sequences in frequency frames, was presented.

In order to ensure smooth transfer of the algorithms from the WP3/WP4 partners to the WP5 SLS and HW platforms, a predefined methodology will be followed, which was detailed in this report. The algorithm transfer to the HW demonstrator involves a three-step procedure. At each stage, the algorithm becomes more embedded and autonomous within the HW demonstrator. This will allow for a robust, iterative, approach. At each of the three stages of the algorithm exchange process, the initial debugging will take place without using a USRP in the testing. This additional debugging stage will ensure that all of the baseband digital signal processing is functioning as expected.

Regarding the transfer of algorithms to the SLS; this also consists of a three-step procedure which converts MATLAB functions to C++ classes. The procedure involves first transferring the error and channel models, then implementing required packet-level statistics, and finally implementing the algorithm in the form of a FSM. It must be noted though that the SLS, being a packet-level simulator, does not explicitly simulate the transmission of individual symbols. In order to simulate HDF strategy, it is possible to simulate a packet error at the relay nodes as the event that the packet decoded is not identical to the result of applying the network code mapping function to the original source packets. The packet error function could be determined by physical layer research, conducted within WP4.

4.2 System level simulation

The role of the system level simulator in the DIWINE project is to serve as a platform for evaluating advanced algorithms in complex topologies, which would be unrealistic to implement in the hardware demonstrator. It also allows benchmarking and verification of algorithms before submitting to the hardware platform for implementation. The added value of the system level simulation for DIWINE is its flexibility, which allows rapid prototyping and verification of wireless network algorithms. Additionally, it also visualises the operation of the network in real-time, a very useful feature in evaluating and debugging new networking protocols, acting as a powerful brainstorming tool. In this report all DIWINE-specific modifications have been detailed, as well as the set of new modules that had to be implemented as part of the simulation platform preparation. Finally, the report presented a proof-of-concept implementation of the massive interference scenario, which employs the interference neutralisation algorithm by TUD, and was successfully transferred to the SLS.

Bibliography

[1] Jan Sýkora, Pavel Procházka, Tomáš Uřičář, Tomáš, Hynek, Miroslav Hekrdla, Dong Fang, Alister G. Burr, and Jinhong Yuan, "Terminal node processing for core demonstration scenarios (initial)", DIWINE deliverable D4.01, 2013.

[2] Pavel Procházka and Jan Sýkora, "Block-structure based extended layered design of network coded modulation for arbitrary individual using hierarchical decode and forward strategy", Proceedings of the COST IC1004 MCM, Bristol, United Kingdom, 2012.

[3] Umberto Spagnolini, Monica B. Nicoli, Gloria Soatti, Alessandra Pascale, Vahid Forutan, Ali Parichehreh, Stefano Savazzi, Jan Sýkora, Tomáš Hynek, Eduard A. Jorswieck, Alister G. Burr, and Mehdi M. Molu, "Cloud network processing for core demonstration scenarios (initial)", DIWINE deliverable D3.01, 2013.

[4] Jan Sýkora, Tomáš Uřičář, Pavel Procházka, Tomáš Hynek, Eduard A. Jorswieck, Zuleita K. M. Ho, Kostas Ramantas, Orestis Georgiou, Alister G. Burr, Umberto Spagnolini, Monica B. Nicoli, Stefano Savazzi, Stefano Galimberti, "Definition of scenarios, models, performance metric and utility targets", DIWINE deliverable D2.01, 2013.

[5] Tomáš Hynek and Jan Sýkora, "Non-cooperative broadcast game for distributed decision map selection of relay wireless network coding processing", Proceedings of the IEEE Workshop on Signal Processing Advanced in Wireless Communications (SPAWC), Darmstadt, Germany, 2013.

[6] Zuleita K. M. Ho, Eduard A. Jorswieck, and Sabrina Gerbracht, "Information Leakage Neutralization for the Multi-Antenna Non-Regenerative Relay-Assisted Multi-Carrier Interference Channel", IEEE Journal on Selected Areas in Communications, vol. 31, no. 9, pp. 1672–1686, 2013.

[7] Osvaldo Simeone, Umberto Spagnolini, Yeheskel Bar-Ness, and Steven H. Strogatz, "Distributed synchronization in wireless networks", IEEE Signal Processing Magazine, vol. 25, no. 5, pp. 81–97, 2008.

[8] OPNET Modeler, http://www.opnet.com, accessed 2013-12-16.

[9] Network Simulator (ns-2), http://isi.edu/nsnam/ns, accessed 2013-12-16.

[10] Tracy Camp, Jeff Boleng, and Vanessa Davies, "A survey of mobility models for ad hoc network research", Wireless Communications And Mobile Computing, Special Issue: Mobile Ad Hoc Networking – Research, Trends and Applications, vol. 2, no. 5, pp. 483–502, 2002.