iJOIN

INFSO-ICT-317941

INFSO-ICT-317941 iJOIN

D6.2

Final proof-of-concept results for selected candidate algorithms

Editor:	Jens Bartelt (TUD)
Deliverable nature:	Public
Suggested readers:	iJOIN GA
Due date:	30 April, 2015
Delivery date:	30 April, 2015
Version:	1.0
Total number of pages:	68
Reviewed by:	GA members
Keywords:	iJOIN
Resources consumed	23.62 PM

Abstract

This final deliverable describes the final evaluation results of the testbeds set up within iJOIN. The description of the individual testbeds from the previous deliverable D6.1 has been updated and is reproduced in the Annex for the reader’s convenience. Each proof-of-concept experiment is described with its objective, set up, the utilized performance measures and finally, the results. It is furthermore described how the results relate to the work of the technical work packages WP2, WP3, WP4, and WP5, and a conclusive evaluation is provided on how the results demonstrate the proof-of-concept of the iJOIN approaches.

List of Authors

Company	Author
HP	Giuseppe Coffano (giuseppe.coffano@hp.com)
	Marco Consonni (marco_consonni@hp.com)
IMDEA	Pablo Caballero (pablo.caballero@imdea.org)
	Luca Cominardi (luca.cominardi@imdea.org)
IMC	Umer Salim (umer.salim@intel.com)
NEC	Peter Rost (Peter.Rost@neclab.eu)
TI	Marco Caretti (marco.caretti@telecomitalia.it)
TUD	Jens Bartelt (jens.bartelt@ifn.et.tu-dresden.de)
UoB	Henning Paul (paul@ant.uni-bremen.de)
	Ban-Sok Shin (shin@ant.uni-bremen.de)

History

Modified by	Date	Version	Comments
Jens Bartelt	April 30th, 2015	1.0	Final version of D6.2

Table of Contents

List of Figures

List of Tables

Abbreviations

AMM	Anchor and Mobility Management
API	Application Programming Interface
ASIC	Application Specific Integrated Circuit
AWGN	Additive White Gaussian Noise
A/D	Analog/Digital
BCJR	Bahl-Cocke-Jelinek-Raviv
BER	Bit Error Rate
BH	Backhaul
BSC	Binary Symmetric Channel
CDF	Cumulative Distribution Function
CPU	Central Processing Unit
CPRI	Common Public Radio Interface
CSI	Channel State Information
CT	Candidate Technology
DMM	Distributed Mobility Management
DMZ	Demilitarized Zone
DQPSK	Differential Quaternary Phase Shift Keying
e2e	End-to-end
eNB	evolved Node B
EPC	Evolved Packet Core
ERD	Error Resilient Decoder
FEC	Forward Error Correction
FPGA	Field Programmable Gate Array
GUI	Graphic User Interface
HO	Handover
IaaS	Infrastructure as a Service
IF	Intermediate Frequency
iJOIN	Interworking and JOINt Design of an Open Access and Backhaul Network Architecture for Small Cells based on Cloud Networks
iLGW	iJOIN Local Gateway
iNC	iJOIN Network Controller
iRPU	iJOIN virtual RAN Processing Unit
iSC	iJOIN Small Cell
IT	Information Technology
iTN	iJOIN Transport Node
KVM	Kernel-based Virtual Machine
LLDP	Link Layer Discovery Protocol

LLR	Log-Likelihood Ratio
LO	Local Oscillator
LTE	Long Term Evolution
MAC	Media Access Control
MCS	Modulation and Coding Scheme
MIMO	Multiple Input Multiple Output
MME	Mobility Management Entity
MUD	Multi-User Detection
NMM	Network Model Module
PC	Personal Computer
PHY	Physical Layer
PDN	Packet Data Network
QAM	Quadrature Amplitude Modulation
QPSK	Quaternary Phase Shift Keying
RAN	Radio Access Network
RANaaS	Radio Access Network as a Service
REST	REpresentational State Transfer
RF	Radio Frequency
RPC	Remote Procedure Call
RRM	Radio Resource Management
SDN	Software Defined Network
SISO	Soft Input Soft Output
SLAAC	StateLess Address AutoConfiguration
SNR	Signal-to-Noise-Ratio
SSH	Secure Shell
TCP/IP	Transmission Control Protocol – Internet Protocol (suite)
TEEM	Traffic Engineering Enforcement Module
UDP	User Datagram Protocol
UE	User Equipment
VLAN	Virtual LAN
VM	Virtual Machine

1 Executive Summary

The goal of iJOIN project is to overcome the enormous data traffic increase caused by the evolution of the pervasive spreading of 4G networks through the deployment of innovative small cells based on a special evolved architecture. The two technology conceptual pillars underpinning iJOIN architecture are Radio Access Network as a Service (RANaaS) and Joint Radio Access/Backhaul design and optimisation, duly presented and explained in iJOIN deliverable document D5.3 [1].

As reported in the deliverable D4.3 [2], Software Defined Networks (SDN) is another fundamental area of investigation for the iJOIN project for meeting requirements of a "*higher degree of configurability" (e.g. granularity of information rates, resource sharing and prioritization of operators, traffic shaping, admission control, and load balancing)* for backhaul (BH) networks.

Inside iJOIN, Work Package (WP) 6 is in charge of running experimental proof-of-concept of the key technology components (in the project glossary, *candidate technologies* or CTs). The setup of three different testbeds, each covering one of the central iJOIN principles (RANaaS, joint access and backhaul design and SDN), the definition of suitable CTs and implementation of interfaces allowing to run CTs over respective testbeds have been previously reported in D6.1 [3]. The final evaluation of the approaches now shows that the proposed technologies do not only work well in theory but also in practice.

An LTE (long term evolution) turbo decoder as key component of the radio access network (RAN) transceiver chain has been implemented on an out-of-the-box cloud platform, showing the feasibility of using commodity hardware for implementation of RAN functions in the cloud. Additionally, a resource scheduling algorithm for ~~a~~ RANaaS platform has been evaluated which allocates virtual processors for the decoding processes in an efficient way.

With the implementation of the joint RAN and BH testbed, the feasibility of mmWave as a backhaul technology in terms of capacity is demonstrated. The evaluation of several algorithms shows that they put different requirements on the reliability of such a link. To cope with a lower reliability of mmWave links, an error resilient decoder can be employed. This testbed has leveraged the demonstration of various algorithms optimising jointly RAN and BH parts of the network, thus providing the proof-of-concept of key iJOIN idea of joint optimisation.

The SDN architecture to manage network functions and controls fits extremely well within iJOIN architecture. Its usefulness is demonstrated in a network management functionality that dynamically reconfigures the local IP anchor for mobile users, thereby reducing latency and backhaul load. This architecture also allows for configuring the forwarding within the backhaul transport network considering the dynamic network conditions.

2 Introduction

WP6 deals with the proof of concept for the proposed optimised network design in iJOIN which encompasses radio access, backhaul, and core network optimisations. For demonstration activities, three different testbeds have been implemented, each one covering different aspects from the iJOIN system design. This document is the last deliverable of iJOIN WP6 and hence the final experimental results for iJOIN demonstration activities have been reported here.

The three iJOIN testbeds and their updates are described briefly in Section 3. The RANaaS cloud platform demonstrates how the proposed technological approaches may benefit from the flexible processing shift to the cloud centres. Two RANaaS instances are implemented: one at University of Bremen (UoB) and the other at Telecom Italia (TI) in Turin. The joint access and backhaul platform, made available by Technical University Dresden (TUD), implements a 60 GHz backhaul following the principle of hardware-in-the-loop. This platform can be used to demonstrate joint access and backhaul algorithms as envisioned in WP2. The third testbed, provided by Institute IMDEA Networks (IMDEA) and Universidad Carlos III de Madrid (UC3M), is the SDN based platform. This is exclusively used to demonstrate the novel network layer algorithms and proposals, developed in WP4.

Section 4 makes the key contributions of this document by providing the detailed experimental results for the proof-of-concept of iJOIN. For each of the testbeds, it describes the core iJOIN concept for this particular testbed and the other proof-of-concept results which are the technology candidates proposed in iJOIN technology work packages WP2-4. For each of these technical demonstrations, the testbed setup is explained followed by the corresponding evaluation criteria and performance metrics and measurements. The most important and interesting parts of this section are the detailed experimental results obtained by running these proposed candidate technologies over their respective testbeds. Section 4.1 is dedicated to RANaaS testbed results. It gives a general assessment of RAN physical layer (PHY) processing from experimental results focusing on the decoding function implemented in the cloud. PHY and cloud resource scheduling and cloud based multi-layer scheduling results are also provided and explained here. Section 4.2 starts with the general assessment of the mmWave backhaul demonstrator for the performance metrics including data rate, backhaul latency and energy consumption. The three following subsections provide experimental results for the three specific joint access and backhaul algorithmic optimisations (Distributed Multi-User Detection using In-Network Processing, Partially Centralised Inter-Cell Interference Coordination and Joint Coding for Access and Backhaul) which have been proposed in WP2. Section 4.3 provides the experimental results from the SDN testbed by showing the results for different network management functions making use of the SDN based approach. The results here focus on the network Mobility Management capabilities investigated in WP4.

Section 5 discusses the achievability of main iJOIN concepts with respect to the experimental results obtained from the three testbeds and makes the links with the theoretical results.

Section 6 provides the summary and the concluding remarks for this deliverable.

The detailed concept, description and the interfaces for the three testbeds had been reported in the previous deliverable from WP6, D6.1 [3] and here reproduced in Annexes A, B and C for the sake of completeness and to keep this document self-contained.

3 Testbed Description Update

The following picture shows the proposed iJOIN logical architecture for which full details appear in D5.3 [1] and how the testbeds developed in WP6 for proof-of-concept of iJOIN cover the most relevant areas of investigation in this project:

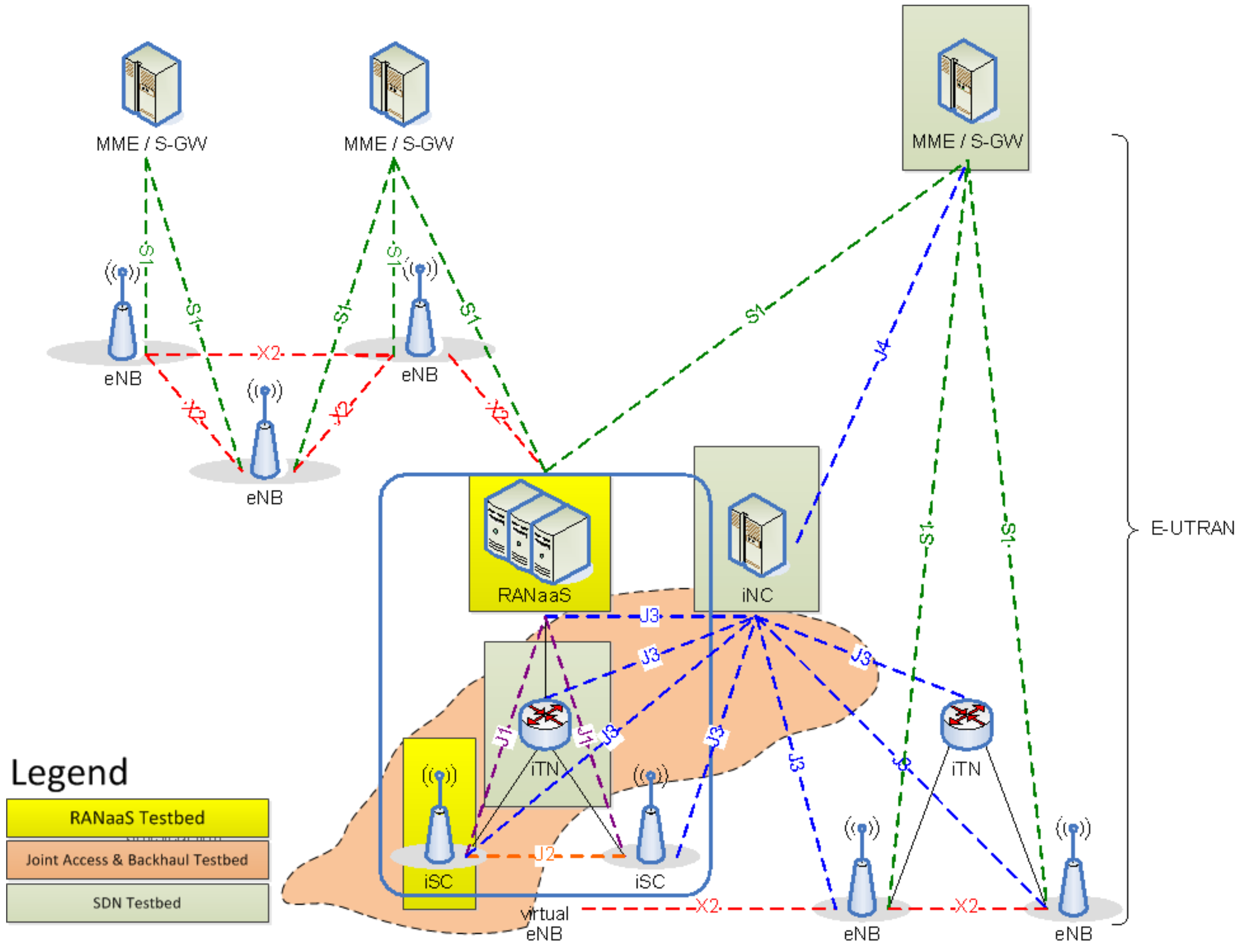


Figure 3-1: iJOIN testbed voverage

The RANaaS testbed is mainly focused on emulating the virtual evolved Node B (eNB) elements which consist of the RANaaS node and iJOIN small cell (iSC) The RANaaS platform hosts radio access network functions, also called RAN services, implemented as programs running in an Infrastructure-as-a-Service (IaaS) cloud platform. Generally speaking, an IaaS cloud platform is a software layer – usually installed on several physical machines connected via a network – that provides services for allocating computing resources in the form of virtual machines, virtual disks and virtual networks. Virtual resources are created for running applications and released when the applications are no more needed.

The joint access & backhaul testbed is intended to be used to investigate and validate the technology for implementing the physical layer of the communication links between the architecture elements. For example, it may provide the physical implementation of J1 and J2 links.

The objective of the SDN testbed is the implementation of algorithms for orchestrating and configuring the network entities in order to optimise the network communications. As such, the effort is mainly focused on the implementation of the iJOIN Network Controller (iNC) while other network entities are emulated.

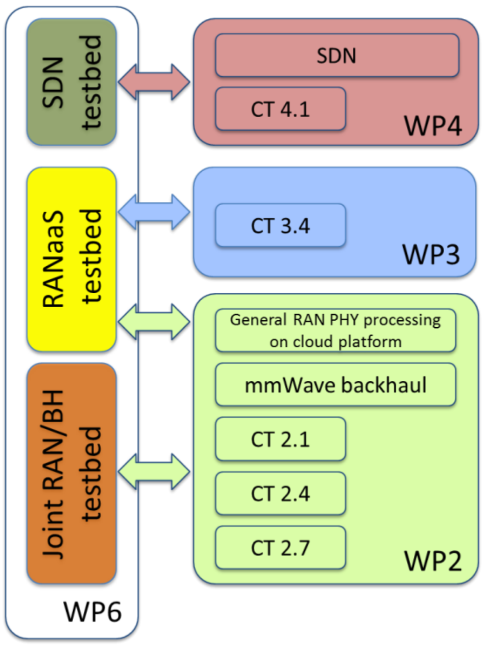


Figure 3-2: Interaction between the WP6 testbeds and the technical work packages

Figure 3-2 illustrates the interaction between the WP6 testbeds and the technical work packages WP2, WP3, and WP4, highlighting the general concepts and CTs that are demonstrated with the testbeds. As can be seen, parts of all technical WPs are represented by the WP6 proof-of-concept, showing that WP6 covers all relevant aspects of the iJOIN approach.

The following subsections provide very brief description of each testbed and also some latest development on these testbeds compared to what was reported in previous WP6 deliverable D6.1 [3]. As this description and the testbed set-up for final evaluation results make frequent references to detailed description and interfaces of each testbed, they have been reproduced from D6.1 in annexes A, B and C for completeness. The minor updates described in the following have been incorporated accordingly.

3.1 RANaaS Testbed

The RANaaS testbed is implemented as a cloud computing platform providing the computational resources for running CTs. OpenStack [4], a free and open-source Infrastructure as a Service cloud-computing platform, is the software suite selected for the implementation of the RANaaS testbed, therefore CTs are realised as Virtual Machines (VMs) running on top of it.

Two RANaaS testbed installations have been implemented during the project: one at University of Bremen (UoB), the other at the Telecom Italia (TI) premises. Both the installations have been deployed on commodity hardware.

A detailed description of the testbed has been provided in D6.1 [3] and reported in Annex A for convenience.

In D6.1, two technology demonstrations on the RANaaS testbed were described. The first one demonstrated the general concept of turbo decoding inside a virtual machine running on a cloud platform and its performance and thus covers WP2 activities. The second one dealt with multi-layer scheduling and robust link adaptation and therefore is focused on WP3. During the deeper investigation of the turbo decoder, in particular, the constraints on it that have to be met in a practical 3GPP LTE system, the question arose whether and to which extent the scheduling algorithm can consider the turbo decoder's constraints and therefore improve overall system performance. For this reason, the dedicated WP3 demonstration on "Multi-Layer Scheduling and Robust Link Adaptation" was dropped and replaced with a joint WP2/WP3 demonstration on the interplay of turbo decoder and scheduler which is entitled "Cloud-RAN Resource Scheduling" and whose description can be found in Section 4.1.2.

Experimental activities have highlighted issues related to the technology used on top of the IaaS layer, when implementing CTs. Some algorithms, that were originally implemented using MATLAB [5] have been re-implemented using 'lower-level' tools in order to have more control on the underlying hardware resources and fulfill the real-time constraints of the application.

As described in section 4.1.2, coding CT's in C++ language and using some specific libraries (e.g. SPUC library [4]), it has been possible to optimise algorithms and have more control on how they are parallelized.

3.2 Joint Access and Backhaul Testbed

The joint access and backhaul testbed consists of a 60 GHz short range wireless link that can be used as BH and a standard personal computer (PC) which can simulate CTs, forming a hardware-in-the-loop platform. With this setup, CTs can be tested utilizing a real-life wireless link to gain insight into the feasibility of using mmWave technology as an alternative to fibre on the last mile of BH networks. A detailed description of the testbed and its interfaces has been provided in D6.1 [3] and can be found in Annex B. In addition to the functionality described in D6.1, a variable code rate for the BH link has been implemented that had previously been described only tentatively. The interface description in Annex B has been updated accordingly.

3.3 SDN Testbed

The SDN testbed is deployed using commodity hardware. The use of commodity hardware allows having a number of nodes large enough to emulate different network topologies at a reasonable implementation cost. The testbed implements both IEEE 802.3 and 802.11 physical transmission protocols. The former, is used for connecting the nodes one to another and to connect the network controller to them; the latter, is used to provide connectivity to user equipments (UEs).

The updates concerning SDN testbed are mainly focused on widening the SDN Testbed with new equipment. This is due by the limitations imposed by the testbed on evaluating the prototyped CTs. Indeed, the testbed torn out to be not large enough in respect of more complex and controlled experiments required by CTs' assessment. Section C.2 reports a detailed description regarding the hardware, the software and the role played by the new equipment in the SDN testbed.

4 Evaluation Results of Candidate Technologies

This chapter becomes the most important contribution of WP6. It provides the detailed experimental results which have been obtained by implementing and running various technical proposals over all the testbeds developed under iJOIN. For all the testbeds, proposals are described and testbed setup procedure is explained. This follows by the experimental results which are discussed with respect to the relevant performance metrics.

4.1 RANaaS Testbed

The first testbed is the implementation of parts of the RAN protocol stack on a cloud-computing platform. The goal of this testbed is to show the feasibility of real-time implementations as well as the control of the computational load in a RANaaS instance. The benefits of this approach are discussed in detail in [6] and [7]. In [8], we performed a comprehensive simulation campaign to evaluate the computational complexity imposed by forward error correction decoding. Furthermore, [9] describes a method based on the framework introduced in [10], which allows to control jointly wireless and computational resources. This testbed integrates both technologies into one demonstrator which is described in the following two sections.

4.1.1 General RAN PHY Processing on a Cloud Platform

In the first part, we describe the testbed which allows for assessing the computational complexity required by forward error correction decoding, in particular a turbo decoder.

4.1.1.1 Description

In this testbed, we assume that the preferred functional split A or B is applied [1], i.e., all functionality above analog/digital (A/D) conversion or symbol detection is centralised in the iRPU. Hence, also the forward error correction decoding must be centralised and represents a major source of computational complexity. In addition, due to the random nature of the wireless channel, also the required complexity to decode a codeword varies. In the case of a turbo decoder, this variance is represented by the number of turbo decoder iterations required to decode the codeword. In addition, the complexity depends on the number of information bits which has been communicated. The overall complexity increases linearly in both number of turbo decoder iterations and number of information bits.

In the previous deliverable D6.1 [3], we presented initial results on the performance of a turbo decoder implemented on the RANaaS testbed. These results have been refined and will be presented in the following.

4.1.1.2 Testbed Set-up

The general properties of the testbed have been detailed in Section 3.1 of D6.1 [3], which is also reproduced in Annex A of this document for convenience of the reader. For this evaluation, a virtual machine equipped with 8 virtual CPUs running Ubuntu Linux 12.04 was set up on the UoB instance of the testbed. In order to measure the performance of the turbo decoder, a simulation chain was implemented in C++ using some functions of the SPUC library [11]. Based on the selected MCS, which are based on the LTE uplink MCS, the program will generate bits of different block length, which are then encoded using a turbo encoder derived from the 3GPP LTE standard, achieving different code rates through rate matching. The bits are then mapped onto 4QAM, 16QAM or 64QAM symbols, depending on the selected MCS. Additive white Gaussian noise (AWGN) corresponding to a certain signal-to-noise ratio (SNR) is added, and subsequently a soft demapping is performed, yielding log-likelihood ratios (LLRs) for every coded bit. These LLRs are then decoded using a turbo loop employing two constituent Bahl-Cocke-Jelinek-Raviv Soft Input - Soft Output (BCJR SISO) decoders. The execution time of the turbo loop is measured using the GNU/Linux kernel's implementation of the clock_gettime() function [12]. The decoded bits are compared to the original bits in order to determine the number of block errors, i.e., the number of blocks not decoded successfully. The whole chain is depicted in Figure 4-1.

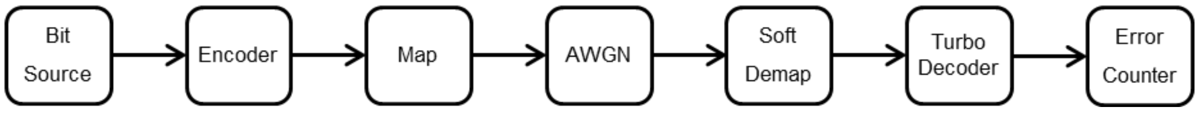


Figure 4-1: Simulation Chain for Assessment of RANaaS Based Decoder

Two different versions of this simulation chain exist. The first one comprises all the function blocks within one single program and uses threads for parallel decoding of different codewords. The second one is split up between demapping and turbo decoding into 2 separate programs which communicate via a TCP/IP connection. Here, a new decoder process is spawned for every incoming connection. While the first one is able generate many measurements within a short time, the second one is able control the arrival rate of blocks to be decoded and the number of concurrent connections, and thus decoding processes. The second version was the focus of the investigations in the previous deliverable D6.1 [3]. Its results are updated later in this section.

4.1.1.3 Evaluation Criteria and Performance Measures

The following results are obtained using the first version of the simulation chain. It runs a simulation for every considered MCS (LTE uplink MCS 6 to 28) and every SNR in the range of 0 dB to 25 dB in steps of 0.1 dB. Every MCS-SNR combination is run as a separate thread, while the total number of threads was limited to 8. Minimum, maximum, average value and variance of the decoding time were measured on a per-thread basis using the clock_gettime() function with CLOCK_THREAD_CPUTIME_ID measurement. This function will use the built-in cycle counters of the CPU cores to measure the execution time. This method has its caveats if threads are moved from one CPU core to another, since their cycle counters might not run synchronously. The virtualization performed by the Openstack platform might increase the inaccuracy even further, as will be seen in the following.

4.1.1.4 Experimental Results

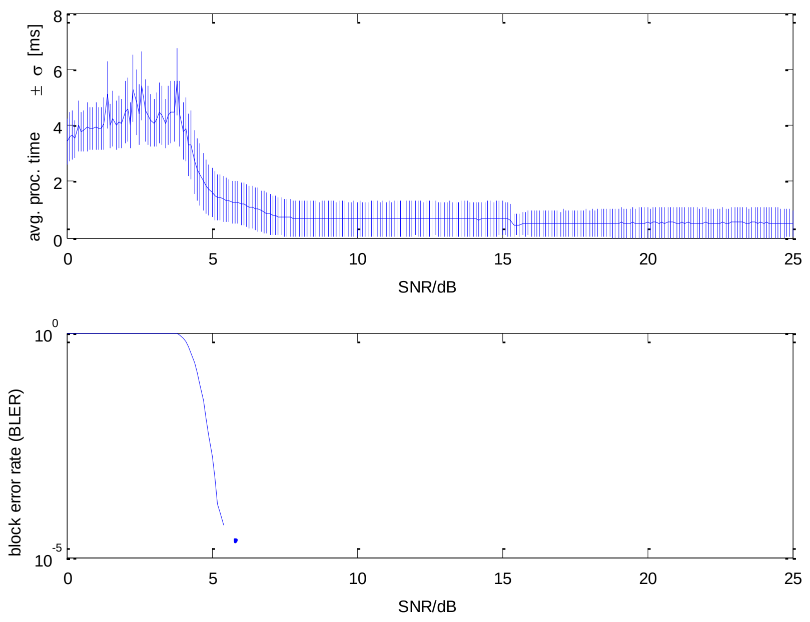


Figure 4-2: Processing time per block for MCS 12

Figure 4-2 shows the processing time ± standard deviation for decoding of one block and the corresponding block error rate. It can be seen that the standard deviation is nearly constant over large ranges of the SNR, but a small step can be observed at approximately 15 dB. The reason for this might be the fact that on cloud platform such as OpenStack, VMs are running concurrently on the same resources. If the CPU resources are thinly provisioned, the actual available CPU resources and its reaction times for a VM will depend on the load of the other VMs. During the measurement campaign, several other VM were performing numerical simulations on the same server hardware.

In order to qualitatively assess the influence of the virtualisation on the decoder performance, the measurements were repeated on a "bare metal" desktop system equipped with a quad core Intel Core i7-2700K CPU running at 3.50GHz with Ubuntu 14.04. The measured decoding times for all MCS 6-28 in the SNR range from 0 to 25 dB are shown in Figure 4-3. It can be seen in Figure 4-3b) that the decoding times on the desktop system in average are higher that on the virtualized system, but don't expose the severe variations that can be seen in Figure 4-3a). Due to time constraints, it could not be figured out whether the decoding time is actually increased by the virtualization or if this is just an artefact caused by impairment of the time measuring function induced by the underlying hypervisor.

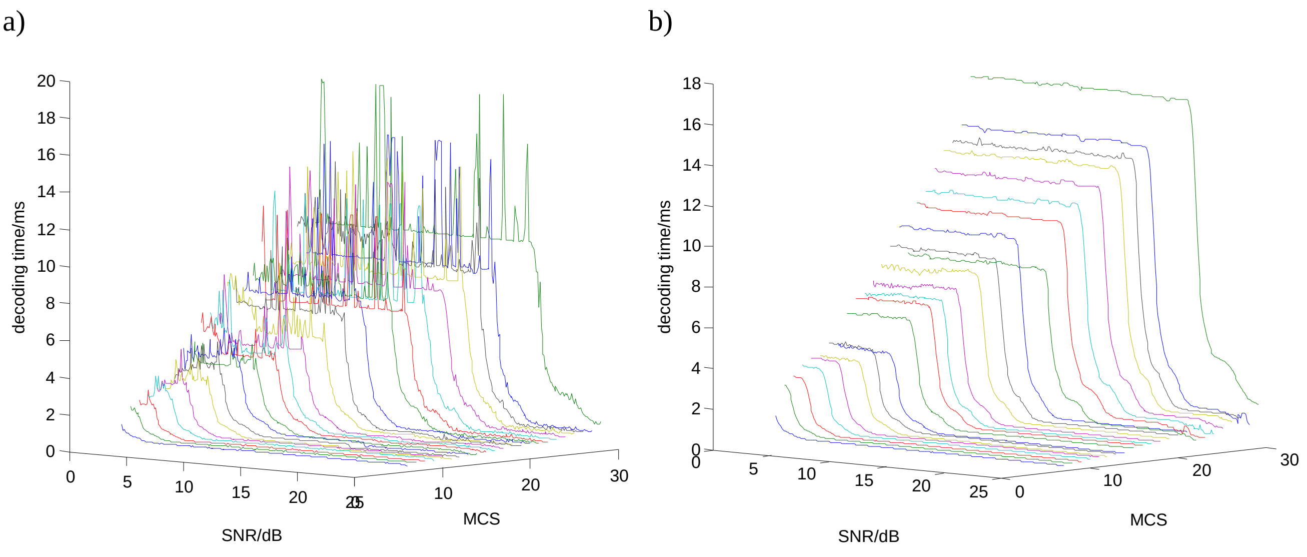


Figure 4-3: Comparison of decoding times on virtualised system (a) and "bare metal" system (b)

The dependency of the decoding time on the employed CPU suggests the introduction of a hardware-dependent factor indicating the normalised decoding performance. If the decoding time is normalised to the product of number of iterations times effective throughput (measured in bits per channel use, bpcu), the normalised values are obtained which can be seen in Figure 4-4. It is striking that in the case of the virtualised system, the normalised value differs significantly for 4QAM, 16QAM and 64QAM MCS between approx.. 0.25 ms/(iterations·bpcu) and approx.. 0.45 ms/(iterations·bpcu). This is not the case for the "bare metal" system where these values only lie in the range of approx. 0.35-0.4 ms/(iterations·bpcu). The fact that the lowest 16QAM and 64QAM MCS have a normalised performance which lies below the remaining ones could not be explained satisfactorily.

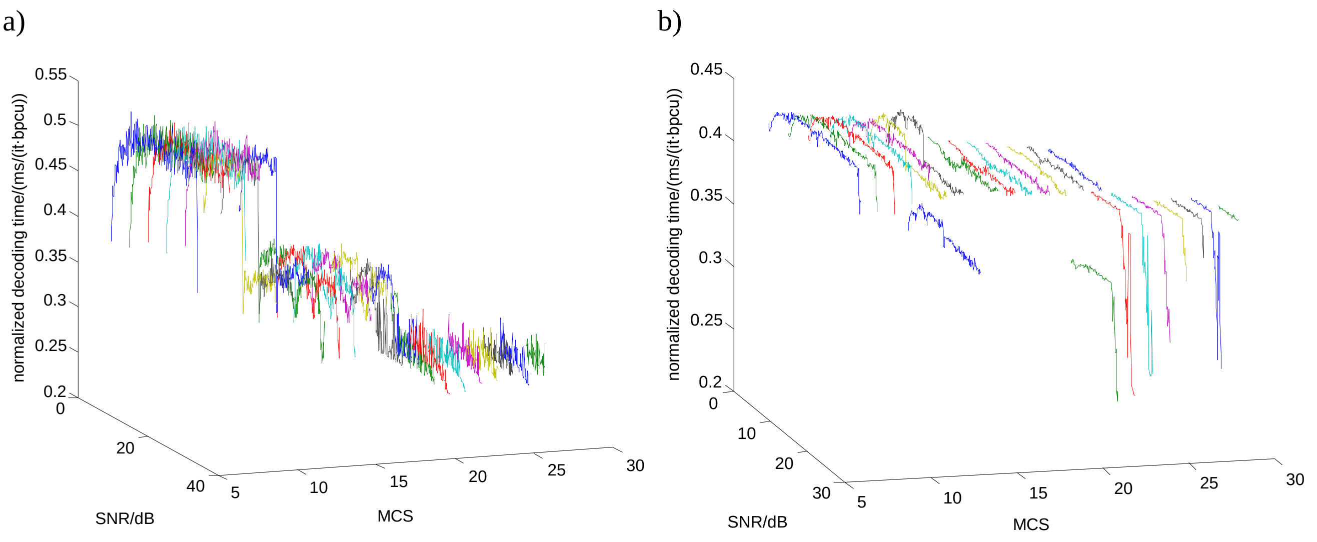


Figure 4-4: Comparison of normalised decoder performance on virtualised system (a) and "bare metal" system (b)

Re-run of simulations

In the previous deliverable [3], results were presented for the case that the processing chain is split between two virtual machines. We showed the dependency of the decoding time and the required number of iterations on the SNR for an exemplary MCS 27. This MCS uses 64QAM modulation and has a relatively high code rate, which leads to higher decoding times than, e.g., lower MCS. Given the precision of the time measurement, this leads to more reliable results.

For these measurements, the CLOCK_MONOTONIC call of the clock_gettime() function was used, which return the actually passed "absolute" time between start and finish of the turbo decoder loop. Even though this time seems to be the practically relevant quantity to assess the decoder performance, the measurement result includes many artifacts stemming from the operating system, its current load and scheduling policy. In order to exclude these influences, the corresponding simulations were re-run with the CLOCK_THREAD_CPUTIME_ID call of clock_gettime() exactly as used above. In order to extend the expressiveness of the results, we also added results for the case where the maximum number of iterations was capped to 2 instead of 8 and extended the simulated SNR range to 16-21 dB. The results are shown in Figure 4-5. It can be seen that a limitation to 2 iterations causes an SNR loss at BLER=0.1 of approximately 1.25 dB, but leads to a significantly reduced processing time per block.

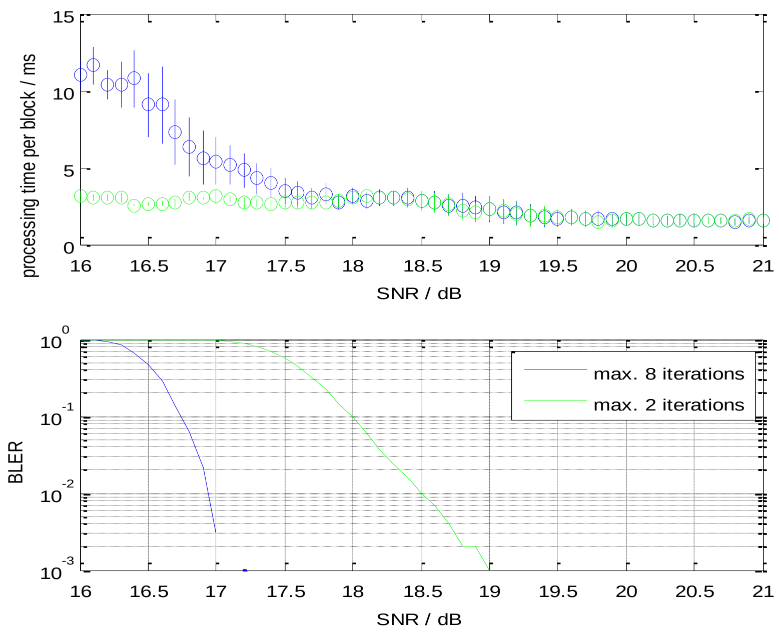


Figure 4-5: Processing time per block and block error rate for MCS 27 for a maximum of 2 and 8 turbo decoder iterations

For the sake of completeness, the histogram of the required number of turbo decoder iterations per block for the extended SNR range of 16-21 dB is shown in Figure 4-6.

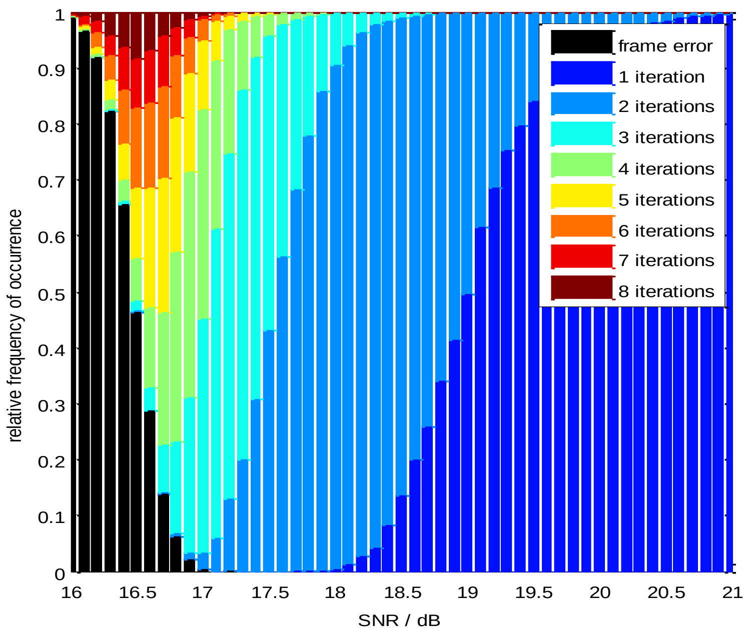


Figure 4-6: Histogram of required number of turbo iterations for MCS 27

4.1.2 Cloud-RAN Resource Scheduling

The second part of the demonstrator is a joint scheduler for wireless and computational resources. This scheduler uses the results of the previously described testbed in order to implement an integrated RANaaS demonstrator.

Note that the development of this scheduler has been performed instead of further developing the semi-deterministic scheduler described in Section 4.1.2 of [3]. The joint RAN/cloud scheduling implementation, which is described in this section, allows for implementing an integrated demonstrator that shows the opportunities of a RANaaS implementation. The semi-deterministic scheduler has been further analysed using an analytical framework as well as simulations in Section 4.4 of [9].

4.1.2.1 Description of Joint RAN/Cloud Scheduling

In [10], we investigated the trade-off between invested data processing resources and achievable data rates. In particular, a framework for the analysis of uplink forward error correction decoding and its required data processing resources has been derived. This framework has been verified using the above testbed. Using this framework, the system can be dimensioned such that with probability $1 - \epsilon$ sufficient data processing resources are provided for an arbitrary base station. However, there is still a probability ϵ that the available data processing resources are not sufficient.

In order to avoid that user codewords are dropped, we described in [9] a joint scheduler for wireless and data processing resources. Let the required complexity for a user served by small-cell k be denoted as C_k. Then, the overall complexity is given by $C_{sum} = \sum_k C_k$ and it is limited by $C_{sum} \leq C_{server}$ where C_{server} is the number of available data processing resources. In order to avoid that users are dropped due to insufficient computational resources, we implement a scheduler that takes into account the computational constraints.

More specifically, the implemented scheduler tests whether $C_{sum} \leq C_{server}$ holds. If it does not hold, it searches for the user which causes the highest complexity, i.e. $k^* = argmax\ C_k$. Then, the modulation and coding scheme of this user is reduced by one step in order to reduce its rate as well as its required computational complexity. We saw in the previous section that the required computational complexity is a strongly non-linear function over the SNR, i.e. reducing the MCS by one significantly reduces the required computational complexity. Usually, the number of required turbo decoder iterations is then reduced to one, i.e. only one turbo decoder iteration is required to decode a codeword.

In [9], we provide a numerical evaluation that the proposed scheduling algorithm is able to avoid any computational outage at the cost of marginal performance loss. For instance, in the case of $\epsilon = 10\%$ the system is dimensioned such that for 90% of all received codewords sufficient resources are provided. In the case that the system is not aware of this limitation, 10% of all codewords would be dropped implying at least 10% performance loss and, in the worst case, even user drops. If our joint scheduler is applied, the performance loss is only about 0.3% which is not perceivable in the operation of a real system.

4.1.2.2 Integrated Demonstrator Setup

In this demonstrator, the above technologies are combined to one demonstrator, i.e. turbo decoder implementation on commodity hardware, computational complexity estimation, and joint scheduling of wireless and data processing resources.

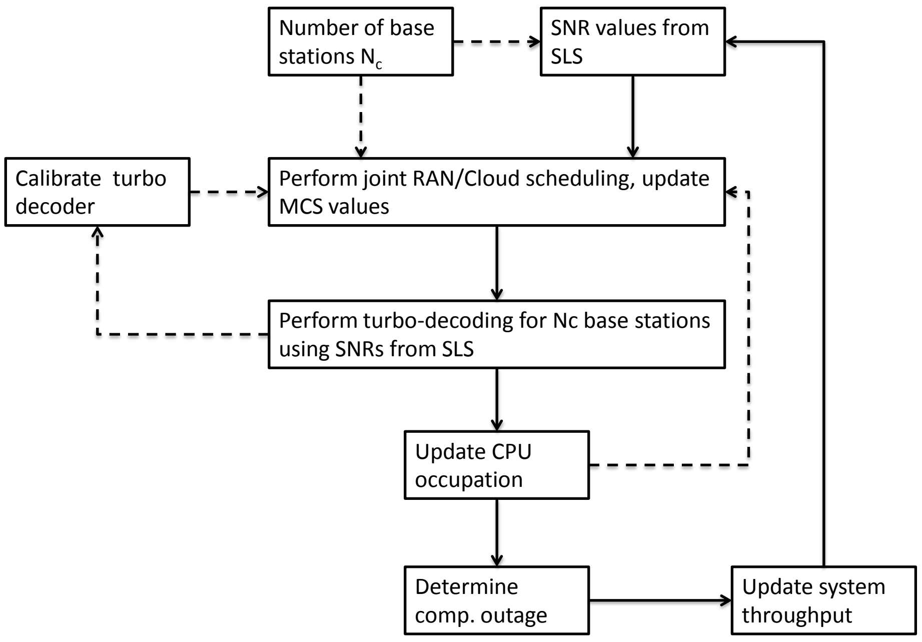

Number of base stations N_c
SNR values from SLS
Calibrate turbo decoder
Perform joint RAN/Cloud scheduling, update MCS values
Perform turbo-decoding for Nc base stations using SNRs from SLS
Update CPU occupation
Determine comp. outage
Update system throughput

Figure 4-7: General structure of demonstrator

More specifically, the structure of the demonstrator is illustrated in Figure 4-7. First, based on the number of base stations that are considered for the demonstration (up to 100), SNR values are taken from a pre-recorded SNR data base. This data base has been generated using a 3GPP LTE compliant system level simulator that considers also mobility (and cell handover) in order to make sure that sufficient time-variant traffic and computational load is present. Based on the given SNR values for each base station (and user), the required decoding time is determined. Given the current CPU occupation, the number of available CPUs, and the required decoding time, the joint RAN/cloud scheduler adopts the individual MCS for each user in order to make sure that no computational outage occurs.

After the MCS selection, for each base station the turbo decoding process is performed accordingly to the description in the previous section. For each user, the required decoding time is recorded and the CPU occupation is updated accordingly. The demonstrator will schedule the users' codewords onto the CPUs and make sure that each available CPU is utilized completely for processing codewords. Here, the demonstrator can operate in 2 different modes: In the first mode, serving as benchmark only, if all CPUs are still occupied with decoding when a new codeword arrives, an additional CPU is added. In the second mode, the codeword is discarded, leading to outage. This case represents the practical system, for which the Cloud-RAN scheduling is designed. However, the demonstrator does not run in real-time but maintains a table of CPUs that are occupied with codewords and updates this table in each round.

After the CPU occupation has been updated, the computational outage is determined, the system throughput is determined, and then the next round of SNR values is drawn. One user within the demonstrator receives consecutive SNR values for one user terminal obtained from the system level simulation.

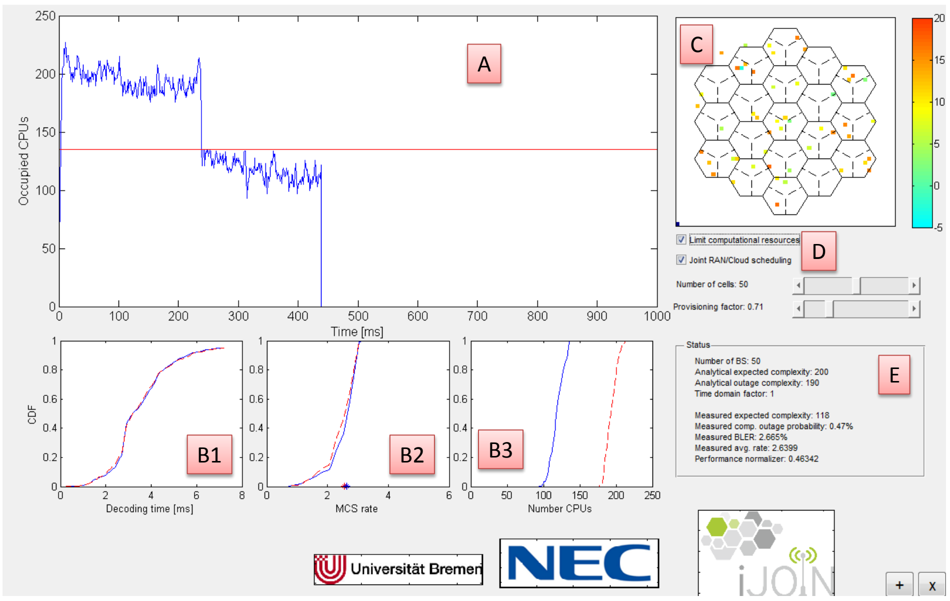
Finally, the demonstrator permanently self-calibrates. This calibration is required to fit the analytical framework and the actual measurements. In order to calibrate the demonstrator, a single indicator is sufficient which measures the required computational time to process one information bit and one turbo decoder iteration. Using this indicator, the developed demonstrator can be deployed on any hardware.

This demonstrator combines PHY layer investigations (i.e., the turbo decoder investigation) from WP2 with RRM techniques investigated in WP3. The link adaptation aspect of RRM, i.e., the MCS selection based on SNR and current CPU load, is directly included into this demonstrator. Additionally, a scheduling of codewords onto a pool of available CPUs is implemented within the demonstrator, which also is a Cloud-RRM technique. However, the actual scheduling of users onto time-frequency resources, i.e., LTE resource blocks, has been sourced out into a full-featured system level simulation, which is performed offline beforehand. This has been done in order to constrain the complexity and thus ensure a satisfactory execution speed of the demonstrator even on standard computers.

4.1.2.3 Evaluation Criteria and Performance Measures

For the evaluation, the number of instantaneously occupied CPUs, the computational outage probability and the average user throughput are used. The ratio of instantaneously occupied CPUs to available CPUs represents the utilization efficiency and will be shown to be almost 100%. The computational outage probability reflects the probability with which a codeword cannot be decoded due to currently missing CPU resources. For expression of the average user throughput, the average MCS rate is used. The ratio between average MCS rate with Cloud-RAN scheduling and without Cloud-RAN scheduling, but unlimited number of available CPU resources reflects the loss in area throughput.

4.1.2.4 Experimental Results



☑ Limit computational resources
☑ Joint RAN/Cloud scheduling
Number of cells: 50
Provisioning factor: 0.71

Status
Number of BS: 50
Analytical expected complexity: 200
Analytical outage complexity: 190
Time domain factor: 1

Measured expected complexity: 118
Measured comp. outage probability: 0.47%
Measured BLER: 2.665%
Measured avg. rate: 2.6399
Performance normalizer: 0.46342

Figure 4-8: Screenshot of the integrated demonstrator

Figure 4-8 shows a screenshot of the demonstrator. In the following, we describe the individual elements of the demonstrator.

Area A indicates the output of the actually occupied CPUs by the current demonstration run. In this particular example, the first 220ms were used to show the unconstrained case, i.e. the performance if there was no computational constraint. We can see that up to 230 CPUs were occupied. Afterwards, the system has been intentionally under-dimensioned with about 130 CPUs in order to show the effectiveness of the described technologies.

Areas B1/2/3 indicate the cumulative distribution of the decoding time for one codeword, the selected MCS rate for codewords, and the number of occupied CPUs. We can see two curves in each area: the dashed red curve indicates the performance for the unconstrained case and the blue curve for the case when the joint RAN/cloud scheduler is enabled. Area B1 shows that the decoding time on this (non-optimised) commodity hardware in the order of 3-5ms in most cases. The application of the joint scheduler does not change this distribution significantly. Area B2 shows the cumulative distribution function (CDF) of the MCS rate. We can see that the MCS selection changes only marginally (also due to the changing channels) which validates the conclusions based on the numerical evaluation discussed in Section 4.6 of [9]. Finally, Area B3 shows the number of occupied CPUs. We can see that the joint scheduler reduces the number of occupied CPUs significantly and maintains the cut-off at 130 CPUs. Hence, although the required number of CPUs is reduced significantly, the actual system throughput is affected only marginally. The demonstrator allows for adding as many curves as required by the user in order to illustrate different setups.

Area C shows the distribution of users based on the system level simulation results. Each dot indicates one user. The simulated environment is an overlay of six hexagonal deployments with 19 sites for each deployment. At each site three macro base-station are set up and within each macro-cell one small-cell is deployed.

Area D shows the interactive elements of the demonstrator. The user can change the number of considered base stations where the value range is restricted to [1; 100]. Furthermore, the system can be intentionally over- or under-provisioned by 50% where 100% corresponds to the 10% outage complexity. The two checkboxes below allow turning on the joint scheduler and enabling the computational constraint. The latter allows for observing the required computational resources if there virtually unlimited resources.

Area E shows information on the current performance such as block error rate, computational outage, average throughput, and average number of occupied CPUs.

4.2 Joint Access and Backhaul Testbed

The core of the joint access and backhaul testbed is the mmWave wireless link. It is used to show the performance of joint access and backhaul technologies using a realistic backhaul. The mmWave link hereby acts as an enabler to the other technologies. However, the hardware of the mmWave link itself is also worth investigating, since mmWave links are a relatively young technology. According to this, the first subsection will investigate the mmWave link in general and the following subsections will cover specific joint access and backhaul technologies.

4.2.1 General Millimeter Wave Technology Evaluation

4.2.1.1 Description

The developed hardware described in Annex B shows how a very simple baseband architecture combined with analog frontends can be used to implement inexpensive and energy efficient mmWave transmitter/receiver modules to provide a wireless backhaul capacity of multiple Gbps on a distance of hundreds of meters. This section evaluates the hardware and implementation used as a hardware-in-the loop to demonstrate joint access and backhaul technologies.

4.2.1.2 Testbed Set-up

The general investigations do not require a special setup and thus use the generic setup described in Annex B.

4.2.1.3 Evaluation Criteria and Performance Measures

The general performance of the demonstrator is evaluated in terms of the following metrics:

- The **data rate** is largely fixed due to the setup, yet shows the feasibility of the system design to provide multi Gbps data rates.
- The **latency** of the demonstrator is estimated based on the required clock cycles for baseband processing.
- The **energy consumption** of the demonstrator is estimated and it is the basis to predict the consumption of a possible integrated solution. The energy consumption can be combined with the data rate to estimate the **energy efficiency.**

4.2.1.4 Experimental Results

Data rate

The data rate of the testbed can be easily computed from basic setup parameters. The symbol rate is $f_s = 3.456\text{ GHz}$ and modulation scheme is differential quaternary phase shift keying (DQPSK) with a spectral efficiency of $M = 2$ bit per symbol. Of the $N_{total} = 101376$ bits per frame, $N_{payload} = 97280$ bits are payload and $N_{header} = 4069$ are header bits. If the code rate is to be assumed as R_c this results in a payload data rate of

$$\begin{aligned} D_{BH} &= f_s \cdot M \cdot R_c^{BH} \cdot \frac{N_{payload}}{N_{total}} = 4.97\text{ Gbps for } R_c = \frac{3}{4} \\ &= 4.42\text{ Gbps for } \mathrm{R_c} = \frac{2}{3} \\ &= 3.32\text{ Gbps for } R_c = \frac{1}{2} \end{aligned} \qquad (4.1)$$

The different code rates can be used to cope with different channel conditions. Figure 4-9 shows the performance of the decoder with the three different code rates over the uncoded bit error rate (BER) of the mmWave channel.

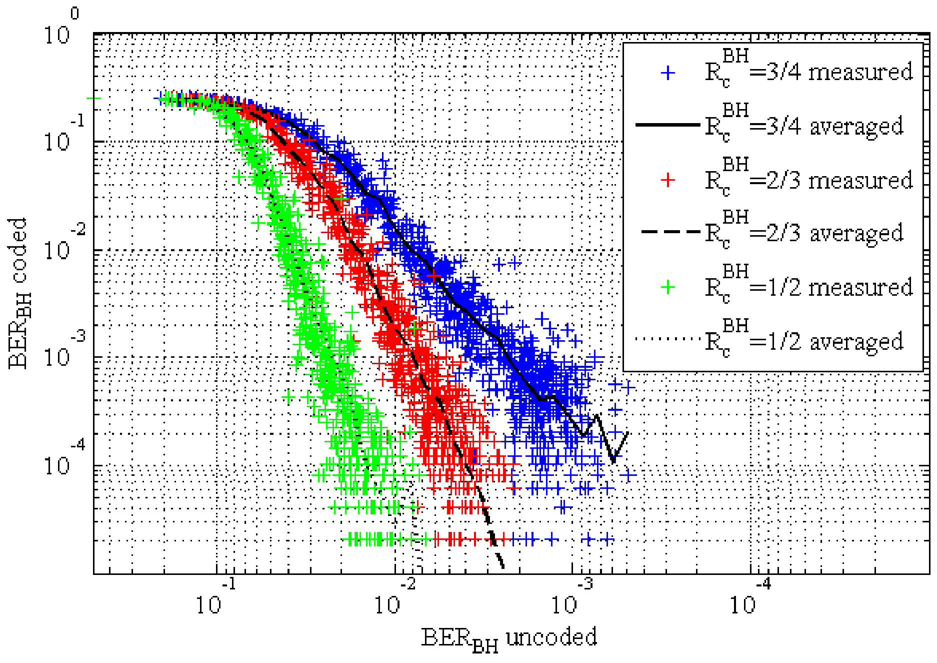


Figure 4-9: Decoding performance using different code rates

The main limitation for the data rate is currently the baseband hardware, yet the basic architecture could be scaled using more advanced field programmable gate arrays (FPGAs). In the European Union, the 60 GHz band offers 9 GHz of spectrum. Assuming that a part of this bandwidth has to be reserved as guard regions, the sampling frequency could be increased to about 8 GHz, resulting in a payload data rate of $D_{BH} = 11.52\ \text{Gbps}$, easily fulfilling e.g. the highest data rate requirement of Common Public Radio Interface (CPRI) of 9.8 Gbps [13].

Latency

The latency of the testbed is mainly limited by the digital baseband unit as the analog frontend only adds propagation times on the order of a few nanoseconds. Due to the FPGA implementation, the baseband latency can be easily calculated from the clock cycles required for the individual processing steps. Figure 4-10 shows number of required clock cycles for the baseband unit as introduced in Annex B.

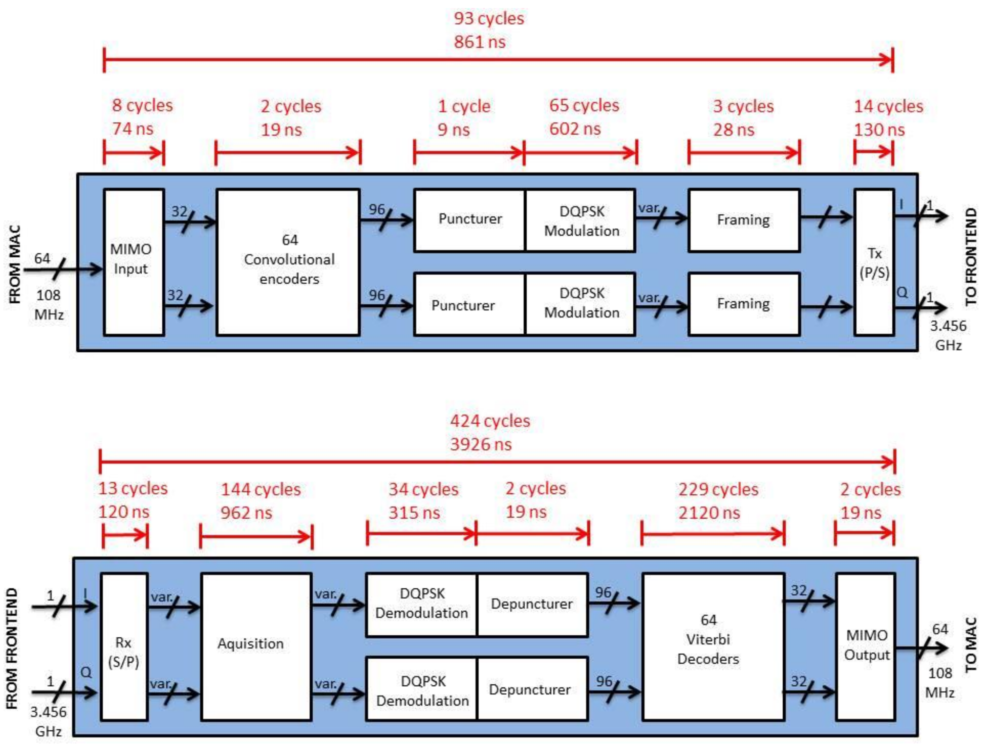


Figure 4-10: Latency of the 60 GHz Baseband

The required processing time can be calculated by dividing the number of required cycles by the clock frequency. The processing time is also given in the figure with the clock frequency being 108 MHz (please note that the individual latencies have been rounded to full ns and hence their sum does not match the overall latency).

As it can be seen, the total latency is about 4.8 µs. This is almost twice the latency requirement of the CPRI standard [13] of 5 µs round trip time. However, it is apparent that the forward error correction (FEC) comprising of the convolutional encoder, puncturer, depuncturer and Viterbi decoder contributes to more than 45 % of the total latency. If we use uncoded BH as proposed by CT2.7, the overall latency reduces to 2.6 µs, almost meeting the CPRI latency constraint. A slight increase of the clock frequency to 114 MHz could reduce the latency to the required 2.5 µs.

Apart from the latency introduced by the hardware itself, more latency is added in a real-life network through transmission via multiple OSI layers. To better measure the latency to be expected from a complete network, we measure the media access control (MAC) layer latency in the testbed by capturing UDP (user datagram protocol) packets before sending them out and when receiving them back after transmission via the 60 GHz demonstrator. Figure 4-11 shows this one-way latency for several different payload sizes. As it can be seen, the latency depends on the actual payload and has a maximum value of about 120 µs.

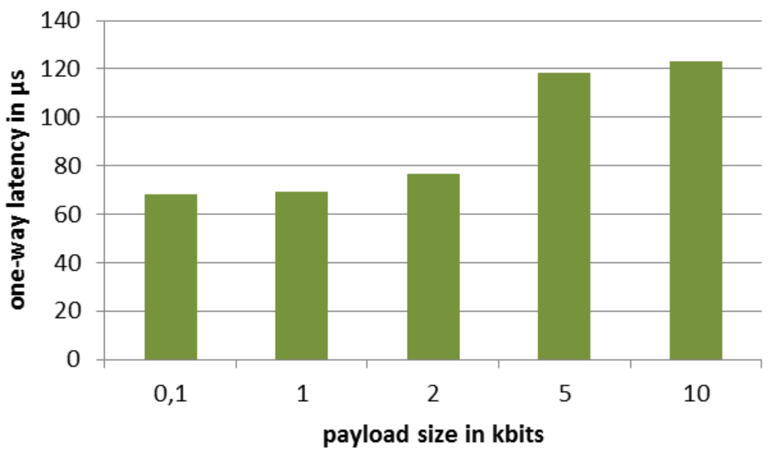


Figure 4-11: MAC-layer Latency of the Testbed

Energy Efficiency

Since the mmWave hardware serves only as a technology demonstrator and needs to be flexible to allow for constant development and improvement, it is not implemented in an energy efficient way, e.g. using FPGAs instead of Application Specific Integrated Circuits (ASICs). However, the architecture itself, especially the employment of a 1-bit A/D converter is the basis for such an energy-efficient implementation. Integrating the baseband into a single chip, we estimate that the total power consumption would be around $P_{BB,tx} = P_{BB,rx} = 1$ W, each for transmitter and receiver. The transmit power given in Annex B as maximum $P_{RF,tx} = 10$ mW accounts for only a very small part of the total power consumption and can be neglected. Even when scaling the demonstrator to achieve larger ranges, the transmit power does not need to be increased, since the additional gain can be achieved by high gain antennas. Table 4-1 shows an exemplary link budget with a transmit power of 5 dBm and 35 dBi antennas, which achieves a range suitable for small cell backhaul.

Table 4-1: Link budget for mmWave link

Parameter	Unit	Value
tx power	dBm	5
tx antenna gain	dBi	35
Max. effective isotropic radiated power (EIRP)	dBm	40
thermal noise (BW 4 GHz)	dBm	-78
rx noise figure	dB	8
receiver noise	dBm	-70
analog losses	dB	6
digital losses	dB	3
req SINR (4QAM)	dB	11.4
receiver sensitivity	dBm	-49.6
rx antenna gain	dBi	35
rain margin	dB	3
oxygen attenuation (300m)	dB	4.5
max pathloss	dB	117.1
max range	**m**	**284**

Consequently, the energy efficiency of mmWave backhaul is estimated as

$$\eta_{\mathrm{BH}} = \frac{P_{BB,tx} + P_{BB,rx} + P_{RF,tx}}{D_{BH}} = 404 \text{ pJ/bit} \tag{4.2}$$

4.2.2 Distributed Multi-User Detection using In-Network Processing

4.2.2.1 Description

In this proof-of-concept, we use the mmWave demonstrator as a hardware-in-the loop testbed. This allows for the simulation of the In-Network Processing algorithms for joint uplink multi user detection (MUD) developed in WP2 [8].

The mmWave demonstrator allows us to use a realistic wireless backhaul transmission channel for the inter-iSC information exchange similar to the ones used in practical systems.

4.2.2.2 Testbed Set-up

For this testbed, a link level simulation framework is used. In this framework the UEs are randomly dropped within a circle with a radius of 50m. The iSCs are regularly dropped on the edge of the circle and are connected in a ring structure via J2 backhaul links. Depending on the distances between UEs and iSCs, path losses are calculated according to the urban micro NLOS (non-line-of-sight) formula and integrated into the channel model. For demonstration purposes, this MATLAB-based framework is extended such that a function call to the MATLAB interface described in Annex B and the application programming interface (API) described in B.3 is inserted in the position of the code where the information exchange between iSCs is simulated. Figure 4-12 illustrates an exemplary setup using 3 iSCs.

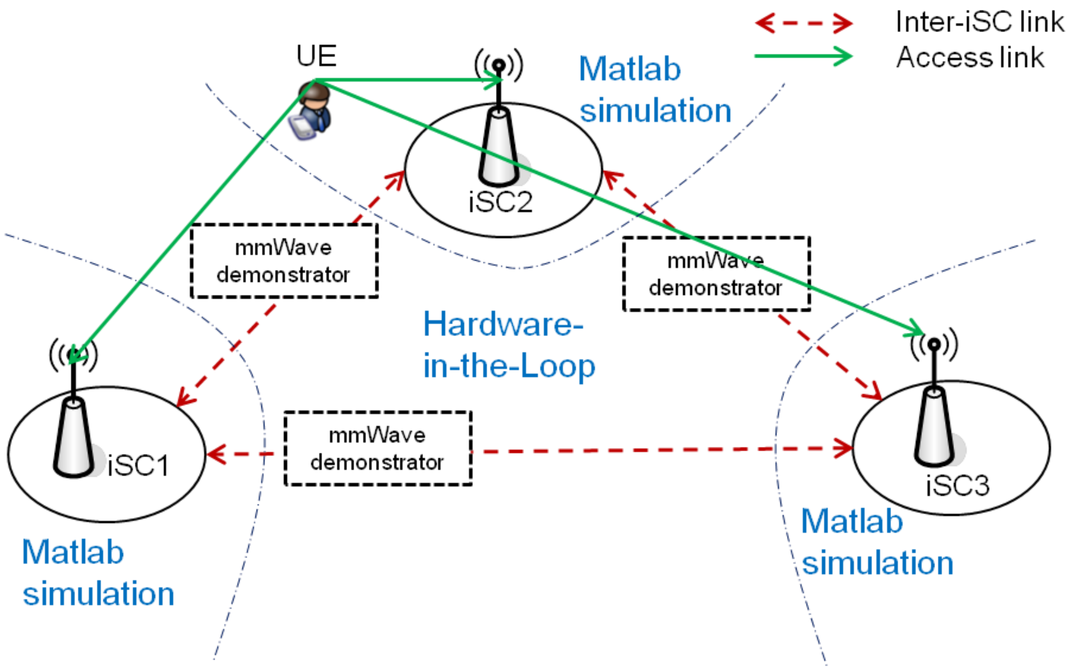


Figure 4-12: Exemplary implementation of CT2.1

While in a practical system the information exchange between iSCs takes place simultaneously, in this case the information on all the selected J2 links between iSCs is transmitted sequentially, since only a single the mmWave demonstrator is available. This which leads to an increased runtime of the simulations. Hence, not all scenarios and parameter sets from CT2.1 can be verified using the testbed.

4.2.2.3 Evaluation Criteria and Performance Measures

The performance metrics are the same as for the simulative assessment of CT2.1, i.e., per UE-iSC-link throughput and frame/bit error rate [8]. The frame and bit error rates for each UE are determined by averaging the error rates over all iSCs in the network. Based on the averaged frame error rates for each UE, the throughput and, correspondingly, the area throughput are calculated.

4.2.2.4 Experimental Results

Figure 4-13 shows the area throughput and the averaged BER over the noise variance for the Zero Forcing Distributed Consensus-based Estimation (ZF-DiCE) algorithm [8]. The scenario considers a MUD of 2 UEs each with 1 antenna transmitting 4-QAM symbols by 3 iSCs each with 2 receive antennas with imperfect mmWave BH links. Since the BH BER tends to fluctuate in the actual hardware, the BH BER is measured for each transmission and then averaged over all transmissions. For comparison the performance for a perfect BH link (BER=0) is also plotted.

For this demonstration, 10 iterations of the ZF-DiCE algorithm were performed, and the inter-iSC communication was quantized with 10 bits per complex component and a dynamic range of -4 to 4. The results show that a higher BER on the BH link deteriorates the performance significantly, e.g. for a BH BER of ~10^{-4} a huge loss in the area throughput can be observed. However, for BH BERs of lower than ~10^{-6} the loss in performance is negligible.

Figure 4-14 shows the performance for the same scenario but with 6 UEs. Obviously, due to an increased number of UEs a lower BH BER of ~$2{\cdot}10^{-7}$ is required for a performance close to a perfect backhaul.

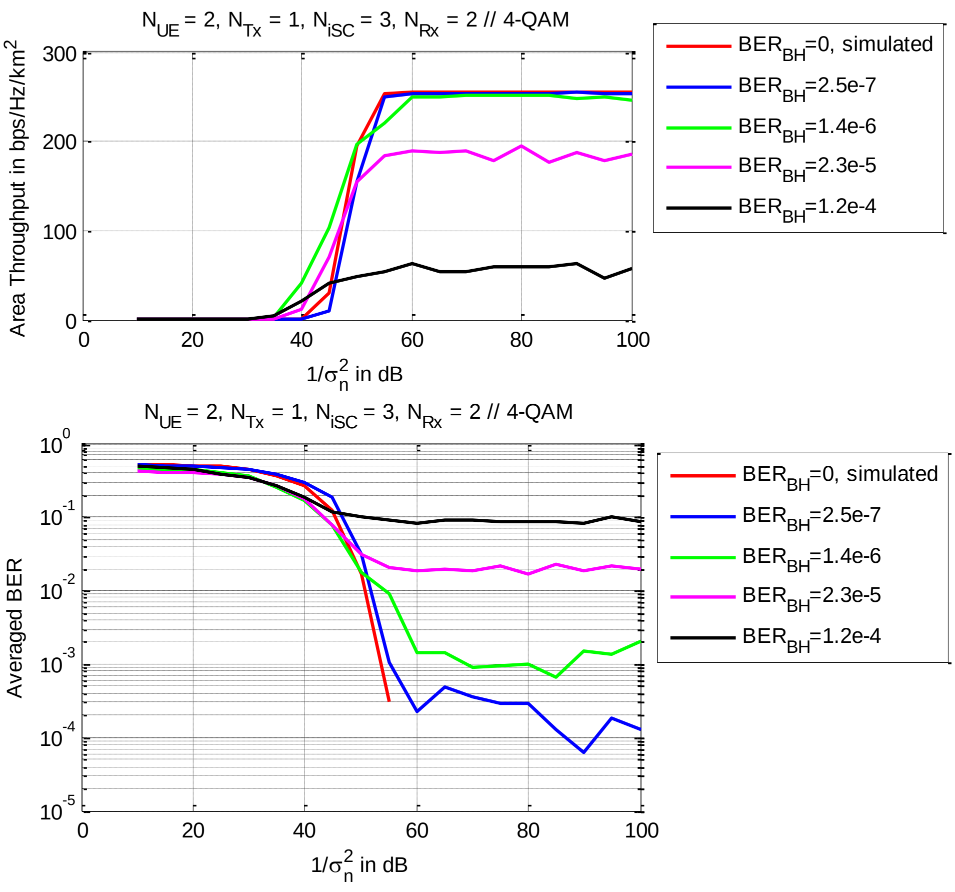


Figure 4-13: Area throughput and BER for different backhaul conditions and 2 UEs

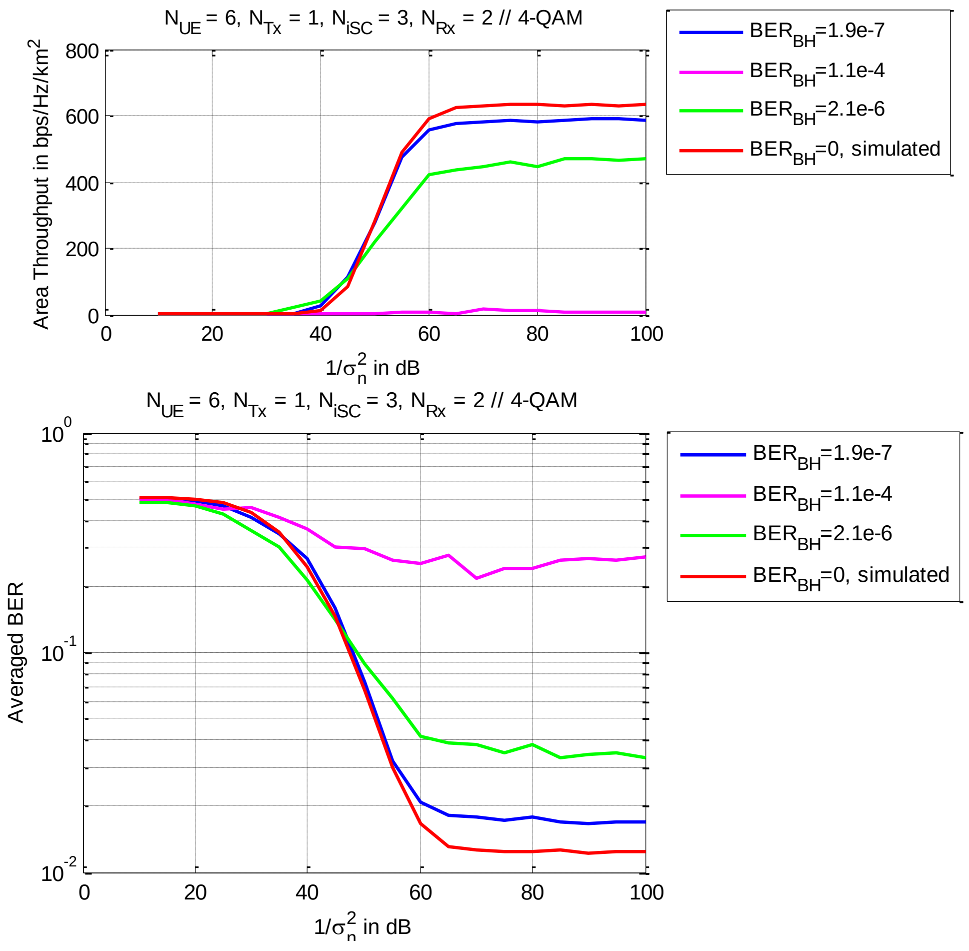


Figure 4-14: Area throughput and BER for different backhaul conditions and 6 UEs

4.2.3 Partially Centralised Inter-Cell Interference Coordination

4.2.3.1 Description

In the state of the art of centralised communication schemes, the multiple input multiple output (MIMO) spatial precoder is usually computed at a central node, for example RANaaS in the iJOIN architecture. The design of the precoder requires an accurate knowledge of the channel state relative to all UEs and all iSCs. Hence, a centralised scheme requires sending all information relative to the channel state to the central node where the pre-coder is determined. Furthermore, this information has to be obtained in a very timely manner as using outdated channel estimates could severely degrade the performance. This implies that backhaul links with large bandwidth and low delay have to be available which is costly and not always possible.

As an answer to this problem, we propose to exploit both the J1 links and the J2 links at the same time, thus spreading the backhaul over all available links and exploiting heterogeneous backhaul technologies in an efficient manner. Splitting the precoder between several nodes, which communicate over limited-capacity links, is a challenging task but provides also increase in the degrees-of-freedom. The objective of this CT is to optimally design the precoder under practical constraints on the J1 and J2 interface, respectively.

The setting under consideration is presented in Figure 4-15.

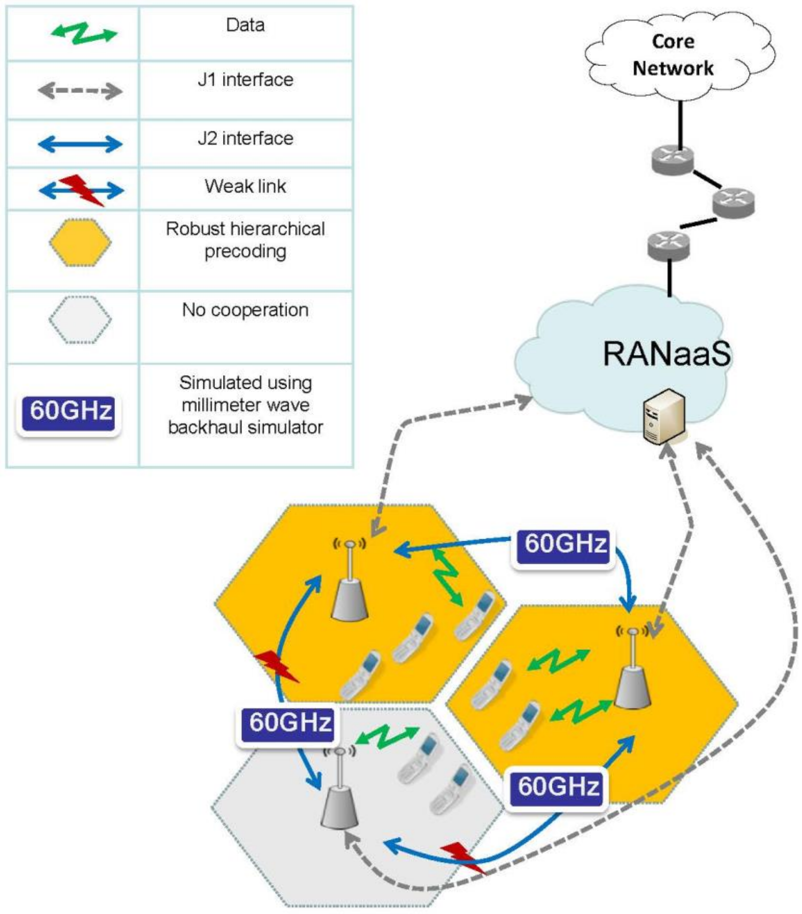


Figure 4-15: Schematic representation of the network configuration considered by the partially centralised approach.

The setting described above is very general and finding an optimal precoding approach is a difficult open problem. Therefore, it has appeared as practically interesting to focus on the "hierarchical" cooperative methods. Indeed, hierarchy is a simple approach to enforce coordination between devices and has been already used in many other areas. In fact, the well-known master-slave organization is a particular case of the hierarchical cooperation with only two cooperating devices. We consider therefore a hierarchical setting where the iSCs can be ordered such that an iSC has knowledge of the channel state information (CSI) available at the previous nodes according to this order, or equivalently to the precoding decision. In practice, we consider a two-level hierarchical cooperation scheme where the iSCs can be split in two groups. The first group has very limited CSI and applies a simple matched precoder while the other group receives the global multi-user CSI and transmits using the hierarchical precoding algorithm, thus taking into account the CSI configuration in the precoder design. More details on this CT can be found in [14] and in the conference paper [15].

4.2.3.2 Testbed Set-up

The 60 GHz testbed would materialize the J2 backhaul links being used to exchange the CSI between iSCs in the demonstration of this algorithm. A system simulator will be running on a laptop in the form of a MATLAB program. This laptop will be connected to the testbed to make use of real mmWave backhaul. The focus of the hierarchical precoding algorithm is on the imperfect CSI such that we restrict our analysis to the sharing of the CSI through the backhaul and we assume that all the user's data symbols are available at all the iSCs.

4.2.3.3 Evaluation Criteria and Performance Measures

This CT aims at maximizing the area throughput when confronted with imperfect CSI and imperfect backhaul links. Thus, we will measure how the area throughput evolves as a function of the error probability in the backhaul links. This will allow to evaluate the robustness of the transmission scheme and to set up design guidelines for the millimetre waves backhaul in terms of required probability of errors.

4.2.3.4 Experimental Results

We have simulated the performance attained in the Wide Area Coverage scenario from which a detailed description can be found in [16]. In this network configuration, we simulate a 3 tier network with an hexagonal distribution of the iSCs and a uniform random dropping of the UEs. We then measure the performance over the 7 central iSCs.

In the proposed contribution, the iSCs form two groups. The first group has low CSI acquisition capabilities and transmits using CSI locally obtained. The second group of iSCs transmits using a more advanced hierarchical precoding scheme and receives from the other iSCs a quantized estimate of the multi-user CSI.

We use the mmWave backhaul simulator to realistically simulate the sharing of the CSI through the J2 backhaul links. This transmission occurs with a given BER and we will evaluate the impact of this given BER over the performance. Extending the simulations to take into account the quantization of the channel estimate during the feedback or the estimation error at the UE can be easily done and does not change the main results.

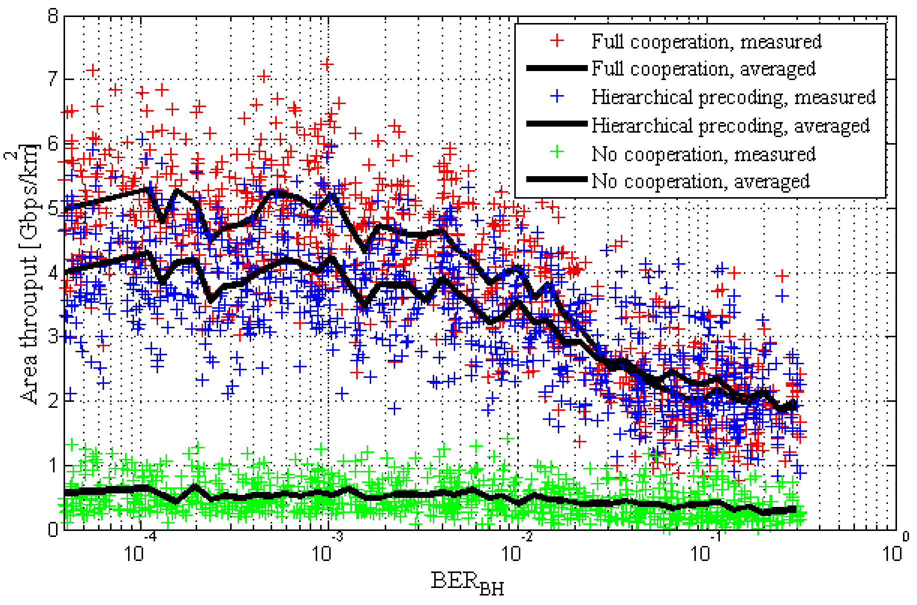


Figure 4-16: Area throughput as a function of the Erroneous BH

We show in Figure 4-16 the area throughput as a function of the BER of the BH links and for three algorithms: one without cooperation between iSCs, on with the hierarchical precoding approach described in Section 4.2.3.1 and [14], and one with fully cooperation, which however is difficult to implement as mentioned above. The basic results are similar to the ones in [14], showing that the hierarchical precoding approach has a slightly lower performance than the full cooperation, however is still much more beneficial than the scheme without cooperation.

More importantly, these testbed results show that the performance of both the hierarchical and full cooperation scheme start to degrade strongly for a BER_{BH} of more than 10^{-3}. Also the performance gap between the two schemes becomes smaller for high BER_{BH}, indicating that both schemes are limited by the BH in that case, and not by the respective algorithms.

4.2.4 Joint Coding for Access and Backhaul

4.2.4.1 Description

In WP2, a wireless backhauling architecture as depicted in Figure 4-17 is proposed, where digitized I/Q data of received signals is backhauled to a central baseband unit before further processing (e.g. decoding). The main difference when using wireless technologies as compared to using fibre for backhauling is higher path loss. The wireless mmWave channel is significantly more attenuated than a fibre channel, which results in a lower average SNR and requires better channel coding to ensure reliable communication. At the same time, the channel conditions of an outdoor mmWave link might vary due to, e.g. rain. To mitigate this, the code rate could be modified as demonstrated in Section 4.2.1. However, this would require some sort of feedback, thereby increasing the latency, as well as reduce the throughput. As discussed in D5.3, strict requirements regarding latency and throughput have to be fulfilled especially for lower layer splits. CT 2.7 therefore investigated a joint coding of RAN and BH. As it can be seen, in Figure 4-17, the data on the BH channel is already encoded from end to end by the RAN FEC. In the joint coding approach, we therefore propose an error resilient decoder (ERD) for the RAN to deal with errors introduced by the BH link. As shown in [17], the BH channel can be modelled as a Binary Symmetric Channel (BSC). It introduces bit-flips that change the distribution of received symbols in a deterministic way. By taking this modified distribution into account, a soft decoder can be constructed that dramatically improves the end-to-end (e2e) BER when using an imperfect fronthaul. This improved decoder can be easily implemented and only requires the BER of the BH channel to be known. The improved BER can be translated into either a higher throughput, higher range of BH links or higher energy efficiency by a reduced transmit power.

4.2.4.2 Testbed Set-up

For this optimised decoder, the UE transmitter, radio access channel and iSC receiver chain until after quantization are implemented in MATLAB, while the backhaul link is provided by the 60 GHz demonstrator. Figure 4-17 highlights which parts of the CT are implemented in MATLAB and which run on the 60 GHz demonstrator. The joint coding approach can also be demonstrated live using a graphical user interface (GUI) shown in Figure 4-18.

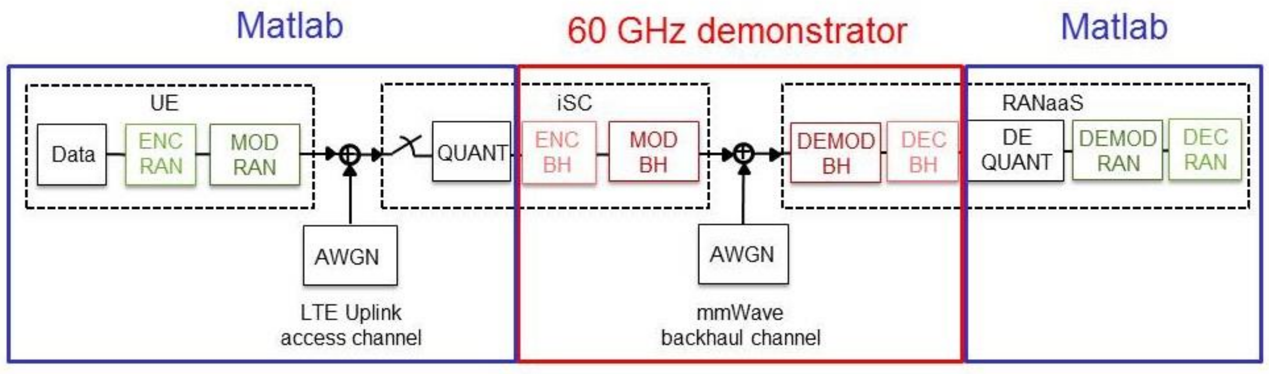


Figure 4-17: System model for mmWave backhaul and testbed implementation

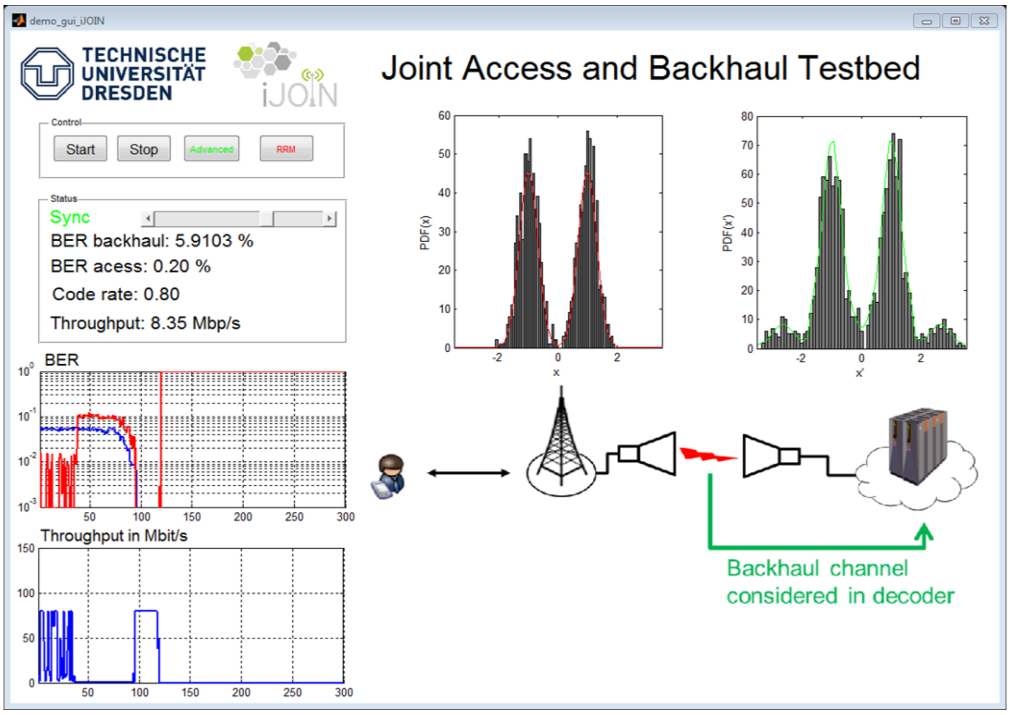


Figure 4-18: Joint access and backhaul testbed GUI

4.2.4.3 Evaluation Criteria and Performance Measures

The main performance metric for the optimised decoder is the end-to-end BER (BER_{RAN}). It is measured assuming genie knowledge of the transmitted data. We evaluate it as a function of the BER on the BH, which is a measure of the BH channel quality. From this, we can draw indirect conclusions on throughput, energy efficiency and costs.

4.2.4.4 Experimental Results

Figure 4-19 and Figure 4-20 show the end-to-end BER (BER_{RAN}) over the BER of the 60 GHz BH link (BER_{BH}) for two different RAN MCS. MCS 6 uses 4 QAM while MCS 15 uses 16 QAM. The MCS were deliberately chosen to show the performance under different modulation schemes. Each “+” sign represents the transmission of a single UDP packet and the black lines plot the average over all transmissions. Additionally, simulated results are displayed as red lines for comparison.

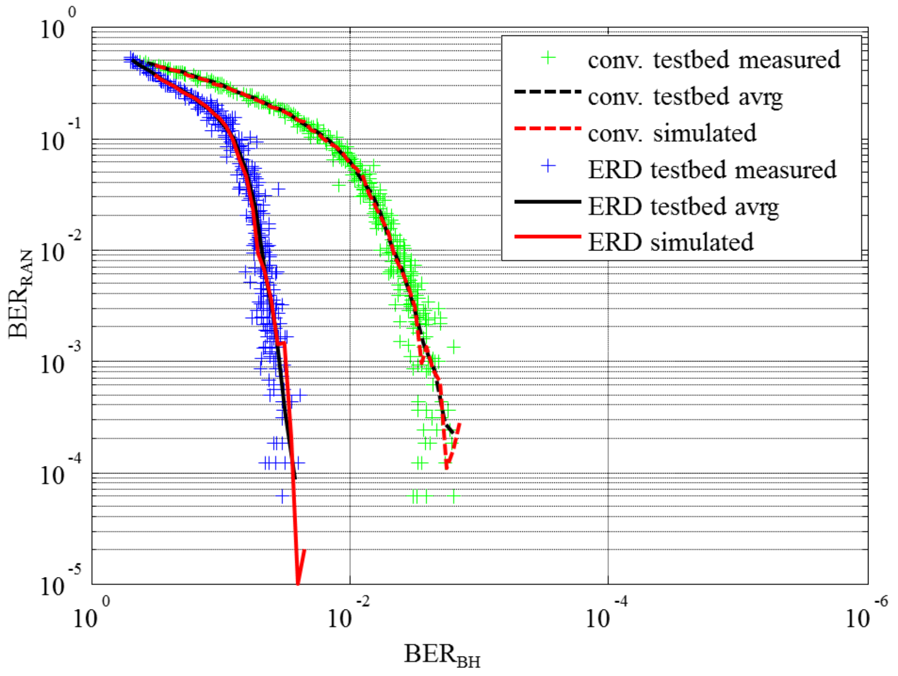


Figure 4-19: Performance of the ERD compared to a conventional decoder for MCS 6

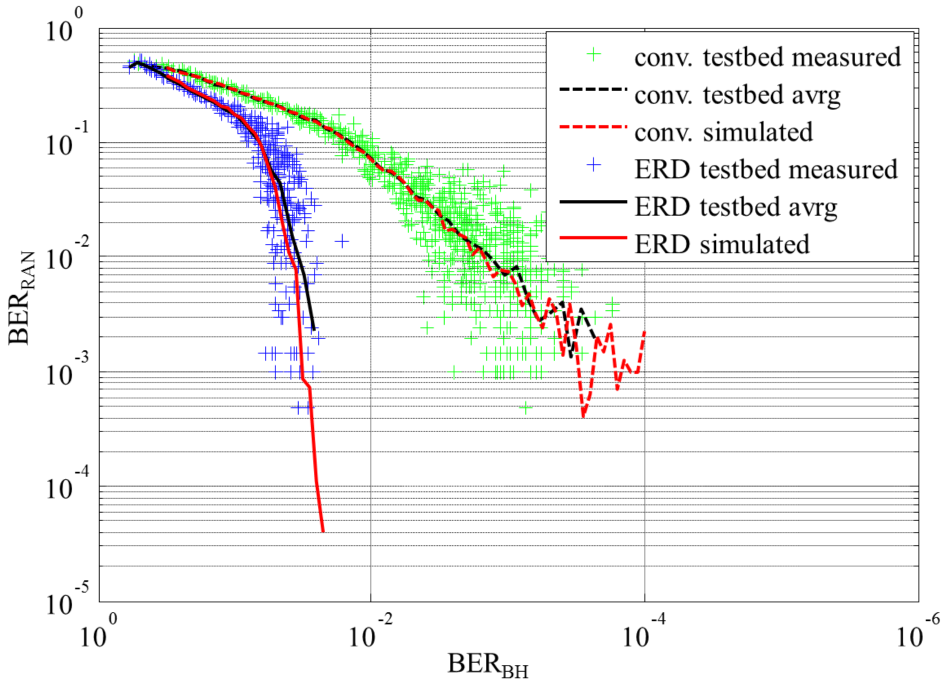


Figure 4-20: Performance of the ERD compared to a conventional decoder for MCS 15

As can be seen, the ERD achieves for both MCs a significantly better performance than the conventional decoder. Additionally, the simulated curves match the experimental results very well, indicating the validity of the results presented in D2.3. Furthermore, the end-to-end performance is already very good for BER_{BH} of more than 10^{-2}, indicating that a network could operate very well with high BER_{BH} if the ERD is employed.

The ERD achieves a gain in the BER_{BH}, i.e. it can achieve the same end to end BER at a worse channel quality. This gain can be mapped to gains either in throughput, energy efficiency or range.

The evaluation of throughput requires the measurement of the performance using a large number of different MCS similar to the throughput evaluation performed for CT2.7 in D2.3 [14]. Due to the fact that the measurements take a long time using real life hardware and that we have shown that the simulated and measured values are very much aligned, a complete throughput analysis is not performed here, as the same results as in D2.3 can be expected, where large area throughput gains are shown.

A gain in energy efficiency can be derived from the fact that the gain in BH BER can be mapped to a gain in BH SNR, which in turn can be mapped to a lower transmit energy. Using the theoretical relation between SNR and BER for 4-QAM

$$\mathrm{SNR} = 2 \cdot \left(\mathrm{erfc}^{-1}(2 \cdot \mathrm{BER})\right)^2, \tag{4.3}$$

the gain for MCS 15 corresponds to an SNR gain of 6 dB. Lowering the transmit power by that amount the resulting energy efficiency is

$$\eta_{\mathrm{BH}} = \frac{P_{BB,tx}+P_{BB,rx}+P_{RF,tx,ERD}}{D_{BH}} = 403\ \mathrm{pJ/bit}. \tag{4.4}$$

This corresponds to an energy efficiency gain of about 0.2 %. Obviously, this gain is negligible and would be even less in practice as the energy consumption of the RANaaS was not considered here. In contrast, using the link budget from Table 4-2, the gain in range is almost 100 %. Consequently, the improved decoder should be utilized to achieve higher ranges instead of limiting the transmit energy to increase energy efficiency. The higher range will lead to higher cost efficiency as less mmWave hops would be required to cover the same distance.

Table 4-2: Link budget and maximum range with and without the ERD

Parameter	Unit	Without ERD	With ERD
tx power	dBm	5	5
tx antenna gain	dBi	35	35
Max. effective isotropic radiated power (EIRP)	dBm	40	40
thermal noise (BW 4 GHz)	dBm	-78	-78
rx noise figure	dB	8	8
receiver noise	dBm	-70	-70
analog losses	dB	6	6
digital losses	dB	3	3
req SINR (4QAM)	dB	11.4	5.4
receiver sensitivity	dBm	-49.6	-55.6
rx antenna gain	dBi	35	35
rain margin	dB	3	3
oxygen attenuation (300m)	dB	4.5	4.5
max pathloss	dB	117.1	123.1
max range	**m**	**284**	**567**

4.3 Software Defined Network Testbed

The objective of the SDN testbed is to partially simulate the iJOIN functional architecture described in D5.3 [1], with particular emphasis to the networking aspects. This section describes the iJOIN candidate technology CT4.1, that is tested using the above mentioned testbed described in Section 3.3.

4.3.1 Description

The developed testbed, described in detail in Annex C, shows how SDN abstraction of data plane can be used for different management purposes as to explore smarter IP anchoring schemes. These schemes select the IP address and associated anchor used by an application based on the characteristics of the application, the available anchors in the network and the characteristics of the UE. It is assumed that multiple potential anchors are available in the network. Hence, the Packet Data Network Gateway (P-GW) is not the one anchoring all the sessions as it is currently most commonly done. The anchor can be the P-GW, an iJOIN transport node (iTN) or an iJOIN local gateway (iLGW). These available anchors are located closer to the user at the edge of the network. A given UE might be using more than one anchor simultaneously, which implies that the UE is using multiple IP addresses (for different applications). In order to provide such UE mobility, IPv6 prefix continuity [18] mechanism is employed. This procedure is commonly called Distributed Mobility Management (DMM).

Figure 4-21 shows an example of a potential scenario. A UE is initially attached to an eNB and has traffic (green line) anchored at iLGW for flows directed to local services (left subfigure). In the right subfigure, the UE gets attached to a second RANaaS. The green flow is still anchored to the initial anchor while a new anchor is selected for new flows. A P-GW is selected for new flows directed outside the RANaaS (red line). The path between the UE and the respective anchors is optimally computed by a path management and routing mechanism.

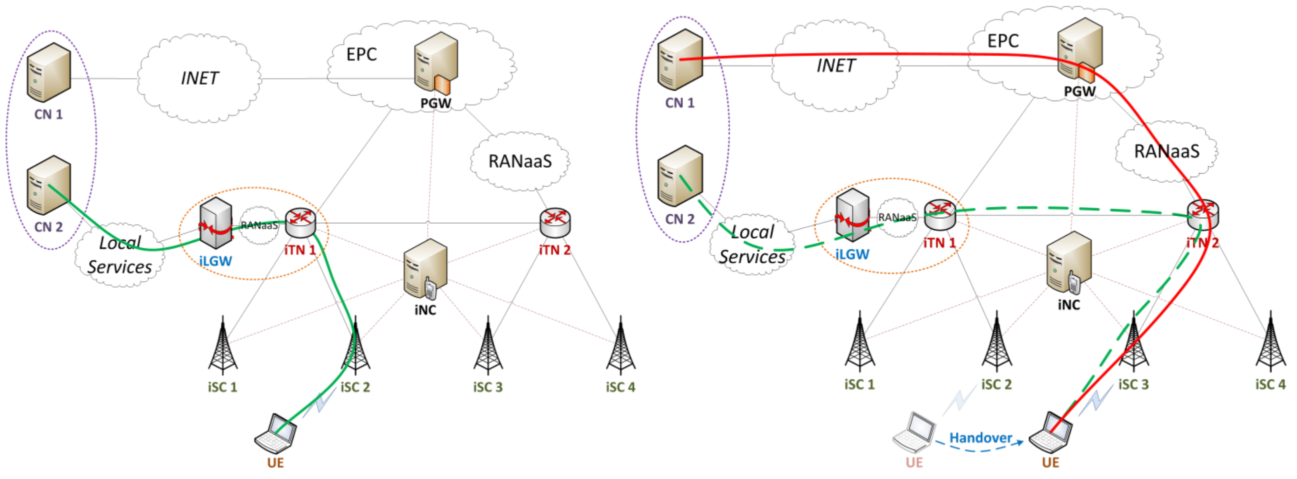


Figure 4-21: Distributed IP anchoring and mobility management demonstrator

To demonstrate this network management functionality iJOIN made use of the testbed flexibility creating a specific topology by configuring the virtual local area networks (VLANs) in a convenient manner thanks to Linux capabilities to create VLAN interfaces and 802.1Q switch forwarding as explained in Annex C.
The whole demonstration is IPv6 based and the UE Node is an IPv6 based node. Once network topology and routing mechanisms are established, algorithms required for this demonstration were implemented as applications in the iNC. A modular structure has been adopted and, accordingly to Ryu [28], an event based system has been used to intercommunicate between different modules. Modules implemented for this demonstration are described below and a map of connection is represented in Figure 4-22.

- Anchor and Mobility Management (AMM): manages the mobility of the UEs attached to the network. When a UE moves from a point of attachment to another, AMM runs different algorithms to select the optimal anchor and configure the new path in the network sending an event with the new path. We refer to D4.2 [19] for the algorithms and the interaction required in the iNC.
- Mobility Management Entity (MME): keeps the connection status of each UE in a database.
- IPv6: performs the standard IPv6 neighbor discovery procedures.

- Traffic Engineering Enforcement Module (TEEM): envisioned action for this module is computation of the best path within the backhaul to support the different traffic and network-wide requirements, providing the necessary conflict resolution functions. By now, we deployed a basic TEEM implementation covering the CT4.3 algorithm implementation.
- Gateway and Radio Iface: configure the gateway and radio functionalities on capable nodes.
- Network Model Module (NMM): builds an abstract resource model of the network and generates an event for the interested modules with the information about topology changes.
- Link Layer Discovery Protocol (LLDP): discovers the network topology using the LLDP protocol [20].
- GUI: represent dynamically the state of the network, including elements that conforms it and statistics about flows and anchors for every link. Figure 4-23 and Figure 4-24 show a screenshot of the implemented GUI.

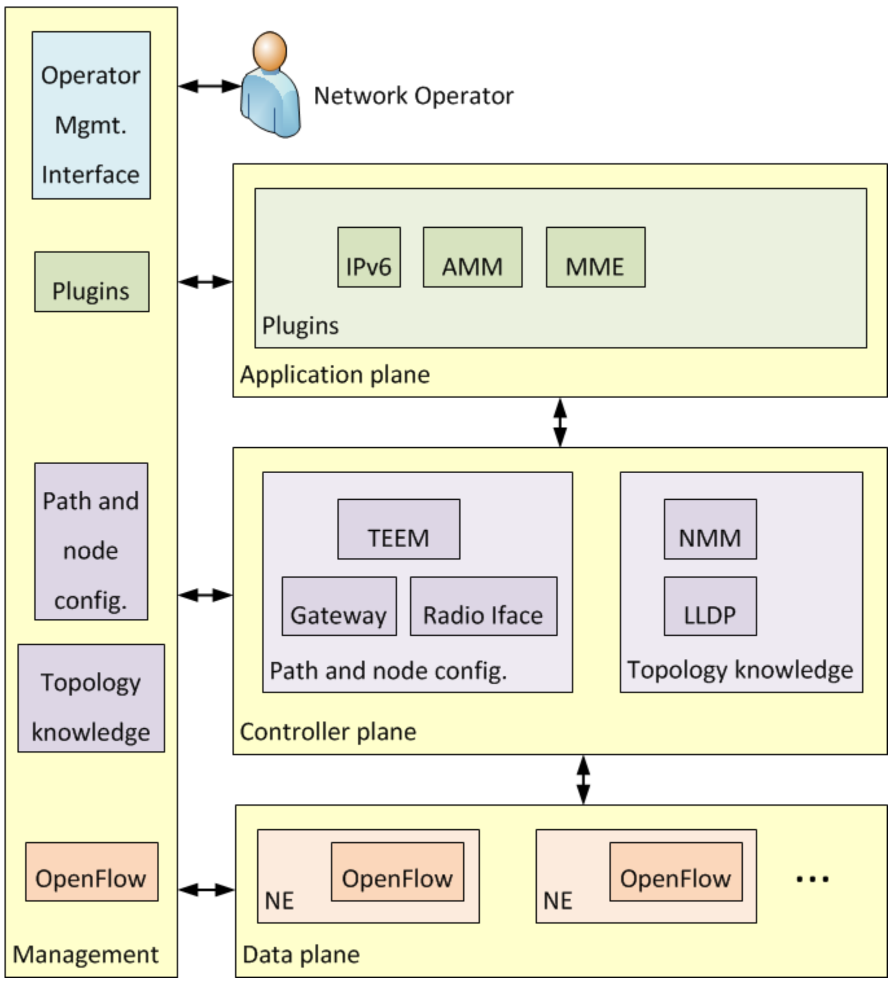


Figure 4-22: CT4.1 SDN modules

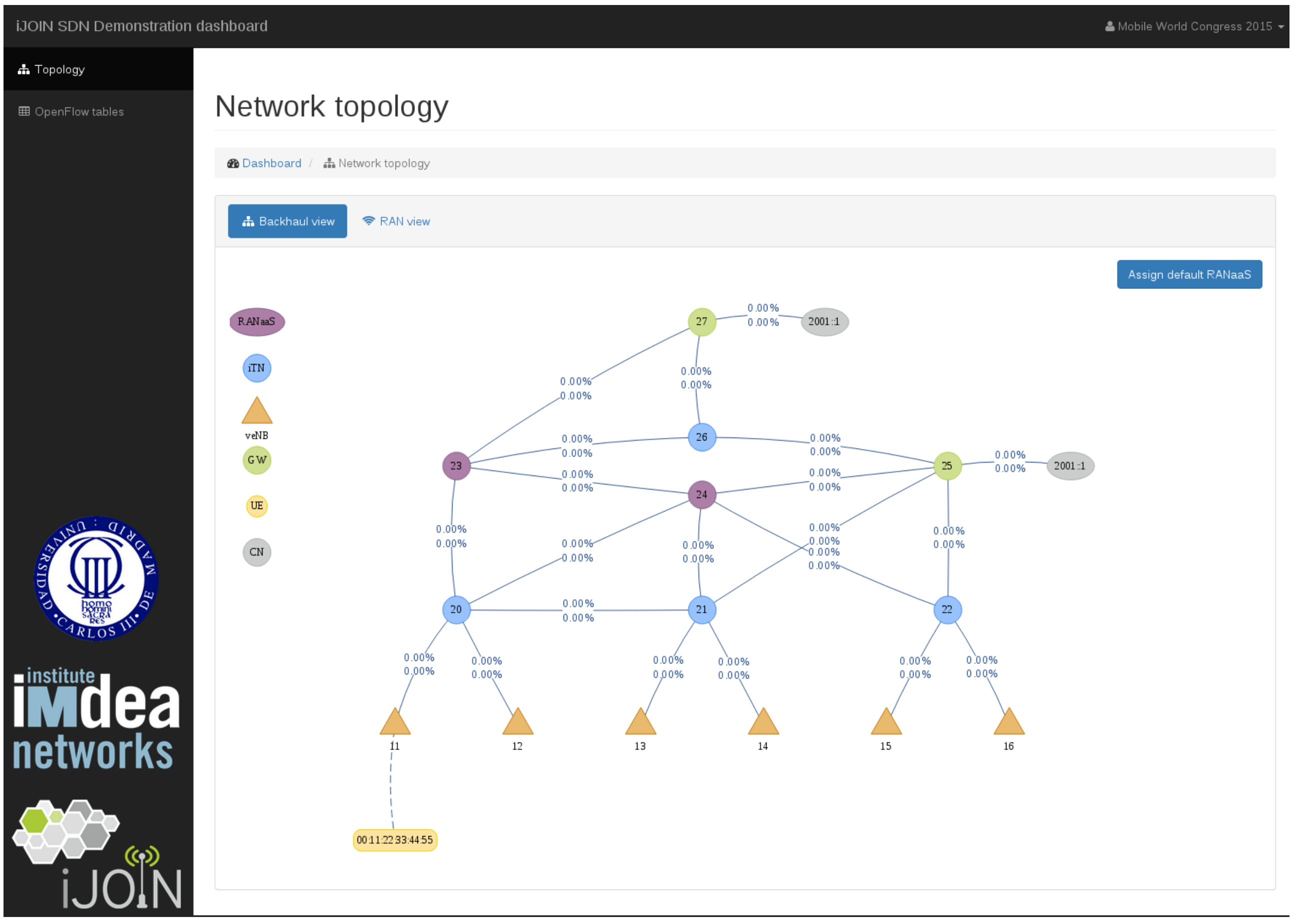
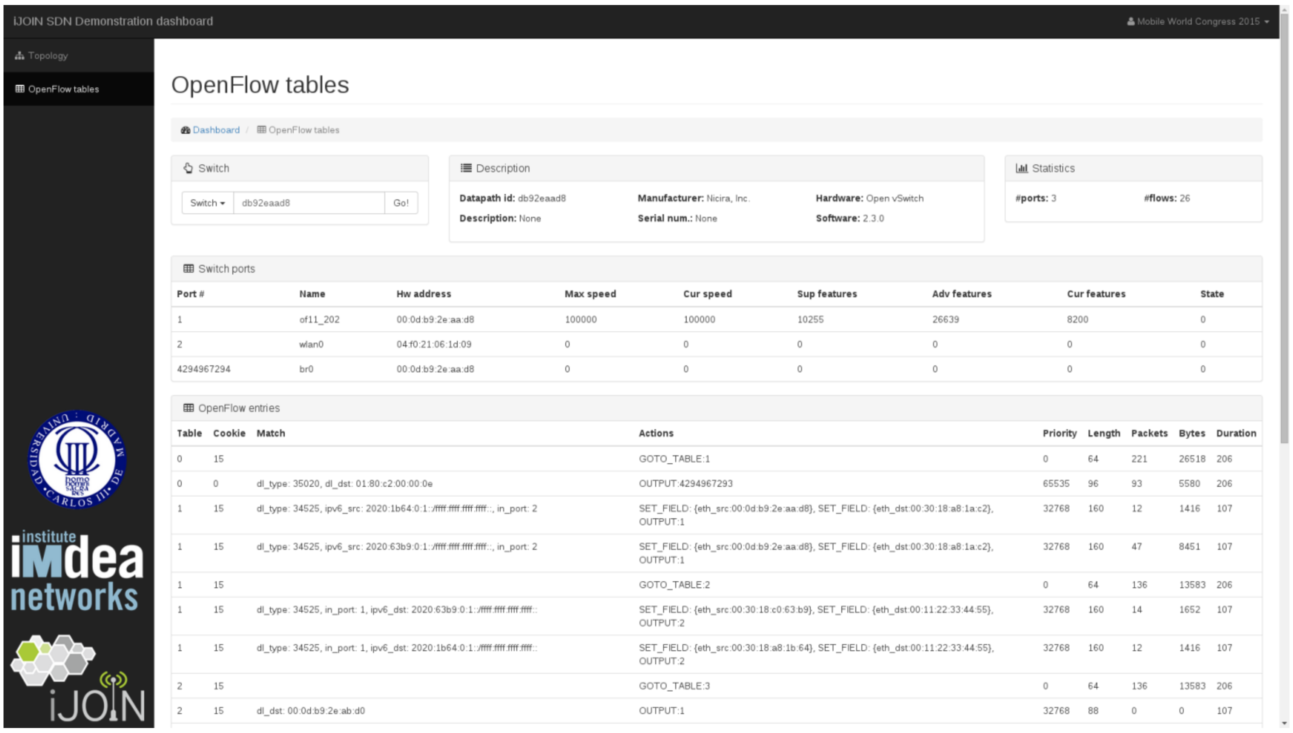


Figure 4-23: Network controller real time view of the network


iJOIN SDN Demonstration dashboard
Mobile World Congress 2015
Topology
OpenFlow tables
Dashboard / OpenFlow tables
Switch
Go!

OpenFlow tables

Switch: db92eaad8

Description

Datapath id: db92eaad8 **Manufacturer:** Nicira, Inc. **Hardware:** Open vSwitch
Description: None **Serial num.:** None **Software:** 2.3.0

Statistics

#ports: 3 **#flows:** 26

Switch ports

Port #	Name	Hw address	Max speed	Cur speed	Sup features	Adv features	Cur features	State
1	of11_202	00:0d:b9:2e:aa:d8	100000	100000	10255	26639	8200	0
2	wlan0	04:f0:21:06:1d:09	0	0	0	0	0	0
4294967294	br0	00:0d:b9:2e:aa:d8	0	0	0	0	0	0

OpenFlow entries

Table	Cookie	Match	Actions	Priority	Length	Packets	Bytes	Duration
0	15		GOTO_TABLE:1	0	64	221	26518	206
0	0	dl_type: 35020, dl_dst: 01:80:c2:00:00:0e	OUTPUT:4294967293	65535	96	93	5580	206
1	15	dl_type: 34525, ipv6_src: 2020:1b64:0:1::/ffff:ffff:ffff:ffff::, in_port: 2	SET_FIELD: {eth_src:00:0d:b9:2e:aa:d8}, SET_FIELD: {eth_dst:00:30:18:a8:1a:c2}, OUTPUT:1	32768	160	12	1416	107
1	15	dl_type: 34525, ipv6_src: 2020:63b9:0:1::/ffff:ffff:ffff:ffff::, in_port: 2	SET_FIELD: {eth_src:00:0d:b9:2e:aa:d8}, SET_FIELD: {eth_dst:00:30:18:a8:1a:c2}, OUTPUT:1	32768	160	47	8451	107
1	15		GOTO_TABLE:2	0	64	136	13583	206
1	15	dl_type: 34525, in_port: 1, ipv6_dst: 2020:63b9:0:1::/ffff:ffff:ffff:ffff::	SET_FIELD: {eth_src:00:30:18:c0:63:b9}, SET_FIELD: {eth_dst:00:11:22:33:44:55}, OUTPUT:2	32768	160	14	1652	107
1	15	dl_type: 34525, in_port: 1, ipv6_dst: 2020:1b64:0:1::/ffff:ffff:ffff:ffff::	SET_FIELD: {eth_src:00:30:18:a8:1b:64}, SET_FIELD: {eth_dst:00:11:22:33:44:55}, OUTPUT:2	32768	160	12	1416	107
2	15		GOTO_TABLE:3	0	64	136	13583	206
2	15	dl_dst: 00:0d:b9:2e:ab:d0	OUTPUT:1	32768	88	0	0	107

Figure 4-24: Network controller real time view of the installed OpenFlow rules

4.3.2 Testbed Set-up

The role adopted by each device of the SDN testbed is displayed in Figure 4-25 and the VLANs have been configured in order to emulate the topology depicted in Figure 4-216. The UE is not reported in the figures since it does not require any configuration.

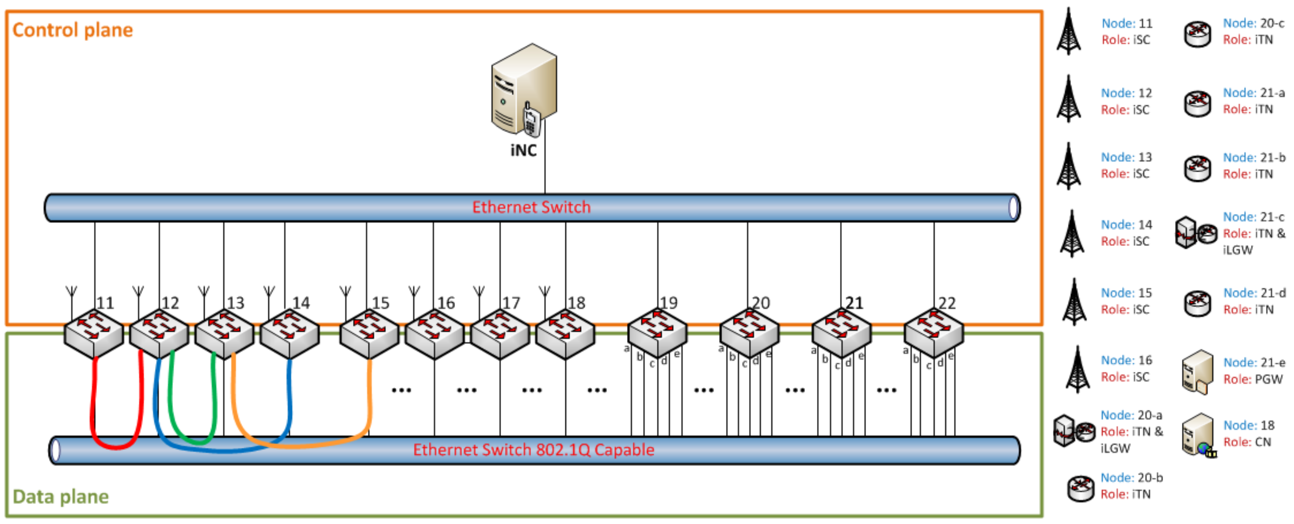


Figure 4-25: SDN devices role

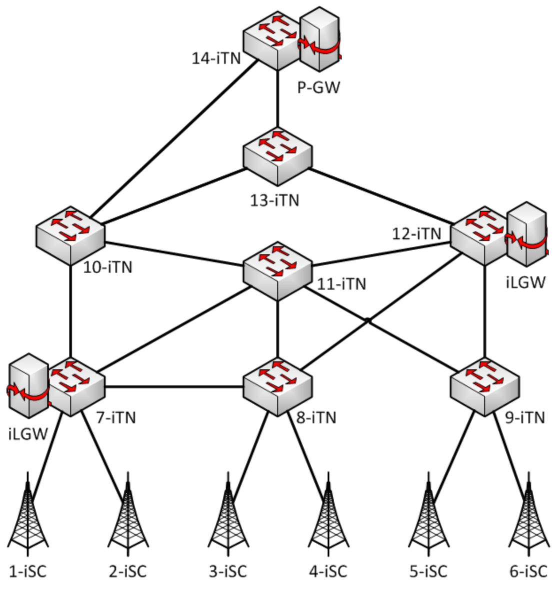


Figure 4-26: Emulated topology

We next summarize some of the functional aspects that are shown using the SDN testbed:

- Initial bootstrapping of the network: it involves the discovery of the network topology, the reporting of the available capabilities and the set-up of the logical network database at the iNC;
- Set-up of monitoring and measurements: it involves the configuration from the Anchor and Mobility Management AMM [19] modules of the required measurements and its execution by the AMM module at the iNC;
- UE initial attachment to the network: it involves the detection of the attachment by the iSC and the notification to the AMM module at the iNC;
- Initial selection of the anchor by the AMM module: both the selection of a centralised anchor (i.e., PGW) or a localised anchor (i.e., an iLGW) are evaluated;
- Set-up of a path within the backhaul network and RAN in cooperation of the AMM and the TEEM modules. For each application, a different path is set up for each application if required/possible;
- UE handover, moving from the current point of attachment to another one. Different types of handovers are evaluated including the selection of a new anchor.

4.3.3 Evaluation Criteria and Performance Measures

In addition to the functional evaluation, the performance of some procedures is assessed as well. Namely, at least the following metrics will be considered:

- Handover latency: time after a UE decides to move until the handover is completed and the UE can resume its communications. Both types of handover are evaluated, i.e. involving selection of a new anchor as well as without re-selection.
- Anchor selection latency: time taken by the network to select an anchor after a Packet Data Network (PDN) connection request is initiated by the UE. This time is measured for different network sizes.
- Signaling load per network event (bootstrapping, UE attachment, anchor selection, handover).
- Utilisation efficiency: offloading of SDN Testbed links with different RANaaS placement and different gateway configurations.

4.3.4 Experimental Results

First of all, functional aspects of the SDN testbed were successfully tested. In order to evaluate performance measures some experiments have been conducted. In the following we will detail the experimental results obtained.

This first experiment presents a Cumulative Distribution Function (CDF) of handover latency where 190 handovers for a UE with 2 anchors were performed evaluating layer 2 and 3 handovers and ping disconnectivity, understanding ping disconnectivity as the time when no ping echo request were answered. CDF curves are presented in Figure 4-27. Regarding the results, we discovered that they are stable, time fluctuations are not high, making our solution reliable in terms of handover time. This variance can be fixed instead of having standard Linux Kernel with the default scheduler, by making use of a pre-emptive kernel in order to assure deterministic behaviour of the application, removing OS-dependency, which is a future improvement for this testbed.

Furthermore, some absolute values for 95% percentile are:

- 95% percentile total layer 2 Hand-Over (HO) time: 18ms
- 95% percentile total layer 2+3 HO time: 44ms
- 95% percentile for ping disconnectivity: 51ms

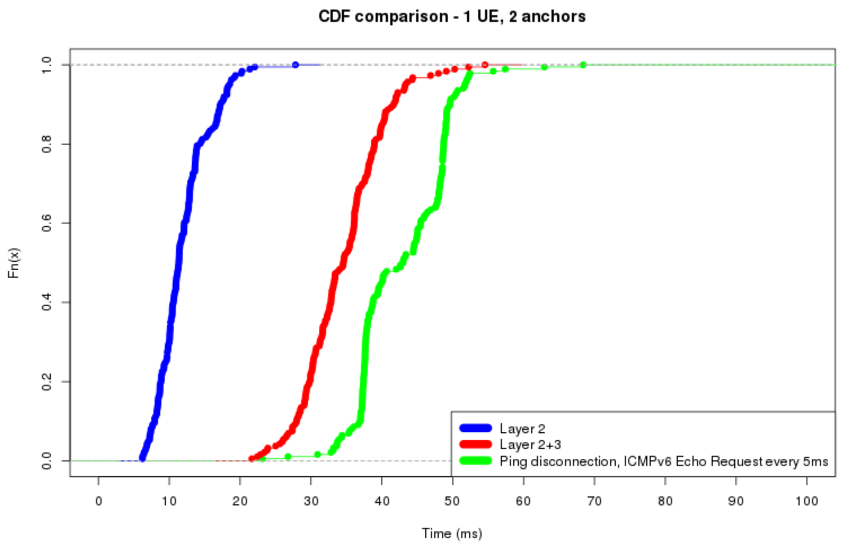


Figure 4-27: CDF for layer 2, layer 3 handovers and ping disconnectivity

One of the aspects also evaluated is the influence of the number of anchors in handover time. A time graph obtained for handover regarding 1 and 2 anchors is presented in Figure 4-28. According to data presented, we observe equidistance between peaks for increasing anchors. The main difference introduced by the increase of anchors comes in the variance of results for different experiment execution. We observed a higher variance in samples where the number of anchors is higher, occurring only in few instances of the experiment. Once again we can remove the OS-dependency of this fluctuation using a pre-emptive kernel.

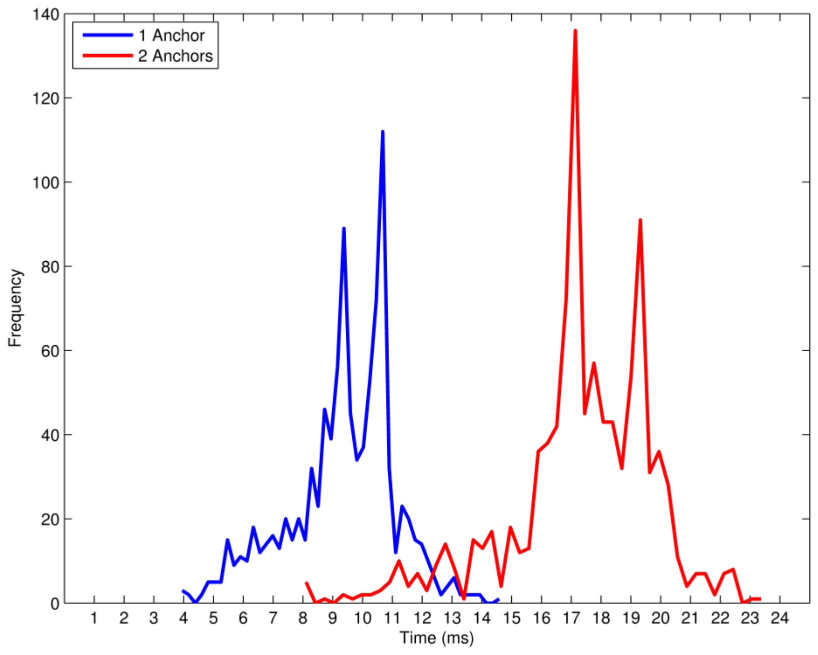


Figure 4-28: Handover time distribution for multiple anchors

Concerning layer 3 handover, a deep analysis has been performed, exploring which operations are generating this handover time and which one affects more drastically to the total handover time along different number of anchors. A breakdown of layer 3 HO time was designed and results are displayed in Figure 4-29. Configuration path computed by TEEM is the operation consuming the highest amount of time and is increasing linearly with the number of anchors. Note that currently the path is computed and configured by demand when the UE connects to the network. Better results could be potentially be achieve using the advanced algorithms investigated by CT4.3. Therefore, the performance is closely tied to TEEM performance. If TEEM is able to preconfigure the path through VLAN's aggregation in the backhaul network, this handover time will become deterministic since the amount of OpenFlow rules to be written for each anchor is constant. That is, TEEM does not configure all the nodes along the path but configures only the iSC and the anchor nodes.
IPv6 UE advertisement performs the standard IPv6 advertisement procedures and is the second hardest operation regarding results. The rest of the tasks such as anchor selection, path computation and association management represent less than 1 ms.

In addition to all these procedures, the UE requires a time to perform the Link Layer handover. In total, the UE suffers a disconnection from the network of 51ms. This value is in line with the 50ms recovery time defined by ITU-T G.783 [21] and Bellcore GR-253 [22]. When a path failure is detected in the network, 50ms is the maximum allowed time for reconfiguring the paths in the network without affecting the connections. Therefore, the UE QoE is not affected by the handover.

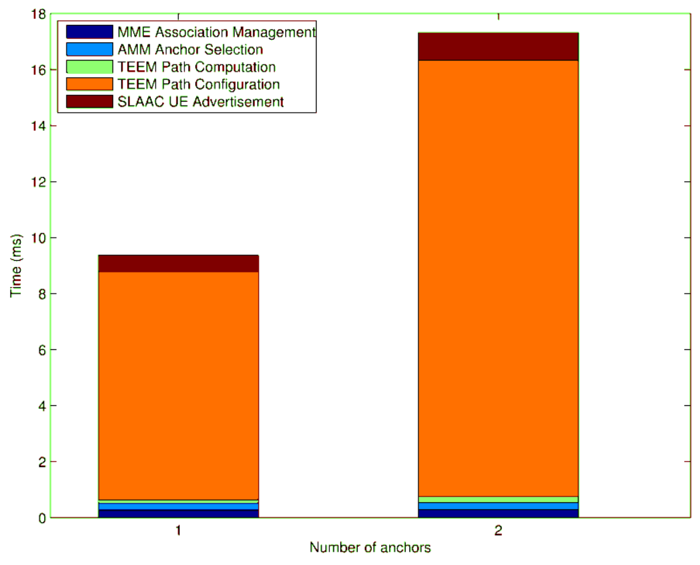


Figure 4-29: Breakdown of layer 3 handover time

Lastly, another experiment was performed to evaluate the impact of the handover procedure on the TCP (transmission control protocol) congestion window. This result is relevant as it directly affects the QoE perceived by the UE. To evaluate this aspect a TCP stream was created and the packets captured. The evolution of TCP sequence numbers over the time has been plotted in Figure 4-30 which reports the impact of multiple handovers on a TCP stream of 60 seconds. A handover every 5 seconds was manually triggered in a round robin fashion on 4 different Points-of-Attachment statically selected. The figure shows how the handover impact is relatively small.

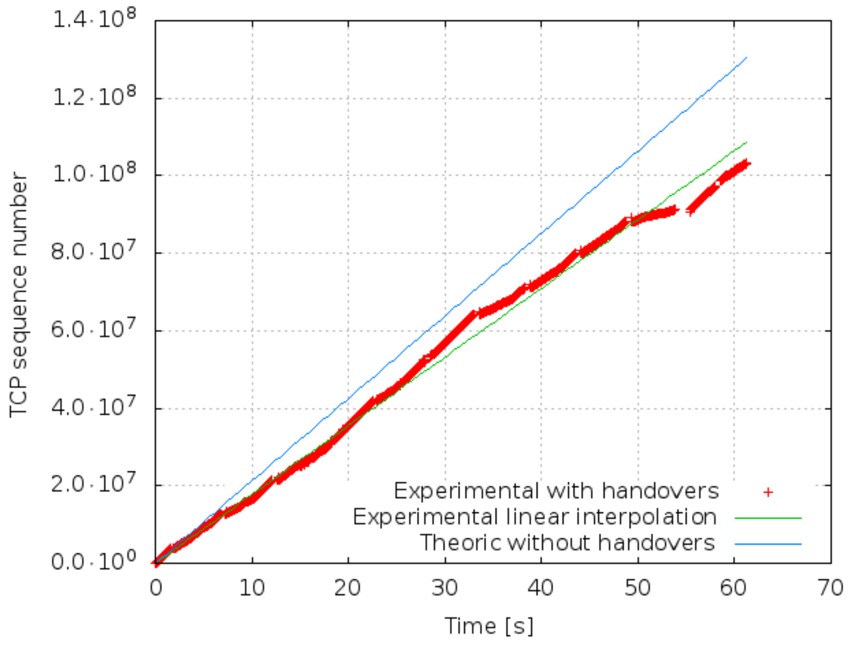


Figure 4-30: Handover impact on TCP streams

Regarding the CT4.1 Utilisation Efficiency assessment, we performed a dedicated experiment. We configured the SDN test-bed according to the RANaaS placement results reported in D4.3 [2]. The results contemplate different RANaaS location depending on the functional split being employed. The evaluated topology (Figure 4-26) reproduces the wide-area scenario depicted in D5.2 [16]. In addition, we consider 4 sub-scenarios where iLGWs are independently switched-on or off. That is, the P-GW is always available, while iLGW at 7-iTN and 12-iTN are enabled/disabled covering all the combinatory possibilities.

In order to assess such scenario, we employ one UE which connects to the test-bed and generates the amount of traffic reported in D5.2 (10-15 Mbps in peak hour). Since the test-bed contemplates 6 iSCs (wide-area scenario envisages 19 iSCs), and the maximum link capacity is 1Gbps and 54Mbps in the backhaul and radio links respectively, we proceed in the following way:

1. We attach the UE to a iSC and we generate the considered amount of traffic during 1 minute;
2. We collect the amount of traffic passed through each link during this interval.

Once we collected the results, we attach the UE to a different iSC and we repeat the measurement. We assume that the UEs' traffic is independent, that is, there is no correlation between the traffic generated in two separate time intervals. Doing so, we find the total link occupation by summing the partial measurement.

Figure 4-31 shows the results for different gateway availability and RANaaS placement. We consider as utilisation efficiency gain the percentage of the total traffic offloaded from the network against the baseline. As it can be noticed, in case of employing both iLGWs we offload the 75% of traffic. This result is valid for each assessed configuration. In case of employing an iLGW at 12-iTN we achieve the minimum network offloading of 46%, while we achieve a higher offloading in case of iLGW at 7-iTN (60%).

Concluding, the results show that CT4.1 can efficiently operate on any functional split. Depending on the functional split, the RANaaS placement may vary and so the achievable gain. The results also highlight also that an iLGW closer to the UEs is more efficient than a farther one. Therefore, we believe that whenever a RANaaS is deployed, the RANaaS should be provided with iLGWs functionalities in order to offload as much as possible the network. We refer to D4.3 [2] for any additional result, consideration and conclusion on CT4.1 Utilisation Efficiency.

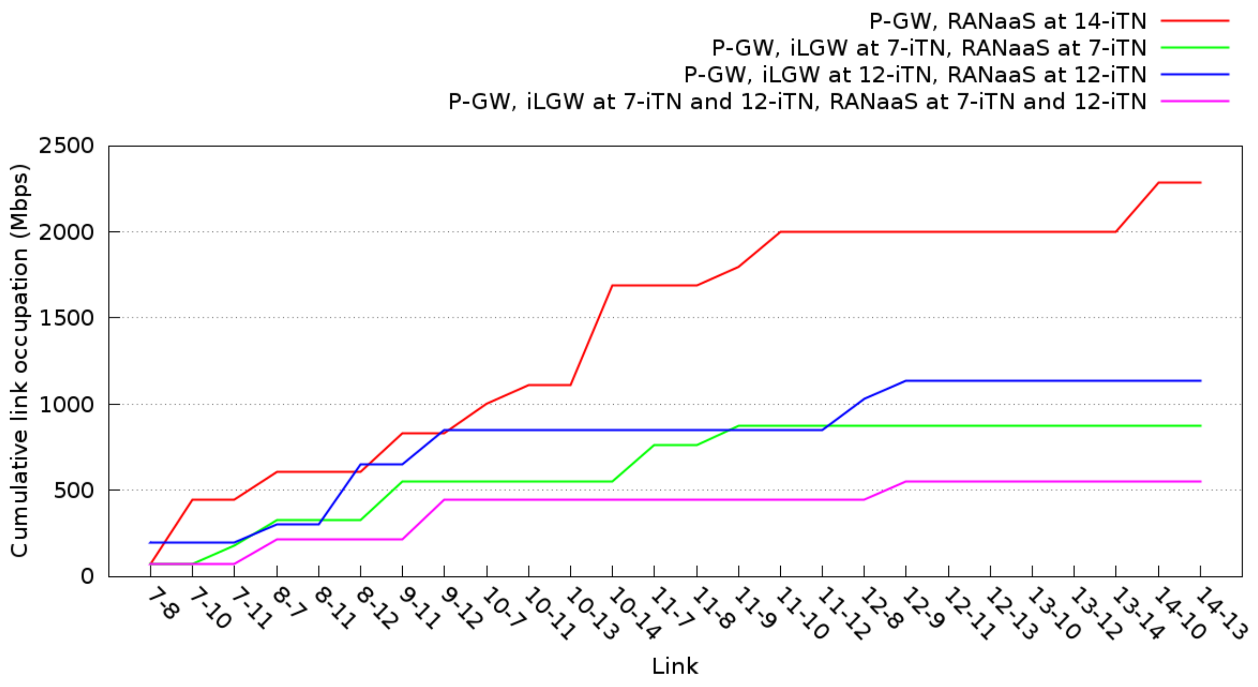


Figure 4-31: Cumulative link occupation of the network

5 Achievability of iJOIN Concepts

This section discusses the achievability of iJOIN concepts with respect to the theoretical advancements made in technical work-packages WP2, 3 and 4, the integration in WP5, and the experimental results obtained in WP6 by running technical proposals over the testbeds developed for iJOIN. The following subsections treat separately these ideas for the three iJOIN testbeds.

5.1 RANaaS

WP2 investigated and assessed the possibility to centralise certain PHY processing blocks in a RANaaS based on general purpose server hardware. Turbo decoding has been identified as a relatively expensive and non-deterministic building block of the RAN protocol stack. Therefore, its RANaaS implementation is particularly challenging and most severely impacted by the 3GPP timing constraints. The results obtained in Section 4.1.1 point out that significant effort need to be put into the implementation on the RANaaS platform in order to overcome these restrictions. Additionally, a modification and improvement of the virtualised OS or the underlying hypervisor is required. Furthermore, iJOIN developed technologies to predict and control the number of turbo iterations which determine the required decoding time. This is an essential building block to control the computational load as well as real time constraints in a RANaaS platform.

The control of the decoding time is based on the computational complexity framework described in [16] and [10]. The framework has been verified experimentally in Section 4.1. Furthermore, CT 3.6 introduced a joint RAN and cloud scheduling which allows for controlling the computational load which has been verified by the RANaaS testbed as well.

5.2 Joint Access and Backhaul

The joint access and backhaul testbed has been set up to provide a mmWave hardware-in-the-loop platform which is used to evaluate the feasibility of mmWave backhaul and to characterize the influence of the backhaul channel on centralised algorithms. As described in previous deliverables and Annex B, the platform has been set up successfully. An Ethernet interface and MATLAB API were provided to enable the implementation of different CTs using the testbed as hardware-in-the-loop.

Testbed experiments showed that mmWave technology is well suited as a “last-mile” backhaul technology. Link budget calculations show that mmWave links can achieve ranges of a few hundred meters at a capacity of a multiple Gbps, which is sufficient even for low layer functional splits that are discussed in D5.3 [1] and D2.3 [14]. Although the capacity limits the number of iSCs that can use the same link, in particular for lower functional splits, they are still a low-cost and simple solution if a fibre-equipped iTN can be reached within a few hops.

Latency measurements have shown that the requirements even of the lowest splits can be met by mmWave technology. The same measurements show that the main part of the latency is introduced not by the actual mmWave hardware but by the necessary MAC protocols. This indicates that if very short latencies are required, i.e. to enable a high degree of centralisation, the number of switches and iTNs between UEs and RANaaS has to be kept low. Hence, the reduction of the iTN latency should be prioritised over reducing the actual hardware latency.

Two schemes have been evaluated that require an extensive use of BH links. It has been shown that they require varying degrees of reliability in terms of BER. While the DiCE algorithm requires a very low BER of about 10^{-7} for forwarding user signals, the hierarchical precoding approach can deal with a BER of about 10^{-3} as it forwards CSI information. This indicates that no unified requirement in terms of reliability applies. To keep the overhead in terms of error protection coding on the BH low, reliability should be adopted to the necessary level, meaning that different channels (e.g., data and control) should be encoded separately, each one using only the required robustness. This approach is similar to the idea of a separation of control and user plane discussed in D3.3 [9].

To mitigate the imperfect nature of wireless channels while keeping the required BH modules simple and the BH latency low, CT 2.7 developed innovative algorithms that promise substantial throughput gains. These results have been verified using the testbed. As demonstrated in Section 4.2.4, the approach of joint coding for access and backhaul improves performance significantly. In addition, it was shown that the experimental results match the simulated results very well, indicating the validity of the evaluations in D2.3 [14].

5.3 SDN

The SDN testbed has been set up to evaluate the feasibility of the designed mobility protocol and to characterize the components of the SDN controller. As described in previous deliverables and Annex C, the platform has been set up successfully. In addition to the Ryu API [29], a REST (REpresentational State Transfer) API has been implemented in order to create new applications able to operate on the network. Indeed, REST is a software architecture style consisting of guidelines and best practices for creating scalable web services. REST is a coordinated set of constraints applied to the design of components in a distributed system that can lead to a more performant and maintainable architecture.

The investigation on SDN-based network management indicates that advanced services can be deployed in the network in a cost effective manner. On the one hand, the time required to implement new services, or simply to change the network policy, is drastically reduced. Therefore, SDN offers the capability to promptly adapt the existing network to new requirements in terms of offered services. Results obtained with the SDN-testbed indicate that SDN can be used to react promptly to external events in order to assure the correct functioning of the network. We demonstrated how this can be achieved on low power commodity hardware. The results for CT4.1, illustrated in Section 4.3, clearly show how the implemented mobility protocol does not affect the UE’s quality of experience and how backhaul network traffic can be offloaded to avoid an over-provisioning of the network.

The implementation of the prototype brought upraised several challenges belonging to classical distributed systems. This is due to the needs of having a resilient, robust and a high availability network controller. A careful software design has been adopted for the controller in order to cope with such requirements, but in order to prove the real scalability of the system, a stress-test should be performed. A stress-test with a high number of UEs is not feasible in iJOIN’s SDN-Testbed but we believe that the adopted software design is a good starting point for implementing a product-ready network controller applicable to an iJOIN system.

The SDN tesbed implementation has been successfully shown at *European Conference on Networks and Communications 2014* (EuCNC2014, Bologna) and *Mobile World Congress 2015* (MWC2015, Barcelona) as detailed in [D7.2, Section 4.4]. In particular, SDN testbed was hosted in the iJOIN demo booth at EuCNC2014 and in the European Commission booth at MWC2015.

A demonstration, called *SDN-based Mobility Management in a Dense Small Cells scenario*, was shown at EuCNC 2014. The demonstration showed how by using an SDN-based Mobility Management a dynamic anchoring is possible. A different anchor selection can be made per user-basis or even per flow-basis. This means that different user flows can be anchored to different points of the network enabling a more fine-grained and ad-hoc mobility support. In addition, this approach allows to reassign an anchor already assigned making possible a dynamic reconfiguration of the network whenever is necessary.

A demonstration, called *Novel network management algorithms*, was shown at MWC2015. The RANaaS concept provides an optimal split of the RAN functions between the mobile network and the cloud taking into account cloud-computing and mobile network parameters. This requires novel network management algorithms capable of efficiently taking into account the constraints from Cloud-RAN, transport network, and radio access network. For instance, if the Cloud-RAN is congested, an inter-Cloud-RAN load-balancing is initiated, which is done by means of mobility techniques. This is done by handing over the mobile terminals to different data centres. This process may trigger a different functional split configuration and also involves additional use cases related to mobility, such as the creation of a new flow or a real handover to a new base station.

6 Summary and Conclusions

This document is the last deliverable of WP6 which describes the proof-of-concept work within iJOIN. The following three testbeds have been implemented in WP6 in order to demonstrate different concepts proposed in iJOIN:

- Radio Access Network as a Service
- Joint Access and Backhaul
- Software Defined Network

This document described briefly these three testbeds. A detailed description of each testbed, including concept and interfaces, has been updated with respect to the last deliverable and reproduced in the annexes in order to make this document self-contained. This detailed description should enable the reader to interact with the testbeds through the available interfaces specific to each testbed.

The key contributions of this document are the final experimental results obtained by running different technology proposals over the three testbeds. These results are the final outcome of the demonstration activities conducted in WP6 and become the proof-of-concept of iJOIN. The experimental results provided in this document are promising. They clearly validate the ideas of an iJOIN system built upon small cells and a functionality split between the iJOIN small cells and the RANaaS platform. The RANaaS testbed has shown the convincing results for the technical scenarios where RAN functions are provided as a service on demand. The mmWave backhaul testbed has been used to generate results showing joint access and backhaul optimisation where the backhaul exchanges take place over real 60 GHz links. This has validated the iJOIN ideas related to joint optimisation of RAN and backhaul. The SDN testbed has clearly demonstrated through experimental results the management of an iJOIN system following the SDN approach separating the control and data plane. Overall, the results clearly indicate the potential of the ideas proposed within iJOIN and confirm the possible evolution of future cellular systems to such a system.

Acknowledgements and Disclaimer

This work was partially funded by the European Commission within the 7th Framework Program in the context of the ICT project iJOIN (Grant Agreement No. 317941).

Annex A RANaaS Testbed Description and Interface

A.1 Concept

The purpose of RANaaS testbed is to provide a system for simulating the RANaaS platform, as identified in the iJOIN logical architecture.

The testbed is based on a cloud Infrastructure as a Service platform providing basic information technology (IT) capabilities like:

- **computation services**: ability to start Virtual Machines (VMs) running an operating system and application software;
- **storage services**: ability to create storage elements (i.e. disk volumes) that can be attached to VMs;
- **networking services**: ability to create network objects like Level 2 (L2) networks, subnets, DHCP services, etc., used for connecting VMs.

In information technology terminology, when higher level services (i.e. software applications) are deployed in a cloud environment, IaaS platform is at the basis of the solution. The following figure summarizes the concept:

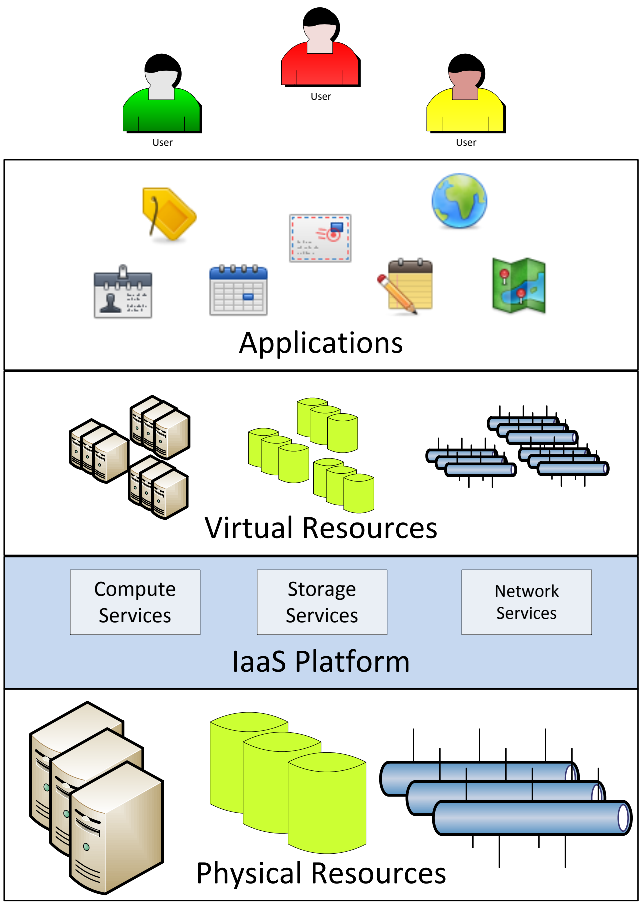


Figure 6-1: IaaS cloud platform

The layer at the bottom of the stack represents the physical resources like physical machines (computing resources), physical disks (storage resources) and physical networks (connection resources).

The IaaS platform is a software layer implementing services for dynamically creating virtual resources (virtual machines, virtual disks and virtual networks) that can be used instead of their physical counterparts in order to deploy and run applications. Virtual resources are provided as services; this means that they are created when needed, used to run applications and removed when the application is not anymore needed. The actual computation happens at the physical level but, because of the presence of the IaaS platform that provides virtual resources, physical resources and applications are not tightly bound together. In this manner, the physical infrastructure can be used for different purposes at different times.

The combination of the IaaS platform and the underlying physical resources is defined as a **cloud**. In general, a **private cloud** is managed by a single organization that is also the owner of the applications running in the cloud computing platform.

In iJOIN, IaaS cloud platform hosts iRPU functions as described by the following picture:

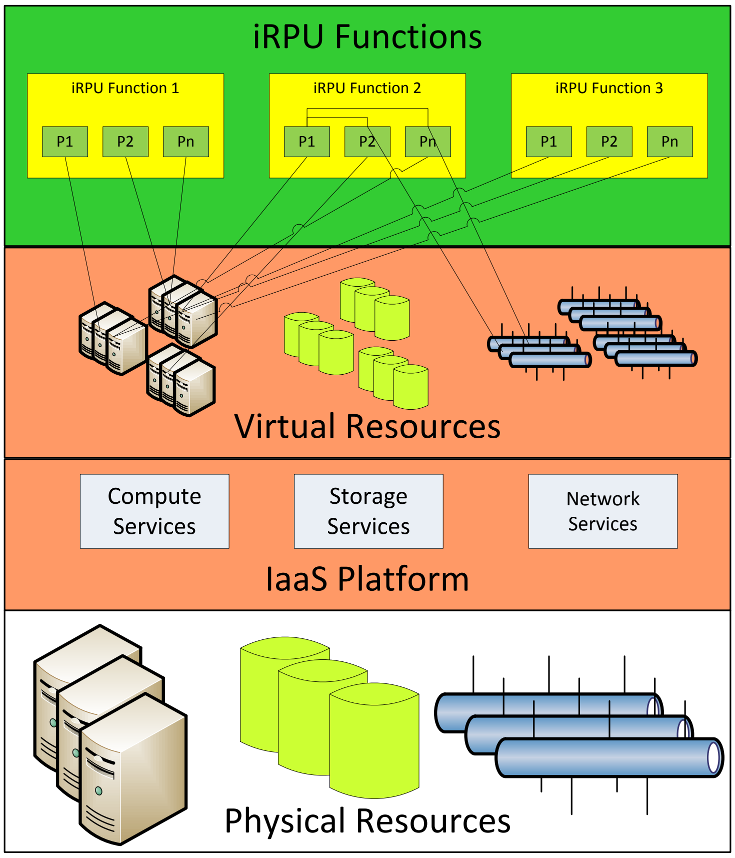


Figure 6-2: RANaaS testbed

They are implemented as computing programs running on top of virtual machines hosted by the RANaaS platform,can communicate via virtual networks and store data on virtual volumes, if needed. In this perspective, virtual objects (e.g. VMs, virtual network and virtual disks) are 'technology enablers' made available by IaaS for implementing the RANaaS. For testing purposes, some VMs simulate iSC nodes.

A.2 Description

OpenStack [4], a free and open-source software cloud computing software platform, is the IaaS system selected for the implementation of the RANaaS testbed. In iJOIN two physical testbed deployments have been implemented: one is hosted at the University of Bremen (UoB), the other at the Telecom Italia (TI) site in Turin.

The next section provides a brief introduction to the OpenStack software architecture by describing the most relevant modules and some indications on a typical OpenStack basic hardware deployment. This information is provided as an introduction for understanding the implementation of the two iJOIN RANaaS testbed instances, as described in the following sections.

A.2.1 OpenStack Software Architecture

OpenStack's architecture can be described providing the following views:

- **logical architecture**: it shows the components that make up the systems and the relationships among them;

- **deployment architecture**: it illustrates how the components are physically installed (deployed) on the hardware infrastructure.

The following picture shows the OpenStack high-level logical architecture:

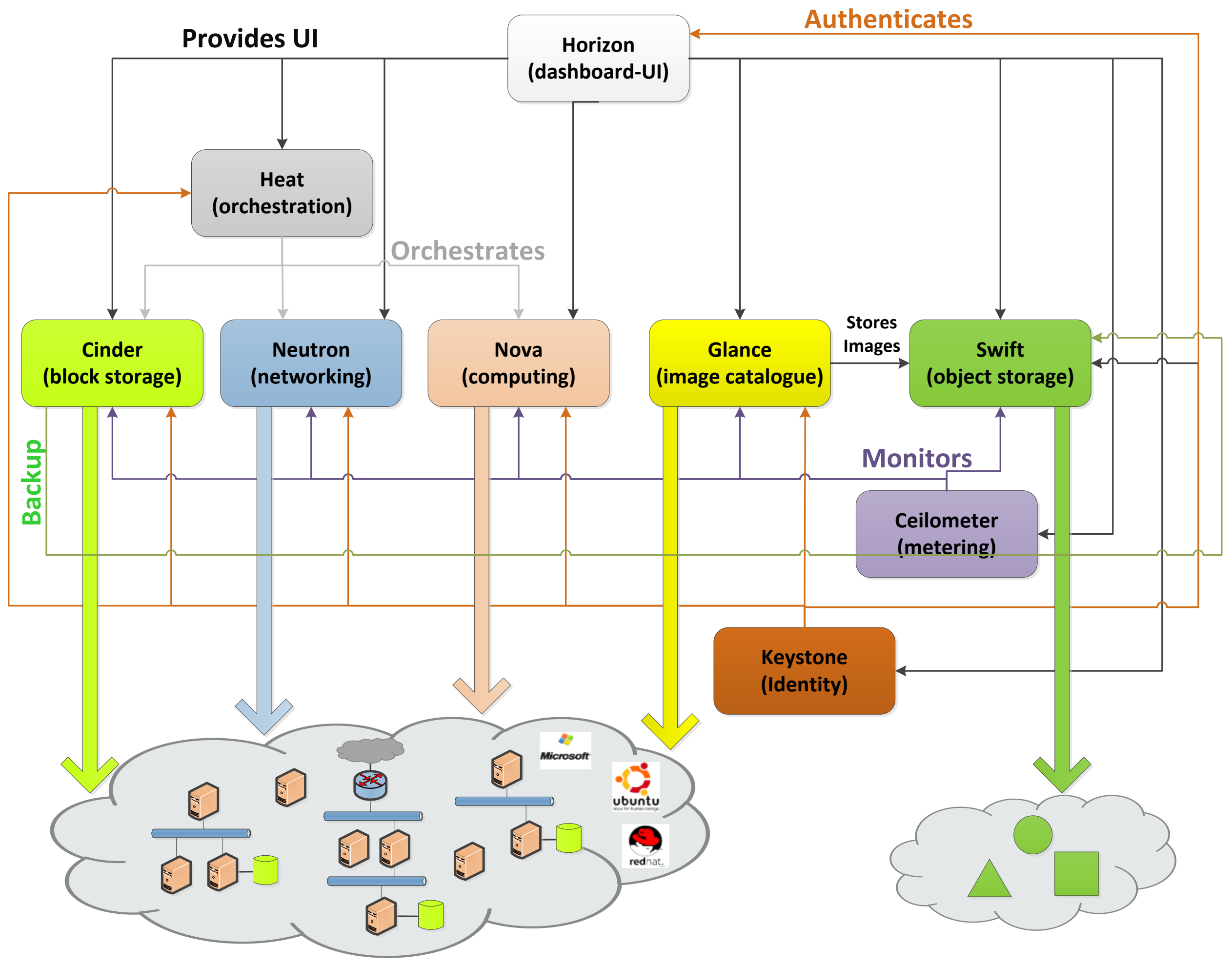


Figure 6-3: OpenStack IaaS cloud platform - logical architecture

OpenStack is composed of the following modules, mapping the fundamental IaaS services:

- **Nova** provides computation services (Virtual Machines)
- **Neutron** provides networking services (Virtual Networks)
- **Cinder** provides block storage services (Virtual Disks)
- **Swift** implements object storage services (files)[1]

In addition, it implements some “ancillary” modules:

- **Horizon** provides a web front-end for managing and controlling purposes;
- **Glance** implements a catalogue for storing virtual machine images;
- **Keystone** implements authentication and authorization;
- **Heat** uses other components for orchestrating the creation/deletion of IaaS object aggregations;
- **Ceilometer** monitors the usage of resources for metering and accounting purposes.

OpenStack modules communicate using a message broker middleware (e.g. RabbitMQ) and store status information in a centralised database (e.g. MySQL); for the sake of brevity, these elements are not shown in the logical architecture.

[1] This service allows users to store ‘objects’ (i.e. files) in the cloud. It can be used, for example, for sharing photos, music, documents, etc. It is not going to be used for the implementation of the RANaaS testbed and it is reported here for the sake of completeness, only.

The OpenStack modules described above are broken down into subcomponents implemented as programs running on physical machines, also named as servers or **nodes**. For example, Cinder - the component providing block storage services - is composed of *cinder-api* subcomponent, *cinder-scheduler* subcomponent and *cinder-volume* subcomponent. In principle, every OpenStack subcomponent can be run on a dedicated node but usually several subcomponents are co-located on a single node. The way the subcomponents are installed on the nodes is described in the **deployment architecture**.

OpenStack can be deployed in several ways but there are some *well-known* and *accepted* good practices for implementing deployment architectures. Usually, the deployment architecture also depends on the scope and the purpose of the cloud and/or the constraints on the physical hardware that is being used (e.g. number and type of physical machines, network connections, etc.).

The following picture shows a typical basic deployment:

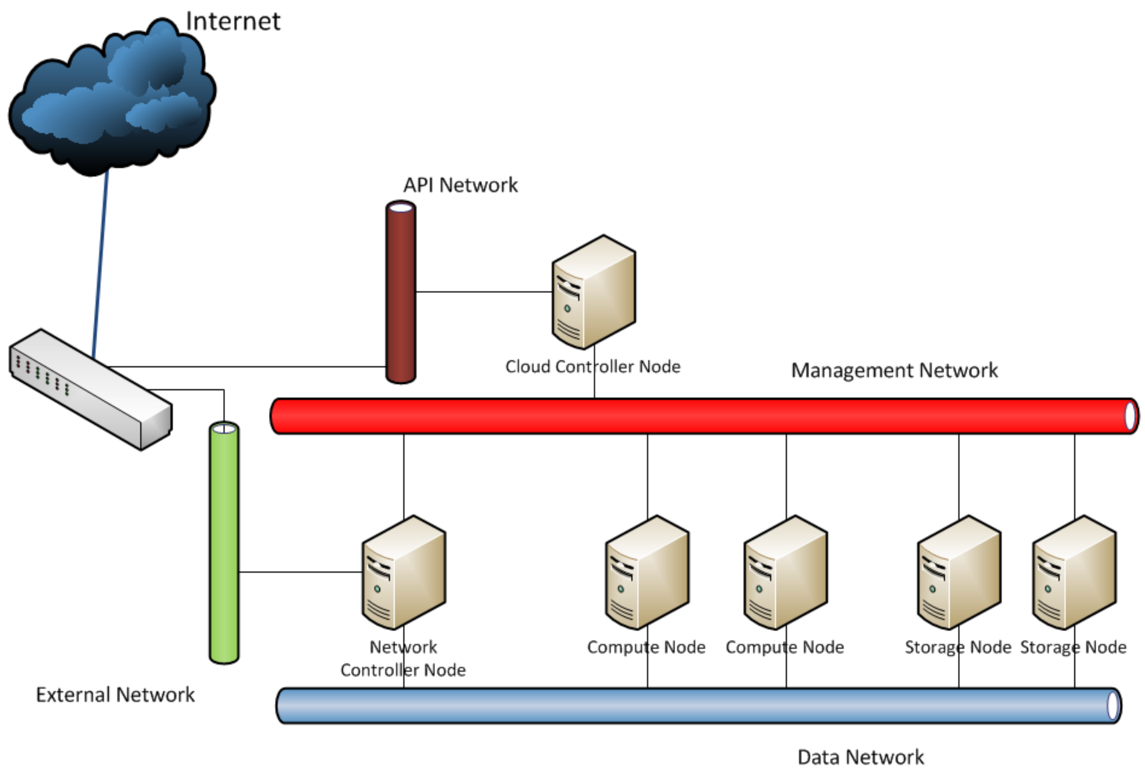


Figure 6-4: OpenStack IaaS cloud platform – basic deployment

The nodes in the deployment architecture are:

- **Cloud Controller Node**: it hosts all the centralised functions like the cloud status database, the message broker, the compute and the storage schedulers, Application Programming Interface (API) endpoints, authentication services, image catalogue, orchestration engine, monitoring and accounting functions, the web dashboard server, etc.;
- **Network Controller Node**: it hosts some network services like DHCP, layer 2 switching, layer 3 routing and also provides access to VMs from Internet;
- **Storage Node**: it hosts Virtual Disks in the cloud;
- **Compute Node**: it hosts the VMs running in the cloud.

The networks in the deployment architecture are:

- **Management Network**: it is used for the communication between the OpenStack elements;
- **Data Network**: it is used for the communication between the VMs running in the cloud and for giving VMs access to Virtual Disks;
- **External Network**: it is used for the communication between the VMs running in the cloud and any other element external to the cloud (e.g. end users on Internet);
- **API Network**: it exposes all OpenStack API endpoints.

It is worth noticing that the deployment described above does not address some important aspects like high availability and load balancing; on the other hand it can be used as the starting point for designing the iJOIN

RANaaS testbed because these aspects are not strictly required for the experimental scope of the project which is mainly focussed on testing and verifying functional aspects.

A.2.2 RANaaS Testbed Deployments

This section describes the iJOIN RANaaS testbeds: the deployment at University of Bremen (UoB) and the implementation at the Telecom Italia (TI) site in Turin.

A.2.2.1 RANaaS Testbed at UoB

The installation at UoB is hosted on HP blade server hardware similar to the deployment detailed above (see Figure 6-4), but in a smaller expansion state. It is equipped with 2 blade servers with the following characteristics:

- Hardware
    - HP BL460c Gen8
    - 64GB RAM
    - 2 Intel Xeon E5-2630 @ 2.6 GHz (corresponding to 12 CPU cores per blade)
- Software
    - Ubuntu 12.04 Operating System
    - OpenStack IceHouse Release

One blade server is a dedicated compute node, the other one is additionally acting as controller and storage node. This allows for the use of up to 20 virtual machines in parallel with one CPU exclusively allocated each.

Since the software running on all instances is identical, virtual machine images generated for one RANaaS testbed are executable also on the other installations, although the available memory and processor resources might differ.

All iJOIN partners interested in demonstration on the RANaaS testbed are given access to the testbed over the Internet, through encrypted SSH (secure shell) tunnelling. This allows even for the direct interaction with the VM instances in real time, if desired.

A.2.2.2 RANaaS Testbed at TI

The installation at TI is hosted on a single server with the following most relevant characteristics:

- Hardware
    - HP DL360 Gen7
    - 48GB RAM
    - 24 Intel Xeon CPU(s) X5660 @ 2.8GHz cores
- Software
    - Ubuntu 14.04 Operating System
    - OpenStack IceHouse Release

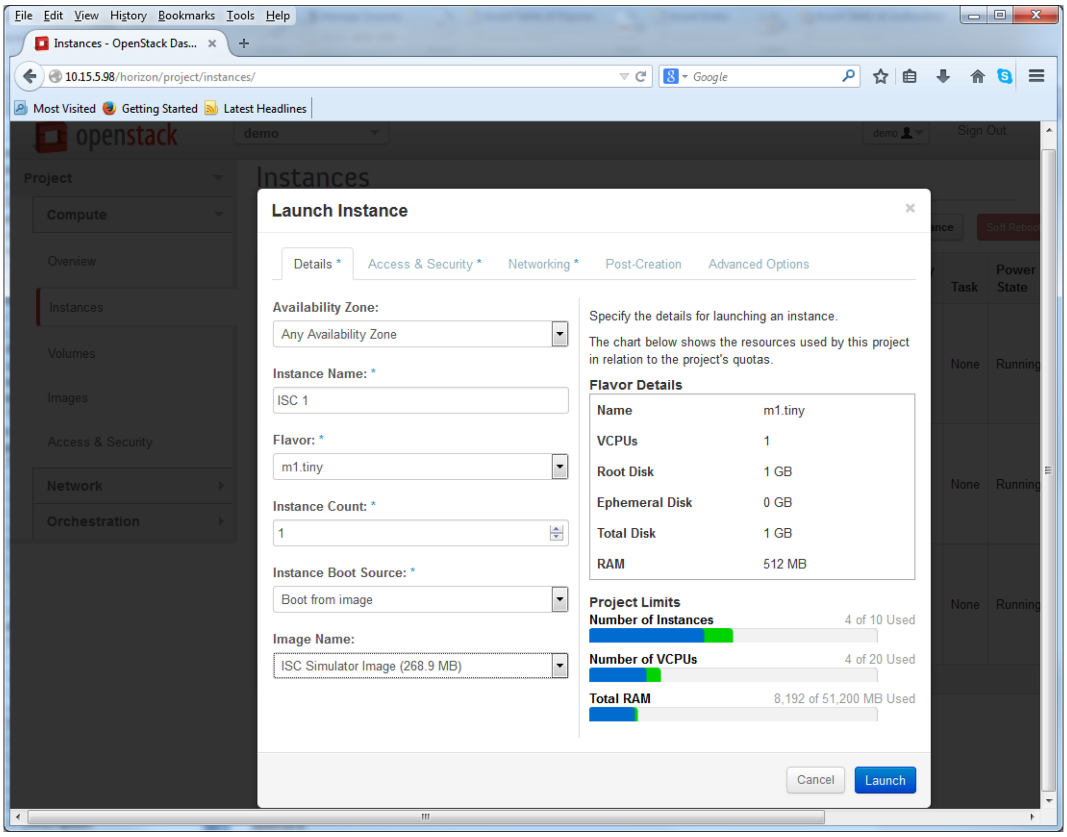
The server is connected to Internet through a switch having a Demilitarized Zone (DMZ) port configured on it. This allows accessing the server from Internet and, at the same time, isolating from the other machines in TI labs. Multiple public IP addresses are assigned to the server in order to be able to access to the different virtual machines (if needed) and to allow parallel configuration and maintenance of the server. The access to all the VMs is available to all project partners interested in demonstration via Internet.

A.3 Interfaces

A.3.1 General Interface Description

As mentioned in the OpenStack architecture (see Section A.2), OpenStack implements Horizon, a web-based application that let the users manage cloud objects through an easy to use point-and-click interface. Using Horizon, cloud users can start/stop virtual machines, create virtual networks, manage virtual machine images, etc.

The following picture shows how a user can start a virtual machine using Horizon:



Availability Zone: Any Availability Zone

Instance Name: * ISC 1

Flavor: * m1.tiny

Instance Count: * 1

Instance Boot Source: * Boot from image

Image Name: ISC Simulator Image (268.9 MB)

Specify the details for launching an instance.

The chart below shows the resources used by this project in relation to the project's quotas.

Flavor Details

Name	m1.tiny
VCPUs	1
Root Disk	1 GB
Ephemeral Disk	0 GB
Total Disk	1 GB
RAM	512 MB

Project Limits

Number of Instances 4 of 10 Used

Number of VCPUs 4 of 20 Used

Total RAM 8,192 of 51,200 MB Used

Figure 6-5: RANaaS – horizon interface

Virtual machines are activated starting from virtual machine images (or **images**, for short) that need to be previously loaded into the OpenStack image catalogue. An image plays the same role of a boot disk for a physical computer: it contains the operating system and the application programs that run on top of it. Annex A.3.2 provides indications for preparing VM images that can be run on an OpenStack cloud.

Using Horizon, it is possible to create and manage several virtual objects in the OpenStack cloud and configure a computing environment that mimics the operating conditions of the iJOIN logical architecture as described in Figure 6-6:

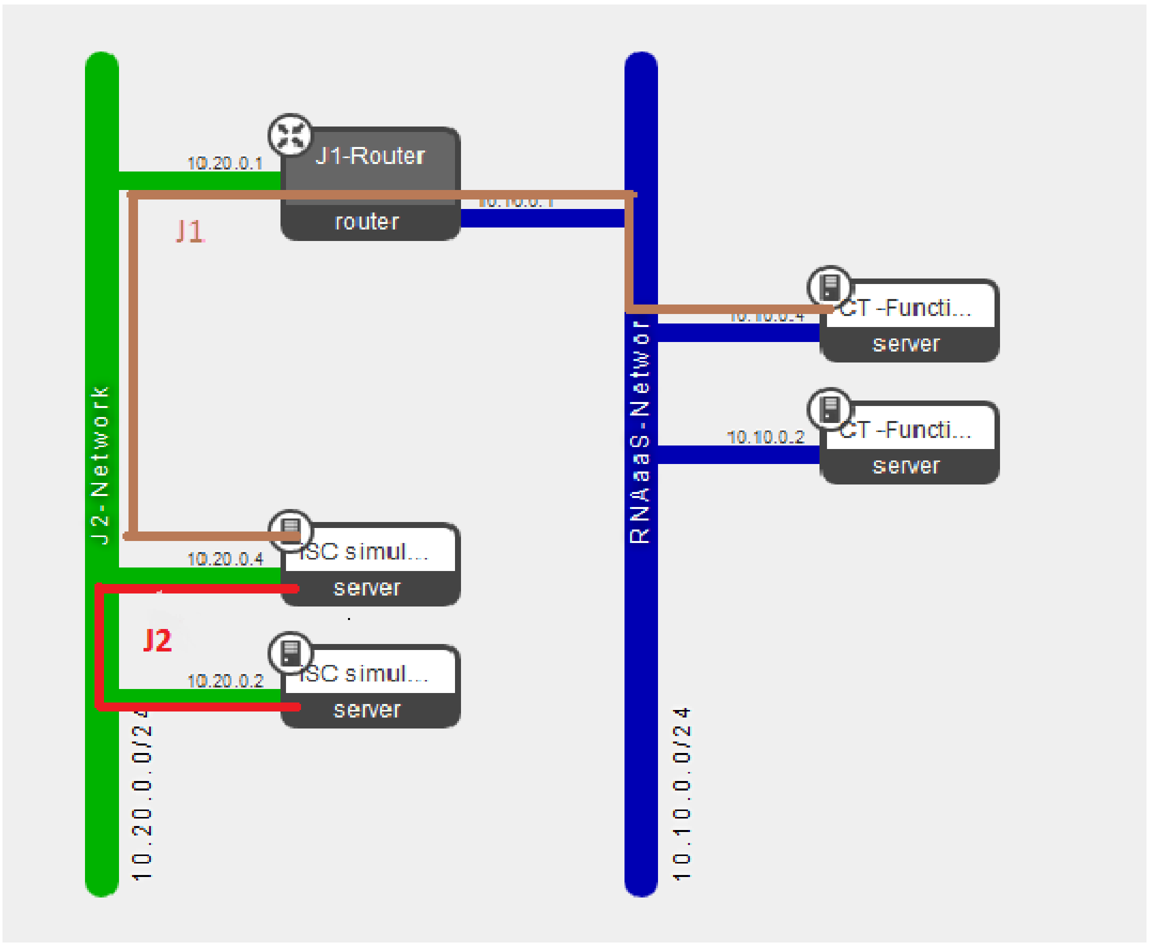


Figure 6-6: RANaaS testbed network configuration

The virtual network in green, named as J2-Network, is used to connect iSC simulators and represents the J2 interface (see Figure 3-1).

CT-Functions, the functions obtained by splitting a CT into several concurrent VMs, are connected to another virtual network, named as RANaaS-Network, which is depicted in blue. This network is actually designed to interface CT functions.

The two networks are connected through a virtual router, J1-Router, and J1 interface is simulated by the path Green-Network/J1-Router/Blue-Network.

For simulating a communication delay on J1 interface, it is sufficient to force network delays on the J1-Router network interfaces using suitable netem[2] commands [23]. It is important to say that the configuration described above imposes some constraint on the IP addresses used for the VMs:

- VMs connected to the same virtual LAN must have IP addresses belonging to the same subnet; for example the VMs that collaborate for implementing a CT must have IP addresses to subnet 10.10.0.0/24 as depicted in the picture above. Same reasoning applies to the VMs simulating iSCs that, in example shown above, have IP addresses on 10.20.0.0/24 subnet.
- J2-Network and RANaaS Network must have different subnets.

[2] An important feature required for testing purposes is the possibility of simulating network transmission delays when virtual machines communicate. This is obtained by taking advantage of netem, a Linux kernel module specifically designed for network emulation functions [15]. netem provides functions for emulating area network delays (even with statistical distributions), packet distribution problems (loss, duplication and corruption), rate control, etc.

A.3.2 Preparing VM Images

In order to run a VM in the RANaaS testbed, it is necessary to prepare a VM image and then publish it into a OpenStack catalogue.

RANaaS testbed provides a catalogue of VM images and when a user wants to start a new VM instance, s/he selects a VM image to boot from. At that point, RANaaS testbed creates a copy of the VM image of the catalogue and starts a ‘VM process’ aimed to play the VM instance in the cloud.

In this perspective, the VM images in the catalogue can be considered as sort of ‘stencils’ for creating VMs running in the RANaaS testbed.

VM images are created ‘outside the cloud’ using a VM player program; then, when the image file is ready, it is uploaded in the cloud’s catalogue. An image must be prepared following some rules in order to integrate with the underlying IaaS platform and for running in a cloud. A full description of the procedure and the steps for creating a VM image is out of the scope of this document and can be found in [24]; this paragraph focuses on the most relevant aspects to be taken into account considering the RANaaS testbed configuration.

VM Image Format

OpenStack supports several image file formats but, in an actual installation, the ones that can be used depend on the hypervisor software selected for running virtual machines. RANaaS testbed uses Kernel-based Virtual Machine (KVM) [25]; therefore the recommended image format is QEMU copy-on-write version 2 format (qcow2).

Before uploading an image to the catalogue, it could be necessary to convert the related file to qcow2 as reported in Ubuntu documentation, KVM/FAQ [26].

VM Image Configuration

In order to have full functionality in the cloud, images must fulfil some requirements.

- **MAC address information mustn’t be hardcoded**. As the MAC address of the network interface card(s) changes every time a VM instance is booted, VM images must not contain any hardcoded reference to MAC addresses.
- **Firewall rules must be disabled.** Firewall functionality is provided by the cloud; therefore it is redundant to configure firewall rules also at VM image level. In addition, setting firewall rules at image level can make it more difficult to troubleshoot networking issues.
- **A SSH server should be installed and run at VM instance start-up**. Although not strictly required (in particular for Windows images), it is very useful having an SSH server installed on the VM images for facilitating troubleshoot and administration operations. Linux operating system natively supports SSH server but there exists a SSH server implementation for Windows operating system [27].
- **cloud-init software package must be installed**. cloud-init software package automatically implements several functions for integrating a running VM instance with the underlying IaaS platform. Some examples are: user creation (a default user account is created in the VM), password injection (when a VM is started the administrator’s password is ‘injected’ into the instance), automatic run of “userdata” script (a script, provided by the user as a parameter at VM instance start-up, is automatically launched), etc. cloud-init package is normally natively installed on the most recent Linux operating system distribution but it is also freely available for Windows operating system [28].

Setting IP Address Acquisition

VMs communicate each other using the network, hence they need to have (at least) one IP address. There are two options for setting an IP address in a VM:

- configuring a fixed IP address (the IP address is statically configured in the image)
- configuring DHCP (the IP address is acquired via DHCP protocol by any instance when it is started).

There are pros and cons for both the approaches.

Fixed IP addresses make the communication between the VMs easier because the software running on each VM can refer other VMs using ‘well known’ IP addresses. As a flip side, it is impossible to run two VM instances obtained by a single VM image on the same virtual network because two VMs with the same IP address cannot be connected to the same network. DHCP, on the other hand, improves flexibility because it is possible to run several VMs obtained by a single VM image but the program(s) running on top of the VMs can’t rely on any pre-determined VM/IP address association because IP addresses are randomly assigned when the VMs start.

It is impossible to say what option is better because it depends on the scope of the test; generally speaking the former technique makes the test design and implementation easier but it is unpractical if the test needs more instances of a given VM image. DHCP IP address assignment, on the other hand, is more flexible but requires some techniques for VMs to ‘know each other’s” IP addresses.

It is worth noticing that mixed techniques can be used; for example VMs implementing iSC simulators could receive via DHCP protocol, while CT’s functions could use fixed IP addresses. In this scenario, the assumption is that iSC simulators initiate the communication to CT’s function(s) using a ‘well known’ IP address.

Annex B Joint Access and Backhaul Testbed Description and Interface

B.1 Concept

The 60 GHz frequency band provides up to 9 GHz of unlicensed bandwidth and is well-suited for wireless links that require data rates of several Gbit/s. Within the iJOIN architecture, this technology is intended for high capacity wireless backhaul on the "last mile" between the RANaaS and an iJOIN Small Cell (iSC). For this kind of application, the wireless channel is typically a line-of-sight channel. This allows to use a simplified transmission scheme (e.g. without channel equalization) to maximize the achievable data rate while keeping the hardware-effort at a minimum.

The demonstrator presented in this section shows the feasibility of such a system design, scaled to be used in a lab environment. It realizes a real-time transmission of binary data utilizing the 60 GHz band with a data rate of multiple Gbit/s over a short distance of approximately one meter. The range is limited due to a low transmit power and low antenna gains but those can in principle be scaled for ranges of a few hundred meters.

B.2 Description

The setup consists of a transmitter and a receiver, each of which comprises a hybrid 60 GHz frontend with directional antennas and a digital baseband processor that is implemented on a Field Programmable Gate Array (FPGA) platform. The simplified system concept builds upon 1-bit data converters at the transmitter and receiver, which in theory would enable a low-power, low-complexity integrated circuit solution in the future. The system design is scalable and not limited to the 60 GHz frequency band. For a real-life backhaul link for transmission over a longer distance the transmission power and antenna gains would have to be increased, yet the architecture and performance should be comparable to the downscaled demonstrator.

The objective of the joint access and backhaul testbed is to evaluate different Candidate Technologies (CTs) in an experimental setup with a real, potentially erroneous backhaul link. For this, simulated backhaul links in the simulation of a CT, which runs on a standard PC in MATLAB, are replaced by the 60 GHz demonstrator, forming a hardware-in-the-loop platform as depicted in Figure 6-7:

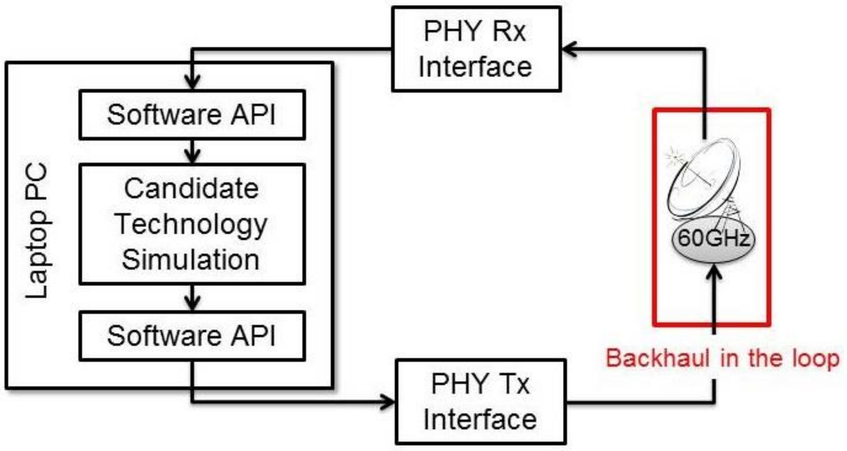


Figure 6-7: Backhaul-in-the-loop platform

The interface between PC and demonstrator is provided by an Ethernet link, limiting the real time data rate between the PC and the hardware platform to around 10 Mbit/s. To ensure easy integration of the hardware demonstrator into the software simulation of the CTs, a simple API is provided, that handles the transmission over the 60 GHz link.

B.2.1 Physical Layer and Digital Baseband

The considered physical layer architecture [29] is shown in Figure 6-8 with the transmitter at the top and the receiver at the bottom.

The architecture uses massive parallelization to achieve a data rate of up to 5 Gbit/s with differential Quadrature Phase Shift Keying (QPSK) modulation.

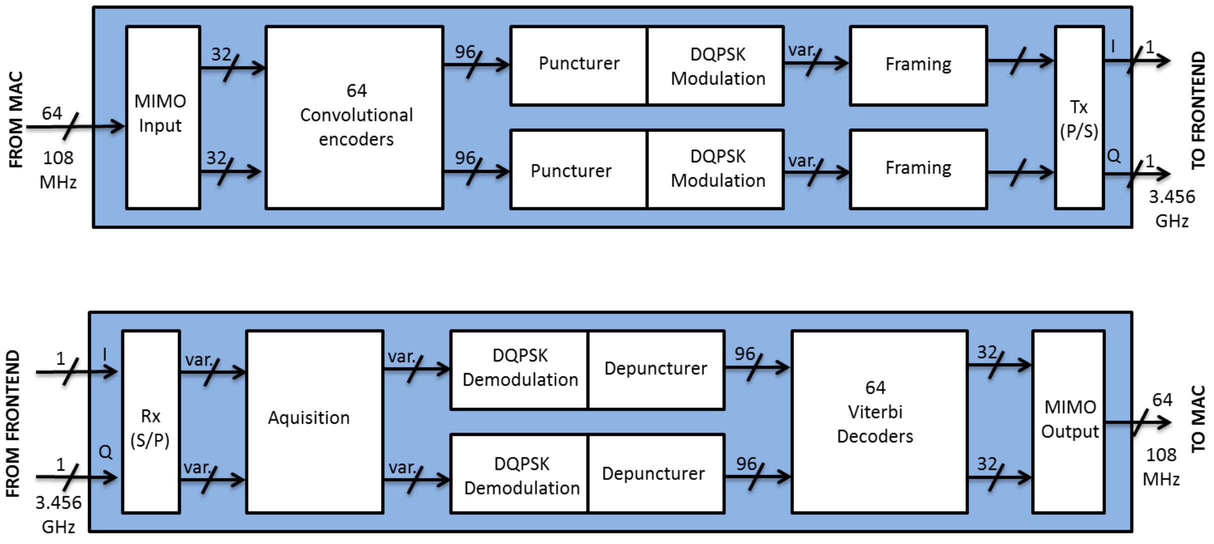


Figure 6-8: Digital baseband architecture of transmitter and receiver of the 60 GHz demonstrator with transmitter (top) and receiver (bottom)

A data sequence of symbol rate 3.456 GHz is split into 2 parallel data sequences corresponding to the two complex dimensions of the QPSK modulation. Each sequence carries 32 parallel (convolutional) codewords with a mother code rate of 1/3. This reduces the processing rate to only 108 MHz for each codeword, but at the cost of 64 en-/decoders.

The differential QPSK modulation enables frequency flat phase equalization at the receiver using the analog carrier recovery proposed in [30]. A residual phase-ambiguity of multiples of 90° is resolved in the digital domain. The bandwidth follows the channelization plan of the ECMA-387 standard [30] with bonding of two channels.

The data is transmitted in a packet-wise manner, with separate packets for the in-phase and quadrature-phase path. The payload of the packets is variable with a default size of 97280 bit. It is preceded with a preamble consisting of three parts: an analog training sequence (2048 bit); a digital training sequence (1024 bit) and a MAC layer signalling field (1024 bit).

Considering the latency when transmitting a single frame, the transmitter data path introduces a delay of 93 clock cycles (0.86 µs at 108 MHz). The receiver data path introduces a delay of 424 clock cycles (3.9 µs), where the main part originates from the Viterbi decoding.

The transmitter and receiver data paths are implemented on off-the-shelf FPGA platforms, each comprising 6 Altera Stratix III and one Altera Stratix II-GX. The integrated transceiver interfaces of the latter serve as the 1-bit data converters that connect to the 60 GHz frontends. The FPGAs are clocked at 108 MHz.

Since the demonstrator uses a simple 1-bit analog-to-digital-converter, it is not possible to calculate the Signal-to-Noise-Ratio (SNR) of the 60 GHz channel. The only channel state information available is the bit error rate before or after decoding. If the distance and code rate are appropriately chosen, the bit error rate is usually zero, meaning that the demonstrator provides an error free link.

An overview of the most important parameters is given in the following table. More detailed investigations on performance parameters can also be found in Section 4.2.1.

Table 6-1: Baseband parameters of the 60 GHz demonstrator

Parameter	Unit	Value
Carrier frequency	GHz	61.56
Bandwidth	GHz	4.320
Symbol rate	GHz	3.456
Modulation		Differential Quaternary Phase Shift Keying (DQPSK)
Coding		Convolutional, mother code rate 1/3
Preamble length	Symbols	4094
Payload length	Symbols	97280
Data rate @ code rate 3/4	Gbit/s	5
Output power of Tx	dBm	10
Maximum range	m	~1
Processing latency (Tx+Rx)	µs	4.79

B.2.2 60 GHz Frontend and Antennas

The radio frequency (RF) frontends have been fabricated from commercial of the shelf bare die components. Both the transmitter and the receiver frontend have a direct mixing, zero Intermediate Frequency (zero-IF) architecture, as shown in Figure 6-9.

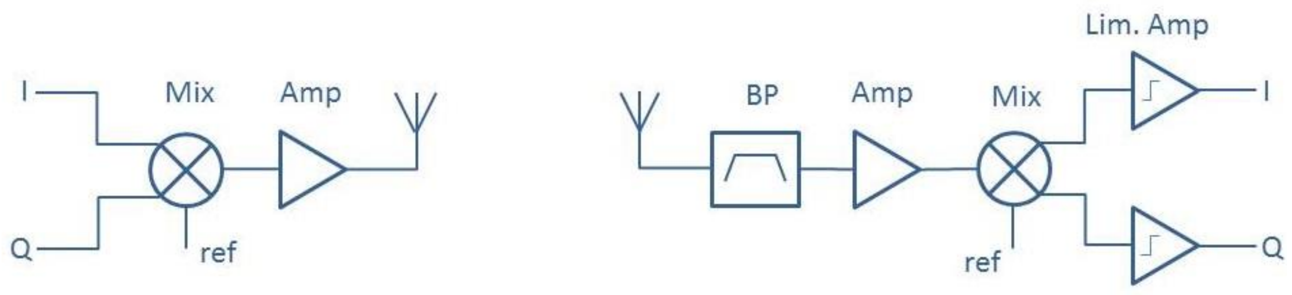


Figure 6-9: Analog frontend of the transmitter (left) and receiver (right) of the 60 GHz demonstrator

The baseband bandwidth of the frontends ranges from 0 to 3 GHz. The upper limit follows from the intermediate frequency (IF) in/-outputs of the mixers. A 7.5 GHz local oscillator (LO) signal has to be supplied from an external source, which is then quadrupled and fed into sub-harmonic mixers. The output power of the transmitter is 10 dBm under default operation conditions. The receiver uses the same mixer architecture and the same LO supply chain as the transmitter.

Since no strong blocking or interference signals are expected for a line-of-sight transmission in the 60 GHz frequency band, the received signal can be filtered after being amplified by a low-noise amplifier. This ensures a noise figure below 7 dB. The overall power consumption for the transmitter is below 900 mW and less than 600 mW for the receiver. Two directional horn antennas are used.

B.3 Interfaces

This testbed provides a hardware-in-the-loop platform that consists of the 60 GHz demonstrator and a PC. The PC runs a simulation of a CT in which one or multiple backhaul links are replaced by the real-life 60 GHz demonstrator link. A simple MATLAB API is provided that allows project partners to easily include the 60 GHz link to demonstrate their CTs.

As described above, the interface between the PC running the simulation and the demonstrator platform is provided by an Ethernet link. This section describes the MATLAB API that can be used by project partners to include the 60 GHz demonstrator in their simulations. To enable partners to test the API without access to the actual testbed, a dummy implementation and testbed emulator are also provided. The dummy implementation differs from the actual implementation in several features, which are highlighted in the respective paragraphs.

Based on the requirements of the involved CTs, the API provides four different functionalities:

- Establish/break a connection between MATLAB/PC and the demonstrator
- Configure the code rate of the demonstrator
- Transmit/ receive bits via the 60 GHz demonstrator
- Evaluate the link quality in terms of BER

This functionality can be provided by four MATLAB functions fully described below.

Establish a connection:

```
function udpHandle = connectTo60GHz()
```

This function establishes the connection to the platform and has to be executed once before the 60 GHz demonstrator can be used. The IP addresses and ports of the 60 GHz transmitter and receiver are hard-coded but they can be accessed in the source code if necessary.

The output is a handle for a UDP object, which is required for all other functions.

Configure the demonstrator:

```
function configure60GHz(udpHandle, code_rate)
```

This function configures the code rate used in the demonstrator for the BH link.

The first input is the handle obtained from `connectTo60GHz()`. The second input is the code rate to be used, possible values are {1/2, 2/3, 3/4}. The different code rates can be used to simulate different channel qualities in terms of BER without having to manually change the channel.

Dummy implementation:

In the dummy implementation, no coding is implemented. Therefore, the input parameter `code_rate` is used to directly set the BER of the demonstrator.

Dissolve a connection:

```
function disconnectFrom60GHz(udpHandle)
```

This function is required for cleanup after the simulation and therefore has to be executed once after the simulation.

Its input is the handle obtained from `connectTo60GHz()`.

Dummy implementation:

In the dummy implementation, this function also stops the execution of the emulator.

Transmit data:

```
function [rx, BER_coded, BER_uncoded] = sendVia60GHz(udpHandle,tx)
```

This function is the basic function for transmitting data. It also evaluates the bit error rate.

The first input is the handle obtained from `connectTo60GHz()`. The second input is the data to be transmitted. It has to be provided in the form of a vector containing “1” or “0”. The vector should have a maximum length of 32 kbit. If any other type of data, e.g. real valued data, needs to be transmitted, it has to be quantized in MATLAB before transmission.

The first output is the received data after the transmission of the input, as a vector of the same length as the input and containing “1” and “0”. If the 60 GHz link does not introduce errors, this output should be identical to the input.

The second output is the coded bit error rate evaluated by the demonstrator by comparing input frames before encoding with output frames after decoding.

The third output is the uncoded bit error rate evaluated by the demonstrator by comparing input frames after encoding with output frames before decoding.

There is also a third error rate that can be evaluated directly in MATLAB without requiring the API: the BER when comparing input and output of the `sendVia60GHz` function. This BER might be different from `BER_coded`, because the MATLAB input will be placed into frames of about 50 kbit length for the 60 GHz transmission. Depending on the size of the input vector of the simulation, these frames will only be filled to a small fraction with actual data and then stuffed with dummy bits. Small BERs might then appear as zero BER in the MATLAB evaluation, but would be evaluated correctly by the demonstrator because it used the whole frame as a basis.

Dummy implementation:

Since no coding is implemented, only one BER is returned as output, which is the BER as evaluated by the emulator.

Auxiliary functions

The API also includes three auxiliary functions:

`bits2char(char)`: converts a bit vector to characters

`char2bits(bits)`: converts characters into a bit vector

`UDPCleanup(udpHandle)`: closes the UDP object and deletes it

Testbed emulator

```
function TUDDemoEmulator()
```

This function can be used to emulate the 60GHz demonstrator. It emulates its UDP connectivity and models the 60 GHz link as a simple Binary Symmetric Channel (BSC). For this, it needs to run on a separate MATLAB instance, either on the same PC as the CT simulation, or on a separate PC that is connected to the one running the CT simulation via a (wireless) local area network. The used IP addresses and ports are hard coded but can be accessed via the source code, if necessary.

The emulator has to be started before `connectTo60GHz` is called. It only terminates if `disconnectFrom60GHz` is called in the CT simulation, otherwise the function will run forever and has to be terminated by hitting ctrl+c in MATLAB. A diagram of an exemplary program flow for using the emulator is depicted in Figure 6-10.

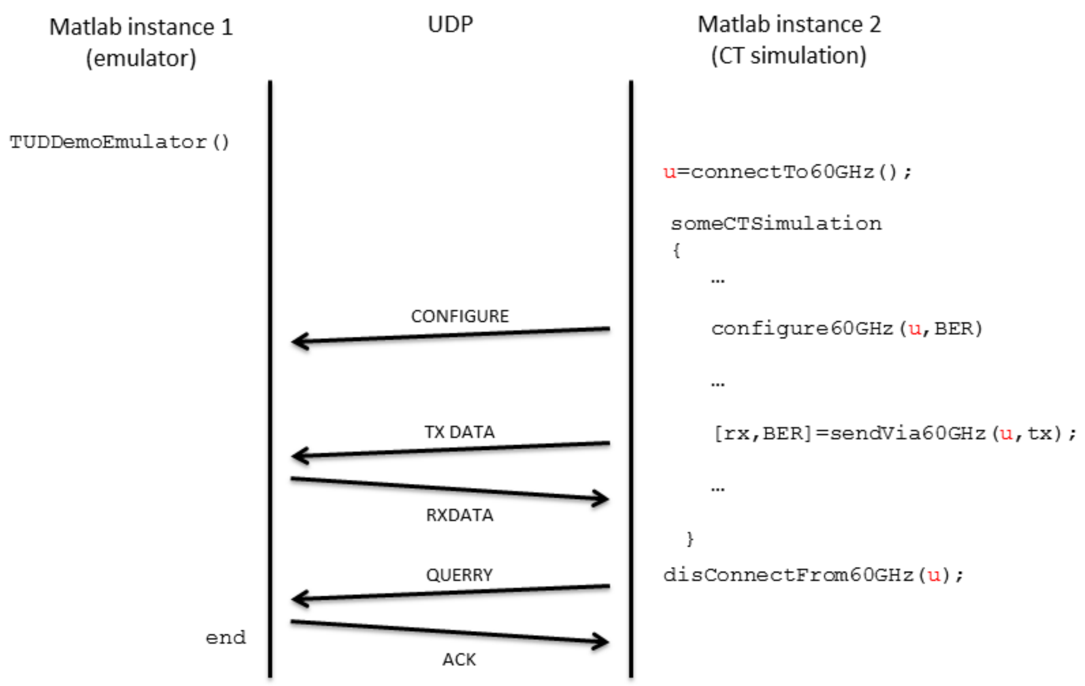


Figure 6-10: 60 GHz emulator program flow

Annex C SDN Testbed Description and Interface

C.1 Concept

Software Defined Networking (SDN) is a new approach to designing, building and managing networks that separates the control and the data forwarding planes. Such separation allows for quicker provisioning and configuration of network connections. With SDN, network administrators can program the behaviour of both the traffic and the network in a centralised way, without requiring independently accessing and configuring each of the networks hardware devices. This approach decouples the system that makes decisions about where traffic is sent (i.e., control plane) from the underlying system that forwards traffic to the selected destination (i.e., data plane). Among other advantages, this simplifies networking as well as the deployment of new protocols and applications. In addition, by enabling programmability on the traffic and the devices, an SDN network might be much more flexible and efficient than a traditional one.

In SDN environments, the Network Controller is responsible to configure the nodes in the network via a common Application Programming Interface (API), namely Southbound API. OpenFlow [31] is one of such APIs and is currently the standard de facto being implemented by major vendors, with OpenFlow-enabled switches now commercially available. On the contrary, the external software applications interact with the Network Controller via a Northbound API and, unlike the Southbound API, there is no common definition of such interface. Figure 6-11 depicts the SDN architecture.

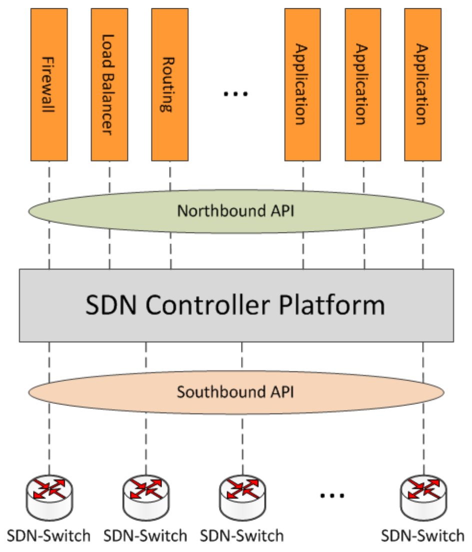


Figure 6-11: SDN architecture

Radio Access Network can exploit such architecture by implementing routing, access control, load balancing, mobility management, etc., as applications running on top of the Network Controller. Indeed, by the use of SDN technologies in a Radio Access Network, it is possible to properly configure the network equipment pursued by several CTs within iJOIN, as explained in [19]. Basing the decision on the status of the network and on the traffic pattern of the User Equipments (UEs) increases the utilisation of the available bandwidth.

The objective of the SDN testbed is to partially simulate the iJOIN functional architecture [19] with particular emphasis to the networking aspects. Under this perspective, architecture nodes - like iJOIN Transport Node (iTN), evolved Node B (eNB) and iJOIN Local Gateway (iLGW) - are emulated by a set of off-the-shelf devices and the Network Controller (i.e., iNC) is introduced for supporting the network orchestration functions required by the CTs.

C.2 Description

The following picture shows the SDN Testbed architecture:

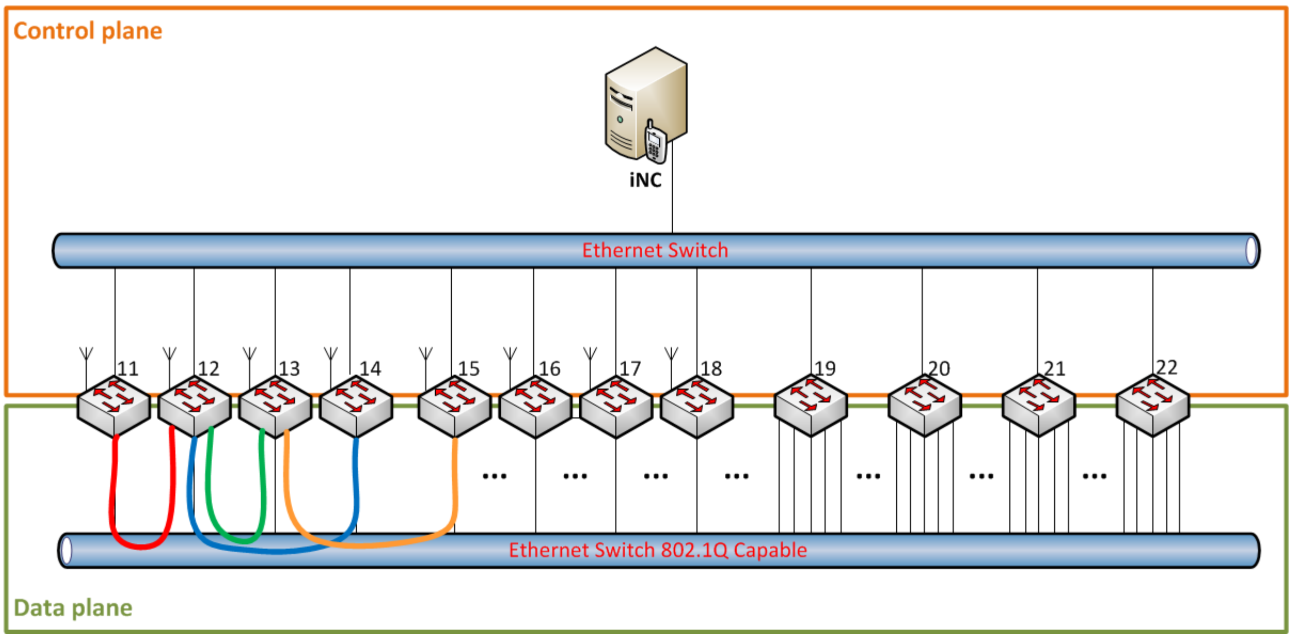


Figure 6-12: SDN testbed architecture

The SDN testbed is deployed using commodity hardware. The use of commodity hardware allows having a number of nodes big enough to emulate different network topologies at a reasonable implementation cost. In details, the testbed is composed by the following equipment:

- Alix2d2 (8 units) [32]: low power single board computers with multiple interfaces. These boards are specifically designed for playing the role of wireless router, firewall, load balancer, VPN etc. Alix2d2 boards are depicted in Figure 6-12 with the boxes numbered from 11 to 18.

  Role: these boards can play the role of iTNs, eNBs and iLGWs [19].

  Hardware:

  - 500 MHz AMD GeodeLX800
  - 256MB DDRDRAM
  - Compact Flash socket equipped with 4GB of memory storage
  - 44 pin Integrated Device Electronics header
  - 2 Ethernet ports 100BASE-TX
  - 2 miniPCI slots equipped with Atheros and Broadcom 802.11g wireless cards

  Software:

  - Debian Wheezy 32bit operating system.
  - Open vSwitch 2.3.1 with OpenFlow 1.3 support
  - Custom Linux Kernel 3.10.11 including Open vSwitch modules
  - Hostpad is used to create 802.11 networks

- Core-MC500 (4 units): high power single board computers with multiple interfaces. These boards are designed for being versatile allowing easy upgrades and customization. Core-MC500 computers are depicted in Figure 6-12 with the boxes numbered from 19 to 21.

  Role: these boards can play the role of iTNs and iLGWs [19].

Hardware:

- 2.7 GHz Intel Core i5-3610ME
- 8GB SO-DIMM DDR3 RAM
- 2.5" SATA SSD with 64GB of memory storage
- 6 Ethernet ports 1000BASE-TX
- 1 miniPCIe available for wireless cards

Software:

- Debian Wheezy 64bit operating system.
- Open vSwitch 2.3.1 with OpenFlow 1.3 support
- Custom Linux Kernel 3.10.11 including Open vSwitch modules

- Desktop (1 unit):

  Role: runs the network controller.

  Hardware:

  - Intel Core 2 Duo
  - 2GB RAM

  Software:

  - Debian Wheezy 64bit as operating system
  - Ryu as network controller with OpenFlow 1.3 support

- Switch with 24-port 100BASE-TX 802.1Q capable (2 units): these switches are used to interconnect the control plane and the data plane of Alix2d2 and Core-MC500 boards.

The SDN testbed implements both IEEE 802.3 and 802.11 physical transmission protocols. The former, also known as Ethernet protocol, is used for connecting the Alix2d2 and Core-MC500 nodes one to another and to connect the network controller to them; the latter, also known as Wi-Fi LAN is used to provide connectivity to UEs.

Eventually, after considering some other network controllers, Ryu [33] was adopted as default controller since it provides software components with well-defined API that makes it easy to create new network management and control applications. Each CT is implemented as a Ryu application, or, what is the same, a program written in Python that make use of Ryu APIs. Ryu supports various protocols for managing network devices, such as OpenFlow, the one chosen in iJOIN. The complete Ryu code is freely available under the Apache 2.0 license.

Regarding the network topology of the SDN testbed, two different topology layers were considered: the physical and the logical topology. The physical topology of the testbed is flat: all Alix2d2 nodes are connected to a single network switch. Alix2d2 provide two Ethernet interfaces which are used as follows: one interface is connected to the controller, performing an out-of-band signalling of control traffic, while the other interface is reserved for data traffic thus separating then data and control traffic. Core-MC500 provide six Ethernet interfaces which are used as follows: one interface is connected to the controller, performing an out-of-band signalling of control traffic, while the other interfaces are reserved for data traffic like in Alix2d2 case. The logical topology of the network is emulated exploiting 802.1Q capabilities of the switch as depicted in Figure 6-12 where the logical connections for data traffic plane are marked in colours. The role played by the equipment depends on the testbed configuration and on the scenario being demonstrated. The configuration of the logical network topology is obtained with a suitable setup described in Annex C.3.

Combining Linux capabilities to create VLAN interfaces and 802.1Q, switch forwarding allows an easy emulation of every desired topology by changing only the VLAN ID assigned to each Alix2d2 and Core-MC500. Linux VLAN interfaces play an important role since it gives the possibility to create as many VLAN interfaces as desired on the same physical interface, enhancing the configuration possibilities of the testbed. The role that every Alix2d2 and Core-MC500 play in the network can also be defined and configured dynamically.

C.3 Interfaces

The SDN testbed is accessible to the project partners from the Internet via a VPN connection.

CT functions are implemented as Ryu application programs running in the testbed controller. CTs operate on the underlying infrastructure (i.e Alix2d2 and Core-MC500 network nodes) solely through Ryu API calls [34].

The logical topology is configured on the Alix2d2 and Core-MC500 before testing the CTs.

The configuration is obtained operating at the Operating System level and using Open vSwitch commands, in order to establish desired network conditions.

Establishment and configuration of VLAN's interfaces allows the creation of custom topologies.

The configuration is done by editing the `etc/network/interfaces` configuration file, as described in the following example:

```
auto eth0
iface eth0 inet static
     address 192.168.10.15
     broadcast 192.168.10.255
     netmask 255.255.255.0
     pre-up ip link add link eth0 name of15_11 type vlan id 1115
     post-down ip link del of15_11 type vlan
     pre-up ip link add link eth0 name of15_12 type vlan id 1215
     post-down ip link del of15_12 type vlan
     pre-up ip link add link eth0 name of15_16 type vlan id 1516
     post-down ip link del of15_16 type vlan
     pre-up ip link add link eth0 name of15_17 type vlan id 1517
     post-down ip link del of15_17 type vlan
     # The CN is the node-18
     pre-up ip link add link eth0 name gw15 type vlan id 666
     post-down ip link del gw15 type vlan
```

Once the VLANs are configured, it is necessary to enable OpenFlow on the created interfaces.

That can be done launching a set of commands detailed below:

```
# Create br0 Open vSwitch bridge device
ovs-vsctl add-br br0
# Set br0 OpenFlow controller
ovs-vsctl set-controller br0 <CONTROLLERIP>
# Set OpenFlow protocols
ovs-vsctl set bridge br0 protocols=OpenFlow10,OpenFlow12,OpenFlow13

# Connect ports to br0
ovs-vsctl add-port br0 <INTERFACE>
ovs-vsctl add-port br0 wlan0

# Start Linux Wireless access point manager
service hostapd start

# Start Open vSwitch manager
service openvswitch-switch start
```

where:

- *`<CONTROLLERIP>`* is the IP address of the controller machine (i.e. the laptop);
- *`<INTERFACE>`* is the name of the Network Interface Card of the device that's being configured (e.g. eth0).

For convenience reasons, the commands shown above can be gathered in a single script program for facilitating the configuration procedure.

References

[1] iJOIN INFSO-ICT-317941, "D5.3 - Final definition of iJOIN architecture," April 2015.

[2] iJOIN INFSO-ICT-317941, "D4.3 - Final definition and evaluation of network-layer algorithms and network operation and managament," April 2015.

[3] iJOIN INFSO-ICT-317941, "D6.1 - Preliminary proof-of-concept results for selected candidate algorithms," April 2015. [Online]. Available: http://www.ict-ijoin.eu/wp-content/uploads/2012/10/D6-1.pdf. [Accessed 22 April 2015].

[4] "OpenStack Project," [Online]. Available: http://openstack.org. [Accessed June 2014].

[5] The MathWorks, Inc., "MATLAB," 1994-2015. [Online]. Available: http://www.mathworks.com. [Accessed 22 April 2015].

[6] D. Wübben, P. Rost, J. Bartelt, M. Lalam, V. Savin, M. Gorgoglione, A. Dekorsy and G. Fettweis, "Benefits and Impact of Cloud Computing on 5G Signal Processing," *IEEE Signal Processing Magazine,* vol. 31, no. 6, pp. 35-44, 2014.

[7] P. Rost, I. Berberana, A. Maeder, H. Paul, V. Suryapralkash, M. Valenti, D. Wübben, A. Dekorsy and G. Fettweis, "Benefits and Challenges of Virtualization in 5G Radio Access Networks," *IEEE Communications Magazine,* 2015 (submitted).

[8] iJOIN INFSO-ICT-317941, "D2.2 - Definition of PHY layer approaches that are applicable to RANaaS and a holistic design of backhaul and access network," Nov 2014. [Online]. Available: http://www.ict-ijoin.eu/wp-content/uploads/2012/10/D2.2.pdf. [Accessed 22 April 2015].

[9] iJOIN INFSO-ICT-317941, "D3.3 - Final definition and evaluation of MAC and RRM approaches for RANaaS and a joint backhaul/access design," April 2015.

[10] P. Rost, S. Talarico and M. Valenti, "The Complexity-Rate Tradeoff of Centralized Radio Access Networks," *submitted to IEEE Transactions on Wireless Communications,* March 2015.

[11] "SPUC - Signal Processing using C++," [Online]. Available: http://spuc.sourceforge.net/. [Accessed 5 September 2014].

[12] die.net, "clock_gettime(3) - Linux man page," [Online]. Available: http://linux.die.net/man/3/clock_gettime. [Accessed 2 April 2015].

[13] "Common Public Radio Interface," [Online]. Available: http://www.cpri.info. [Accessed 22 April 2015].

[14] iJOIN INFSO-ICT-317941, "D2.3 - Final definition and evaluation of PHY layer approaches for RANaaS and joint backhaul-access layer design," April 2015.

[15] P. de Kerret, R. Fritzsche, D. Gesbert and U. Salim, "Robust Precoding for Network MIMO with Hierarchical CSIT," in *ISWCS 2014*, Barcelona, 2014.

[16] iJOIN INFSO-ICT-317941, "D5.2 - Final Definition of iJOIN Requirements and Scenarios," November 2014. [Online]. Available: http://www.ict-ijoin.eu/wp-content/uploads/2012/10/D5.2.pdf. [Accessed 22 April 2015].

[17] J. Bartelt and G. Fettweis, "An Improved Decoder for Cloud-Based Mobile Networks under Imperfect Fronthaul," in *Globecom 2014 Workshop - Wireless optical network convergence in support of cloud architectures (GC14 WS - WONC),* Austin, 2014.

[18] C. Bernardos and J. Zuniga, "PMIPv6-based distributed anchoring, draft-bernardos-dmm-distributed-anchoring-05," 2015.

[19] iJOIN INFSO-ICT-317941, "D4.2 - Network-layer algorithms and network operation and management:

candidate technoligues specification," 2014. [Online]. Available: http://www.ict-ijoin.eu/wp-content/uploads/2012/10/D4.2.pdf. [Accessed 9 April 2015].

[20] IEEE Std. 802.1ab, "IEEE 802.1ab-rev - Station and Media Access Control Connectivity Discovery," 2009.

[21] ITU-T G.783, "Characteristics of synchronous digital hierarchy (SDH) equipment functional blocks".

[22] Bellcore GR-235-core, "Synchronous Optical Network (SONET) Transport Systems: Common Generic Criteria".

[23] "netem description," [Online]. Available: http://www.linuxfoundation.org/collaborate/workgroups/networking/netem. [Accessed June 2014].

[24] "OpenStack Virtual Machine Image Guide," [Online]. Available: http://docs.openstack.org/image-guide/content/. [Accessed June 2014].

[25] "Kernel Based Virtual Machine," [Online]. Available: http://www.linux-kvm.org/page/Main_Page. [Accessed June 2014].

[26] "Ubuntu Documentation - KVM/FAQ," [Online]. Available: https://help.ubuntu.com/community/KVM/FAQ. [Accessed June 2014].

[27] "OpenSSH for Windows," [Online]. Available: http://sshwindows.sourceforge.net/. [Accessed June 2014].

[28] "Cloud-Init for Windows instances," [Online]. Available: http://www.cloudbase.it/cloud-init-for-windows-instances/. [Accessed June 2014].

[29] G. Fettweis, F. Guderian, S. Krone, "Entering the path towards terabit/s wireless links," in *Design, Automation Test in Europe Conference Exhibition*, March 2011.

[30] A. C. Ulusoy, G. Liu, M. Peter, R. Felbecker, H. Y. Abdine, and H. Schumacher, "A BPSK/QPSK Receiver Architecture Suitable for Low-Cost Ultra-High Rate 60 GHz Wireless Communications," in *Proceedings of the European Microwave Conference (EuMC10)*, September 2010.

[31] O. N. Foundation, "OpenFlow Switch Specification - Version 1.3.0," June 2012. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf. [Accessed September 2014].

[32] "alix2d2 System board - Data Sheet Page," [Online]. Available: http://www.pcengines.ch/alix2d2.htm. [Accessed July 2014].

[33] "Ryu SDN Framework - Project Web Page," [Online]. Available: http://osrg.github.io/ryu/. [Accessed July 2014].

[34] "Ryu API Reference," [Online]. Available: http://ryu.readthedocs.org/en/latest/api_ref.html. [Accessed July 2014].