



**5G Programmable Infrastructure Converging
disaggregated network and compUte REsources**

D4.2 Complete design and initial evaluation of developed functions

This project has received funding from the European Union's Framework Programme

Horizon 2020 for research, technological development and demonstration

5G PPP Research and Validation of critical technologies and systems

Project Start Date: June 1st, 2017

Duration: 30 months

Call: H2020-ICT-2016-2

Date of delivery: 30th November 2018

Topic: ICT-07-2017

Version 1.0

Project co-funded by the European Commission

Under the H2020 programme

Dissemination Level: Public

Grant Agreement Number:	762057
Project Name:	5G Programmable Infrastructure Converging disaggregated neTwork and compUte REsources
Project Acronym:	5G-PICTURE
Document Number:	D4.2
Document Title:	Complete design and initial evaluation of developed functions
Version:	1.0
Delivery Date:	November 30 th 2018
Responsible:	Huawei (HWDU)
Editor(s):	Kostas Katsalis (HWDU), Daniel Camps (I2CAT)
Authors:	Chia-Yu Chang (EUR), Robert Schmidt (EUR), Navid Nikaein (EUR), Daniel Camps-Mur (I2CAT/UPC), Ilker Demirkol (I2CAT/UPC), Eduard García-Villegas (I2CAT/UPC), Matteo Grandi (I2CAT), Marco Bonola (CNIT), Salvatore Pontarelli (CNIT), Igor Freire (BWT), Peter Legg (BWT), Kostas Katsalis (HWDU), Siyu Tang (HWDU), Jobin Francis (TUD), Jay-Kant Chaudhary (TUD), Jesús Gutiérrez (IHP), Vladica Sark (IHP), Eckhard Grass (IHP), Thierno Diallo (UNIVBRIS-HPN), Anna Tzanakaki (UNIVBRIS-HPN), Nikos Makris (UTH), Paris Flegkas (UTH), Kostas Choumas (UTH).
Keywords:	Network Function Virtualisation, Physical and Virtual Network Function, Radio Access Network, Transport Network, Synchronisation Service, Network Slicing, End-to-end Service
Status:	Final
Dissemination Level	Public
Project URL:	http://www.5g-picture-project.eu/

Revision History

Rev. N	Description	Author	Date
0.1	Definition of table of contents	Kostas Katsalis (HWDU)	16 April 2018
0.2	Structure revision	Daniel Camps-Mur (I2CAT)	2 July 2018
0.3	Structure revision	Kostas Katsalis (HWDU)	19 July 2018
0.4	Structure revision, Executive summary and X-Ethernet Technology (section 3.4)	Kostas Katsalis (HWDU)	12 Sept 2018
0.5	Added 5G Programmable Infrastructure Converging disaggregated network and compute Resources (section 3) Added IEEE 1588 over IEEE 802.11ad (section 4.2) Introduction of section 4	Daniel Camps-Mur (I2CAT) Igor Freire (BWT)	10 October 2018
0.6	Added OPP functions, section 3.6	Salvatore Pontarelli (CNIT)	16 October 2018
0.7	Synchronisation harmonizer in section 4.3	Jesús Gutiérrez (IHP), Vladica Sark (IHP)	18 October 2018
0.8	LTE/5G RAN as VNFs implementation (section 2)	Nikos Makris (UTH)	18 October 2018
0.9	Optimal functional split (section 2.1)	Jobin Francis (TUD)	20 October 2018
0.10	Flex-e technical component analysis	Kostas Katsalis (HWDU)	21 October 2018
0.11	Added section 1, introduction for section 3, structure updates on sec. 4	Kostas Katsalis (HWDU)	28 October 2018
0.12	Revised OPP contribution, added intro for section 2, added description for technical components, added section 5	Salvatore Pontarelli (CNIT) Kostas Katsalis (HWDU)	5 November 2018
0.12	added description for technical component subsection 2.2, 2.3	Chia-Yu Chang (EUR)	5 November 2018
0.12	added description for technical components subsection 2.4	Nikos Makris (UTH)	5 November 2018
	added description for technical components subsection 2.5	Daniel Camps (I2CAT)	5 November 2018
0.13	Added segment routing (section 3.5) and conclusions in section 6	Kostas Katsalis (HWDU)	8 November 2018
0.13	TSON subsections in section 3.2	Thierno Diallo (UNIVBRIS)	8 November 2018
0.2	First revision	Kostas Katsalis (HWDU), Thierno Diallo (UBristol)	19 November 2018
0.3	Revision of subsections 2.6, 4.2	Daniel Camps (I2CAT) Matteo Grandi (I2CAT) Peter Legg (BWT)	22 November 2018
0.4	Revision of subsection 2.5	Daniel Camps (I2CAT), Matteo Grandi (I2CAT)	26 November 2018
0.5	Revision of subsection 4.3	Jesus Gutiérrez (IHP), Vladica Sark (IHP)	27 November 2018
1.0	Final revision and Submission	Jesús Gutiérrez (IHP)	30 November 2018

Table of Contents

EXECUTIVE SUMMARY	11
1 INTRODUCTION.....	12
2 VNFS AND PNFS FOR DYNAMIC 5G-RAN DEPLOYMENTS	14
2.1 Technical Component 1: Optimal functional split	15
2.1.1 Summary Description.....	15
2.1.2 Used programmable platforms and APIs	15
2.1.3 Function Design/Implementation/Evaluation	15
2.1.4 Packaging for the 5G OS	18
2.2 Technical Component 2: Implementation of functional split using the OAI platform	18
2.2.1 Summary Description.....	18
2.2.2 Used programmable platforms and APIs	19
2.2.3 Function Design/Implementation/Evaluation	19
2.2.4 Packaging for the 5G OS	24
2.3 Technical Component 3: Flexible Functional Splits	24
2.3.1 Summary Description.....	24
2.3.2 Used programmable platforms and APIs	24
2.3.3 Function Design/Implementation/Evaluation	25
2.3.4 Packaging for the 5G OS	26
2.4 Technical Component 4: Disaggregated Heterogeneous Base Station functionality	27
2.4.1 Summary Description.....	27
2.4.2 Used programmable platforms and APIs	28
2.4.3 Function Design/Implementation/Evaluation	28
2.4.4 Evaluation Methodology and experiment setup	29
2.4.5 Packaging for the 5G-OS	32
2.5 Technical Component 5: Wireless Transport Technologies with Functional Split Support	33
2.5.1 Summary Description.....	33
2.5.2 Used programmable platforms and APIs	34
2.5.3 Function Design/Implementation/Evaluation	34
2.5.4 Evaluation Methodology and experiment setup	35
2.5.5 Packaging for the 5G OS	39
3 TRANSPORT SLICING FOR CONVERGED WIRED-WIRELESS FH/BH NETWORKS AND INTEGRATION WITH 5G-PICTURE ORCHESTRATOR.....	40
3.1 Introduction	40
3.2 Technical Component 1: Time Shared Optical Network (TSON).....	40
3.2.1 Summary Description.....	40
3.2.2 Used programmable platforms and APIs	42
3.2.3 Function Design/Implementation/Evaluation	45
3.2.4 Packaging for the 5G-OS	46

3.3	Technical Component 2: Flex-E	47
3.3.1	Summary Description	47
3.3.2	Used programmable platforms and APIs	47
3.3.3	Function Design/Implementation/Evaluation	47
3.3.4	Packaging for the 5G OS	51
3.4	Technical Component 3: X-Ethernet	52
3.4.1	Summary Description	52
3.4.2	Used programmable platforms and APIs (WP3)	52
3.4.3	Function Design/Implementation/Evaluation	52
3.4.4	Packaging for the 5G OS	55
3.5	Technical Component 4: Segment Routing	57
3.5.1	Summary Description	57
3.5.2	Used programmable platforms and APIs	57
3.5.3	Function Design/Implementation/Evaluation	58
3.5.4	Packaging for the 5G OS	63
3.6	Technical Component 5: Open Packet Processing (OPP)	64
3.6.1	Summary Description	64
3.6.2	Used programmable platforms and APIs	64
3.6.3	Function Design/Implementation/Evaluation	64
3.6.4	Packaging for the 5G-OS	67
3.7	Technical Component 6: Solution based on IEEE 802.11 technologies, both for access and BH (802.11ac modems)	67
3.7.1	Summary Description	67
3.7.2	Used programmable platforms and APIs	68
3.7.3	Function Design/Implementation/Evaluation	68
3.7.4	Packaging for the 5G OS	73
4	PNFS AND VNFS TO SUPPORT SYNCHRONISATION SERVICES IN CONVERGED FH/BH NETWORKS	74
4.1	Introduction	74
4.2	Technical Component 1: IEEE 1588 over IEEE 802.11ad	74
4.2.1	Summary Description	74
4.2.2	Used programmable platforms and APIs	74
4.2.3	Function Design/Implementation/Evaluation	80
4.2.4	Packaging for the 5G OS	86
4.3	Technical Component 4: Synchronisation harmonizer	87
4.3.1	Summary Description	87
4.3.2	Used programmable platforms and APIs (WP3)	88
4.3.3	Function Design/Implementation/Evaluation	88
4.3.4	Packaging for the 5G-OS	89
5	5G-PICTURE INTEGRATION PLANS	90
5.1	WP4 functions in the context of WP3 and WP5	90
5.2	The 5G-PICTURE approach to P/VNF on-boarding to 5G OS	91

5.3	Initial WP4 integration plans toward D4.3 and WP6	92
6	SUMMARY AND CONCLUSIONS.....	93
7	REFERENCES.....	ERROR! BOOKMARK NOT DEFINED.
8	ACRONYMS.....	98

List of Figures

Figure 2-1: RAN network topology for three-tier RAN entities. 14

Figure 2-2: System model. a) Multi-cell massive MIMO cellular system (left) b) An individual RU (right). 15

Figure 2-3: SE and EE maximisation problems. 16

Figure 2-4: Network throughput as a function of FH capacity a) MRT (left) b) ZF (right). 16

Figure 2-5: Energy efficiency as a function of FH capacity a) MRT(left) b) ZF(right). 17

Figure 2-6: Block diagram of a massive MIMO system with two-stage beamforming. 17

Figure 2-7: Performance results for two-stage beamforming a) Network throughput(left), b) UE rate CDF (right). 18

Figure 2-8: UE initial access procedure from [21]. 19

Figure 2-9: Implementation results of the F1-C interface over the OAI platform. 20

Figure 2-10: Output distortion comparisons between original and two modified A-law scheme..... 23

Figure 2-11: Performance comparison of different A-law compression scheme enhancements. 23

Figure 2-12: Example configuration files for CU (left) and DU (right) when applying F1 interface. 24

Figure 2-13: Two-level abstraction provided via FlexRAN+ architecture framework. 26

Figure 2-14: 5G RAN architecture for CU/DU operation. 27

Figure 2-15: F1oIP packet format for sending data to the heterogeneous DUs. 28

Figure 2-16: Experiment setup for heterogeneous DUs in NITOS..... 30

Figure 2-17: Policies Evaluation for different FH transport. 30

Figure 2-18: CPU and MEM utilisation on the CU side for varying number of managed DUs..... 31

Figure 2-19: Policies Evaluation for varying delay on the FH. 31

Figure 2-20: UE RTT times for different technologies for varying delay on the FH. 32

Figure 2-21: NSD description for supporting disaggregated heterogeneous RAN VNFs in OSM..... 33

Figure 2-22: OAI IF4p5 functional split over Sub-6 wireless transport..... 34

Figure 2-23: Software based NIC bonding architecture. 35

Figure 2-24: Single-hop set-up used to perform the bonding tests. Each IEEE 802.11ac WiFi card is MIMO 2x2..... 35

Figure 2-25: Throughput of each WiFi interface transmitting alone. 36

Figure 2-26: UDP and TCP throughput achieved by bonding two interface together. 37

Figure 2-27: UDP throughput degradation due to high queueing interrupt activity that overload the CPU capacity. 38

Figure 2-28: Bonding architecture based on MP-TCP. 38

Figure 2-29: MPTCP throughput measured using different congestion control algorithms and ToS. 39

Figure 3-1. TSON ODL SDN controller. 40

Figure 3-2 Restconf request getting by the OSM module. 41

Figure 3-3 TSON node implemented in Xilinx VC709 FPGA platform..... 42

Figure 3-4 TSON implementation in FPGA embedded in the server..... 42

Figure 3-5 Extended OpenFlow message for TSON. 43

Figure 3-6: General SDN agent design to control a standalone TSON implementation..... 43

Figure 3-7 Estimation of the latency in the TSON network. 44

Figure 3-8. Evaluation of the DO DC and data plane in TSON network..... 45

Figure 3-9: End-to-end packets lost. 46

Figure 3-10: End-to-end measured Jitter. 46

Figure 3-11. TSON network integrated in the 5G OS. 47

Figure 3-12: Flex-E layer between Ethernet MAC and PCS showing the FlexE Shim distribute/aggregate sub-layer in PCS/PMD..... 49

Figure 3-13: Flex-E demonstrator experimental setup..... 49

Figure 3-14: Flex-E allocated slots per channel (4 channels supported)..... 49

Figure 3-15: Tesegine traffic generator configuration example. 50

Figure 3-16: Per Flex-E channel statistics reports. 50

Figure 3-17: Network slicing performance using Flex-E channels over 100G PHY link for vlan4:15slots and vlan 6: 5 slots out of 20. 51

Figure 3-18: Delay and jitter results for two different Flex-E channels. 51

Figure 3-19: Experiment topology. 52

Figure 3-20: Test environment 1. 53

Figure 3-21: Test environment 2. 53

Figure 3-22: Test environment...... 54

Figure 3-23: Test environment. 55

Figure 3-24: NETCONF-based programmability of X-Ethernet switches..... 56

Figure 3-25: Segment routing testbed..... 60

Figure 3-26: Router R1 OSPF configuration. 60

Figure 3-27: OSPF-SR LSA message..... 61

Figure 3-28: R4 routing table (left) - R4 MPLS table (right). 62

Figure 3-29: SR database information for router R4. 62

Figure 3-30: Probes logic. 65

Figure 3-31: OPP pipeline. 65

Figure 3-32: deployment of OPP nodes for handover management. 66

Figure 3-33: XFSM table of the OPP-2 node. 67

Figure 3-34. Deployment scenario for the joint access-backhaul function..... 68

Figure 3-35. Design of the SWAM technical component. 69

Figure 3-36. Testbed used in the evaluation. 70

Figure 3-37. SWAM evaluation of access-backhaul isolation. 71

Figure 3-38. SWAM Gateway relocation evaluation. 71

Figure 3-39. SWAM Mobility evaluation. 72

Figure 4-1. Layer model for IEEE 1588 operation over IEEE 802.11ad with summary of corresponding investigations..... 75

Figure 4-2: Blu Wireless Typhoon platform overview..... 76

Figure 4-3: Transparent clock layer model. 76

Figure 4-4. Cabled setup with direct baseband link between PTP master and slave hosts. 77

Figure 4-5. Point-to-multipoint wireless test setup. 78

Figure 4-6. Transparent clock test setup using BH2s connected directly via coaxial cables. 78

Figure 4-7. Setup for synchronisation performance measurements using standalone master and slave devices communicating through Typhoons in TC mode. 78

Figure 4-8: phc2sys usage for synchronisation between independent Linux time counters. 79

Figure 4-9: Timestamping of PTP event MPDU/MSDU on aggregated 802.11 frames. 81

Figure 4-10: Timestamp fetching latency. 82

Figure 4-11: Maximum time offset under No Ack policy and with normal Ack. 83

Figure 4-12: Maximum time offset with and without ToS and QoS prioritisation on cabled point-to-point setup. 84

Figure 4-13: Histogram of the slave RMS time offset with and without prioritisation. 84

Figure 4-14: Moving RMS and maximum time offset estimations taken at the PTP slave with PTP transport through TC in the course of 24 hours. 85

Figure 4-15: PPS rising edges generated by the PTP Slave (SyncBox/PTPv2) and Master (M600) when directly connected, captured over ~16min with infinite persistence. 85

Figure 4-16: PPS rising edges generated by the PTP Slave (SyncBox/PTPv2) and Master (M600) when connected through two Typhoon TCs, captured over ~16min with infinite persistence. 86

Figure 4-17: Detailed block diagram of the synchronisation harmonizer function. 89

List of Tables

Table 1-1: Summary table with components presented/status.	12
Table 2-1: Original A-law compression scheme.....	21
Table 2-2: Scheme of the first candidate to improve original A-law compression.	22
Table 2-3: Scheme of the second candidate to improve original A-law compression.	22
Table 2-4: OAI IF4p5 required transport capacity.	34
Table 3-1: NETCONF protocol operations.	56
Table 3-2: Basic software components used to build a virtual segment routing testbed.	58
Table 3-3: Segment routing basic terminology.....	59
Table 4-1: Summary of resources used for the IEEE 1588 over IEEE 802.11ad test setup.	77
Table 5-1: Relation between each WP4 functions, the WP3 platform where this function needs to be executed, and the way this function can be integrated with the 5G OS developed in WP5.	90

Executive Summary

This document corresponds to deliverable D4.2, "Complete design and initial evaluation of developed functions" of the Horizon 2020 5G-PICTURE project. To implement the functionalities required by the concepts proposed by 5G-PICTURE, a set of technical components was identified in deliverable D4.1, together with the necessary state of the art analysis per technology domain. This deliverable aims to provide the complete specification of these network functions (developed in the context of Task 4.1, Task 4.2 and Task 4.3 activities), as well as an initial evaluation of their performance. The technical components are relevant to a) Virtual Network Functions (VNFs) and Physical Network Functions (PNFs) for dynamic 5G-RAN deployments (Task 4.1); b) Transport Slicing for converged wired/wireless FH/BH networks and integration with 5G-PICTURE orchestrator (Task 4.2); and c) PNFs and VNFs to support synchronisation services in converged FH/BH networks (Task 4.3). The technical components description presented in this document will identify what is the status of the development and evolution of the technology enablers presented in 5G-PICTURE.

In the case where a platform for data plane programmability, programming models and hardware abstractions developed in WP3 is used to support a network function deployment, the necessary pointer to deliverable D3.2 is introduced. In the case where no specific/specialised hardware platform development was required to support the deployment of some network functions, the details of the software solution are described in this document. In this deliverable we also provide the technical means through which the designed network functions are exposed and become available to the orchestration and control plane comprising the 5G-PICTURE Operating System (OS) developed in WP5.

For each network function, the complete performance evaluation together with a service chain integrating some of the developed functions in Task 4.1, Task 4.2 and Task 4.3, namely network functions relevant to functional splits, transport network slicing, and synchronisation primitives will be presented in deliverable D4.3.

1 Introduction

The 5G-PICTURE project focuses on developing integrated transport network solutions, addresses limitations and support novel Disaggregated-RAN (DA-RAN) and Cloud-RAN (C-RAN) architectures, as they are defined by 3GPP [1][2]. 5G-PICTURE proposes a novel architecture based on the flexible functional splits design paradigm. The optimal “split” can be flexibly decided, based on a number of factors such as the transport network and the service characteristics.

The project adopts the concept of transport network slicing, resource and service virtualisation across technology domains, while another dimension is related to the development of a unified programmable control and management framework called 5G OS, used to coordinate the underlying heterogeneous technology domains and support orchestration and end-to-end service provisioning across the various infrastructure domains. This deliverable is related to all Tasks in WP4, namely VNFs and PNFs for dynamic 5G RAN deployments (Task 4.1), Transport slicing for converged wired-wireless FH/BH networks and integration with 5G-PICTURE Orchestrator (Task 4.2), and PNFs and VNFs to support synchronisation services in converged FH/BH networks (Task 4.3). The goal of deliverable D4.2 is twofold. On one hand, this deliverable aims to analyse and evaluates a number of technical components defined in deliverable D4.1 related to each task. On the other hand, it aims to provide insights on how each technical component can be integrated within the 5G-PICTURE 5G-OS delivered in the context of WP5 activities.

In deliverable D4.1 [3], a number of 16 technical components have been described by the project partners. Table 1-1 shows the presented and analysed technical components in deliverable D4.2. Three technical components regarding synchronisation services are also depicted. However these were reprioritised and will be presented in deliverable D4.3.

For each technical component in deliverable D4.2 the detailed aspects related to the project are investigated. This investigation essentially looks into the description of how the technical component functions are implemented and the used evaluation methodology and experimentation/evaluation results related to KPIs declared in deliverable D4.1 [3].

Table 1-1: Summary table with components presented/status.

Technical Component		Delivered In
Section 2: VNFs and PNFs for Dynamic 5G-RAN deployments		
1	Optimal functional split	D4.2
2	Implementation of functional split using OpenAirInterface (OAI) platform	D4.2
3	Flexible Functional Splits	D4.2
4	Disaggregated Heterogeneous Base Station functionality (declared as LTE/5G RAN as VNFs implementation in D4.1)	D4.2
5	Wireless Transport Technologies with Functional Split Support	D4.2
Section 3: Transport slicing for converged wired-wireless FH/BH networks		
6	Time Shared Optical Network (TSON)	D4.2
7	Flex-E	D4.2
8	X-Ethernet	D4.2
9	Segment routing for enhanced VPN	D4.2
10	Open Packet Processing (OPP)	D4.2
11	Solution based on IEEE 802.11 technologies, both for access and BH (802.11ac modems)	D4.2
Section 4: PNFs/ VNFs to support synchronisation services in converged FH/BH		
12	IEEE 1588 over IEEE 802.11ad	D4.2
13	IEEE 1588 over off-the-shelf IEEE 802.11ac	D4.3
14	Heterogeneous synchronisation transport testbed	D4.3
15	Synchronisation harmonizer (Evaluate in D4.3)	D4.3
16	Over-the-air synchronisation for FH networks	D4.3

In the case that platforms/APIs built in the context of WP3 are used to implement a function a pointer to deliverable D3.1 [4] or forthcoming deliverable D3.2 [5] is provided, while if no platform from WP3 is used, a description of the platform or used is described inline. Furthermore, for each technical component we elaborate how its functions can be packaged for the 5G OS, sketching the descriptor that will be used to onboard this function to the 5G OS.

Note that technical component 14 (Table 1-1) refers to the wireless testbed of the NITOS facility that will be exploited for the evaluation of the IEEE 1588 (PTP) synchronisation technique. Other technical components will be tested in the NITOS testbed and evaluated over multiple wireless technologies (Sub-6 GHz and mmWave), using either the Sub-6 GHz or 60 GHz spectrum. Furthermore, integration of the synchronisation harmonizer that is able to work across the different domain technologies is planned for deliverable D4.3 when the relevant WP3 platforms are ready to be interconnected and feature IEEE 1588.

Organisation of the document

Section 2 describes the technical components for VNFs and PNFs for Dynamic 5G-RAN deployments.

Section 3 describes the technical components for transport slicing for converged wired-wireless FH/BH networks.

Section 4 describes the technical components for PNFs and VNFs to support synchronisation services in converged FH/BH networks.

Section 5 presents integration plans.

Section 6 concludes the deliverable and describes the next steps.

2 VNFs and PNFs for Dynamic 5G-RAN deployments

In this section, we extend our initial study among several technical components introduced in deliverable D4.1 [3] for the 5G-RAN aspect. The design and development of these RAN network functions aim to provide the radio access functionality required by the 5G-PICTURE infrastructure provider exploiting telecommunication and cloud resources, while also they can be exposed to 5G-PICTURE tenants in a dynamic and efficient manner as a service. Notice that we reuse some applied terminologies and technologies introduced in deliverable D4.1 [3], such as the central unit (CU), distributed unit (DU), radio unit (RU), fronthaul (FH), midhaul (MH), and the defined RAN functional splits by the third generation partnership project (3GPP) and enhanced common public radio interface (eCPRI). To recap these introduced terminologies and compare with the terminologies adopted by different organisations, we provide an example RAN deployment topology in Figure 2-1 with the three-tier entities: RU, DU and CU. We can observe that different terminologies are applied by different standardisation bodies and industry fora as summarised in the table shown in Figure 2-1, such as 3GPP [6][7][1], next generation mobile network (NGMN) alliance [8], CPRI forum [9], next generation FH interface (NGFI) [10], IEEE 1914 [11], xRAN forum [12], telecom infra project (TIP) [13]. Moreover, there are two tiers of possible functional splits between these entities, i.e., the high level split (HLS) between DU and CU and the low level split (LLS) between RU and DU. Currently, 3GPP standardizes the F1/V1 interface between the radio link control (RLC) and packet data convergence protocol (PDCP) sublayers for 5G/4G, as the solution for the HLS. On the other hand, several possible physical layer functional splits (e.g., 3GPP functional splits option 6, 7, and 8 defined in [14]) are still surveyed for the LLS.

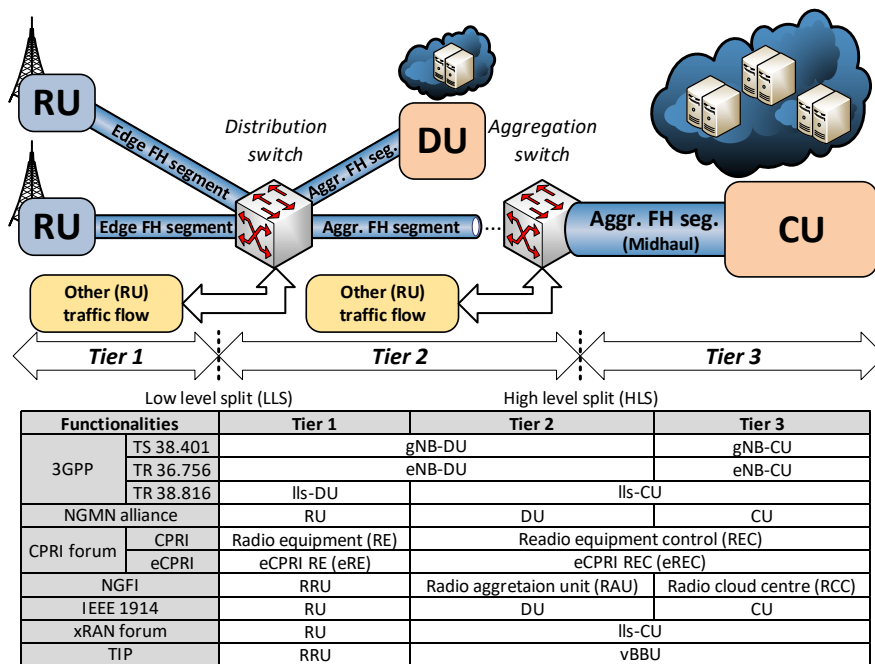


Figure 2-1: RAN network topology for three-tier RAN entities.

Based on the above recap on the RAN domain technologies, in the following paragraphs, we elaborate on the design and development status of the technical components introduced in deliverable D4.1 [3]. An overview of these components is as follows. In subsection 2.1, we focus on the low level physical layer functional split and introduce the derivation of optimal functional split of the RAN entities for specific use case. In subsection 2.2, we update the development status on the functional split over the OpenAirInterface (OAI) platform. In subsection 2.3, the control framework for flexible functional split is updated with preliminary results. Moreover, we provide the development and evaluation status when applying heterogeneous radio access technology (RATs), such as Wi-Fi and LTE, at the DU level in subsection 2.4. We provide the latest evaluation outcome when applying Sub-6 GHz technologies as the FH transportation in subsection 2.5. For each technical component, we further detail in terms of (1) how these derived VNFs/PNFs can utilise the underlying programmable platforms and APIs provided by tasks of WP3, and (2) how these functions can be packed and managed by the 5G OS developed in WP5. Note that some technical components will be developed continuously, and thus the more updates will be given in the future deliverable D4.3.

2.1 Technical Component 1: Optimal functional split

2.1.1 Summary Description

Massive MIMO is a promising technology to improve the spectral efficiency (SE) and energy efficiency (EE) of transmission in 5G RAN. It involves equipping the transmitter with large number of antennas, which enables it to generate narrow, high-energy beams and to serve multiple users at the same time [15]. Despite its many advantages, providing a transport link to massive MIMO-based RUs is a challenge. CPRI-based FH is not feasible as the transport requirements would impose an excessively high transport network data rate. Therefore, split 7-2 is considered for massive MIMO in which beamforming is moved fully or partially to RU. In this technical component, we optimize the access network parameters such as transmit powers of UEs and downlink beamformers for split 7-2 to improve the SE and EE.

2.1.2 Used programmable platforms and APIs

The active antenna distributed unit (AADU) from Airrays [4] is the platform intended for this technical component. It is a rectangular antenna array with up to 64 antenna elements. The elements are spaced 0.5λ and 0.7λ in the horizontal and vertical direction, respectively. Since the AADU is still under development, the integration results are not provided in this deliverable and will be reported in deliverable D4.3. The beamforming algorithms developed in this technical component will be implemented to some extent in the massive MIMO but not fully owing to the lack of channel estimation on the platform as it is done at the BBU, required interfaces, and limited computational capability. Loading the beamforming weights that are computed offline/in BBU is being explored.

2.1.3 Function Design/Implementation/Evaluation

We now discuss the design of algorithms to optimise the downlink transmit powers of UEs and partial beamforming at DU and RU. We then present some initial evaluation results.

Optimising Allocation of UE Transmit Powers

As mentioned before, beamforming is offloaded to RU in split 7-2. This avoids the scaling of FH capacity with the number of antennas. However, the gains from coordinated beamforming across multiple cells is limited as the beamforming weights are computed by RU. Therefore, we focus on optimising the powers assigned to different UEs based on the long-term channel gains. This avoids the need for feeding the channel gains to DU thus, further reducing the FH overhead.

Figure 2-2 (a) shows the system model, which involves a DU, multiple massive MIMO-enabled RUs, and several UEs. The RUs are connected to DU through capacity-limited FH. As shown in Figure 2-2 (b), each link between RU and UE is characterised by its signal-to-interference-plus-noise ratio (SINR) and the power assigned to the UE (P_k). Also, note that the data rate to any RU is constrained to less than the capacity of the FH link denoted as C , which is determined by the FH technology. For this system, we formulate SE and EE maximisation problems (Figure 2-3):

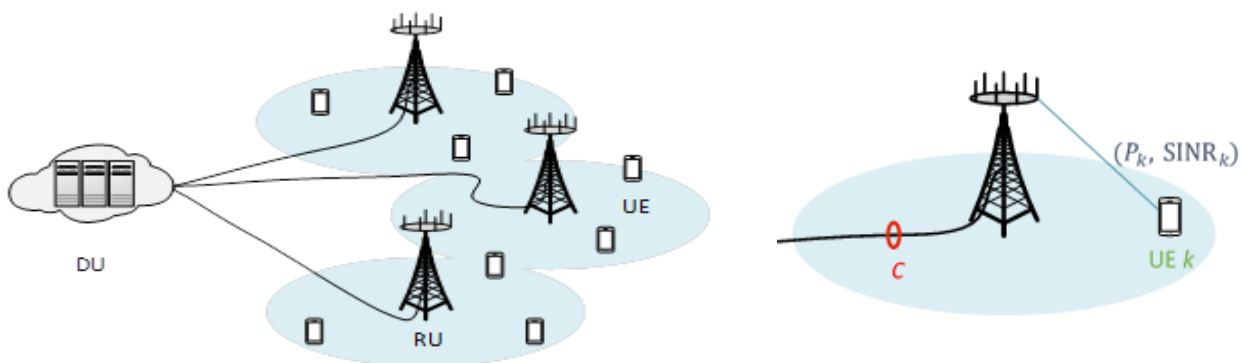


Figure 2-2: System model. a) Multi-cell massive MIMO cellular system (left) b) An individual RU (right).

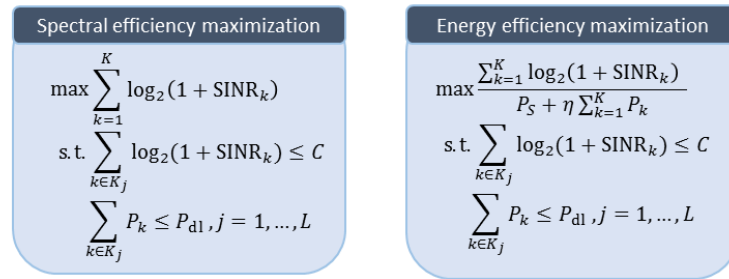


Figure 2-3: SE and EE maximisation problems.

These optimisation problems are non-convex and finding optimal UE powers is hard. We employ successive convex approximation (SCA) framework to find the optimal transmit powers. In it, a simpler convex optimisation problem is solved in an iterative manner. The complexity of the algorithm is polynomial in the number of users and the algorithm monotonically improves the SE/EE and finds a local optimum. The details of the algorithm can be found in [16].

Simulation Results: We simulate a 7-cell hexagonal cellular layout with wrap around and 70 UEs are randomly dropped in the network area. We consider time division duplex transmission. The pilot sequences for uplink training are allocated randomly to UEs and are reused in each cell. For beamforming at RU, we consider maximal ratio transmission (MRT) and zero forcing (ZF). While the former tries to maximise the received signal power at UE, the latter attempts to cancel out interference from other UEs in the cell. We compare the performance of the proposed SCA approach against a baseline scheme in which transmit powers are not coordinated across RUs.

Figure 2-4 (a) and Figure 2-4 (b) show the results of network throughput, which is the sum of the users' SEs averaged over several channel realisations for MRT and ZF beamforming, respectively. We see that the SCA approach achieves a significantly higher network throughput than the baseline scheme. For example, the SCA approach improves the throughput by 54%, 29%, and 25% at $C_{\text{plc}} = 200, 300,$ and 400 Mb/s, respectively, for MRT beamforming. The gains for ZF beamforming at $C_{\text{plc}} = 500, 700,$ and 900 Mb/s are 59%, 38%, and 34%, respectively.

We see that the network throughput increases as the FH capacity increases before saturation. This is because the FH constraints become more relaxed as the FH capacity increases and eventually, the throughput is determined only by the power constraints. We also see that ZF beamforming achieves a significantly higher throughput than MRT beamforming for all the schemes. This is because the former manages the interference better than the latter.

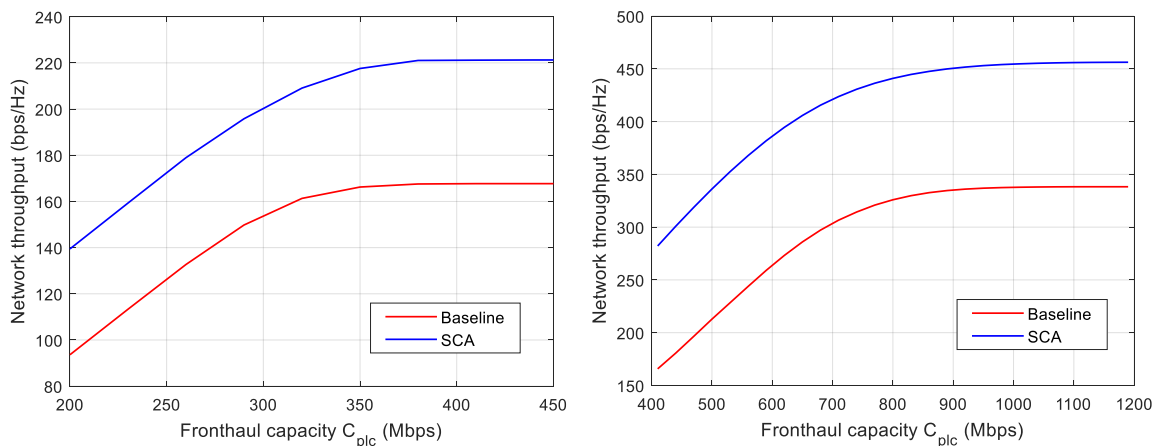


Figure 2-4: Network throughput as a function of FH capacity a) MRT (left) b) ZF (right).

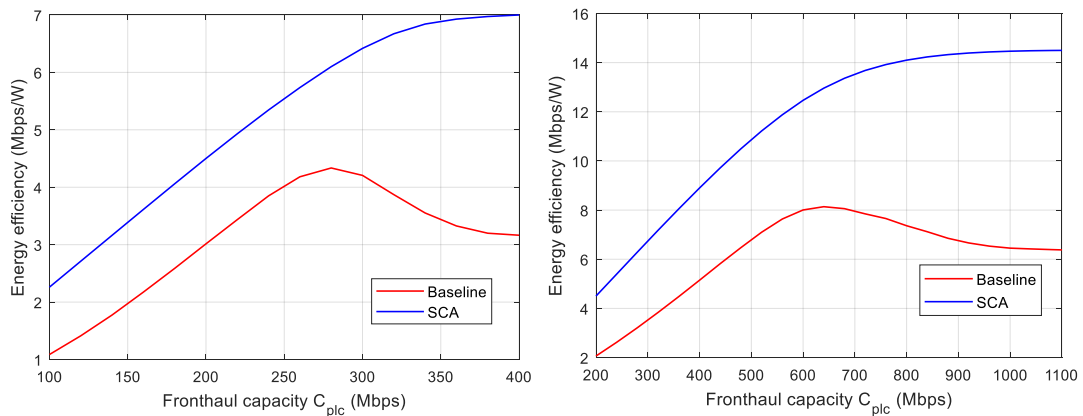


Figure 2-5: Energy efficiency as a function of FH capacity a) MRT(left) b) ZF(right).

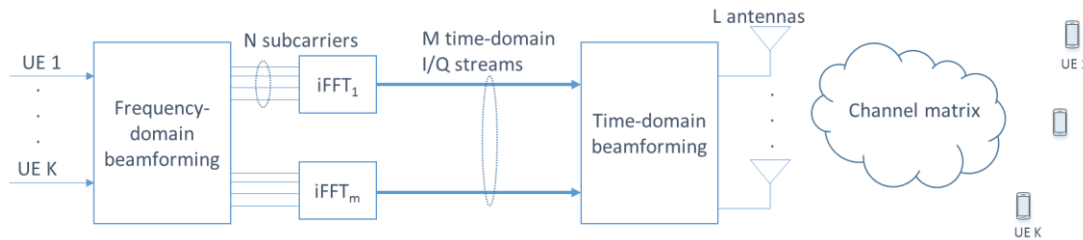


Figure 2-6: Block diagram of a massive MIMO system with two-stage beamforming.

Figure 2-5 (a) and Figure 2-5 (b) show the energy efficiency results of MRT and ZF beamforming, respectively. We see that the SCA approach achieves a significantly higher EE than the baseline scheme. For example, the SCA approach improves EE by 49%, 53%, and 122% at $C_{plc} = 200, 300,$ and 400 Mb/s, respectively, for MRT beamforming. The gains for ZF beamforming at $C_{plc} = 400, 600,$ and 800 Mb/s are 73%, 56%, and 91%, respectively.

For the SCA approach, we see that EE increases as C_{plc} increases before saturating. However, for the baseline scheme, EE first increases and then decreases. This is because the RUs transmit at higher powers as FH capacity increases. Beyond a certain point, however, this increase in transmit power does not translate to an increase in sum rate.

Two-stage Beamforming

Two-stage beamforming is an approach to obtain some of the gains of coordinated beamforming without overwhelming the FH link. This is achieved by using coordinated beamforming in one stage. In addition, it lowers the computational requirements. As the name implies, beamforming is performed in two stages, partly before the IFFT operation and partly after. In the latter, beamforming is applied to the time-domain I/Q samples. Hence, it is also referred to as time-domain beamforming. This stage of beamforming is a function of only the long-term channel gains; hence, needs to be adapted at a much slower time-scale reducing the computational requirements at RU. On the other hand, the beamforming before the IFFT operation is different for different physical resource blocks (PRBs). Hence, it is referred to as frequency-domain beamforming. The block diagram is shown in Figure 2-6.

The FH overhead resulting from sending beamforming weights is higher than a fully distributed beamforming but lower than a fully coordinated beamforming. This is because the effective channel matrix dimension is lowered by time-domain beamforming.

Designing Beamforming Matrices: For frequency-domain beamforming, we employ the signal-to-leakage-plus-noise-ratio (SLNR) beamforming. Here, the matrices are computed from the lower dimensional effective channel, which is the combination of time domain beamforming and the actual channel matrix. For time-domain beamforming, we employ the Eigen vectors of the sum of the channel covariance matrix of UEs as the columns of the beamforming matrix [17]. This choice of beamformers relies on the fact that the channel covariance matrix is the same for all the PRBs.

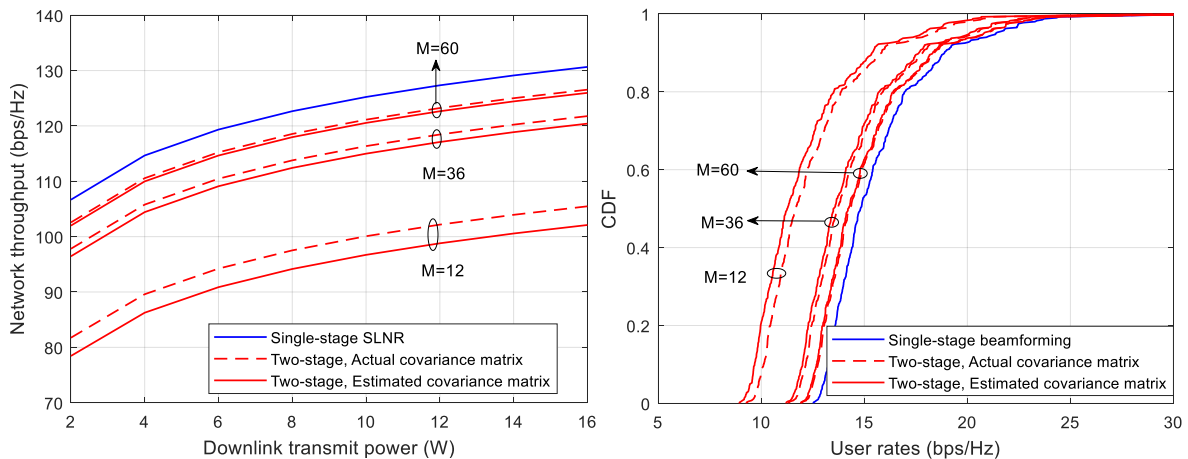


Figure 2-7: Performance results for two-stage beamforming a) Network throughput(left), b) UE rate CDF (right).

The RU may not know the channel covariance matrix of UEs. Hence, we develop a method to estimate the time-domain beamforming matrix from uplink Sounding Reference Signal (SRS) measurements. To have an appropriate estimation, we assume channel reciprocity. It can be shown that the covariance matrix of the uplink received signal is the sum of channel covariance matrices of UEs and a scaled identity matrix due to thermal noise. Hence, the Eigen vectors of the uplink received signal is same as the Eigen vectors of the sum of the channel covariance matrix of UEs. We use sample covariance matrix, which can be efficiently computed from uplink measurements, for the covariance matrix of uplink received signal.

Simulation Results: We simulate single-cell network with $L=96$ antenna RU serving $K=8$ UEs in the downlink. A standard channel covariance model for massive MIMO antenna array is considered [18]. We compare the performance of two-stage beamforming against single-stage SLNR beamforming, which is an upper limit on the performance for the former.

Figure 2-7 (a) plots the network throughput as a function of the downlink transmit power. We see that performance of two-stage beamforming increases as the number of time-domain I/Q streams increases and approaches the single-stage SLNR curve. When the number M of time-domain I/Q streams increases to $M=60$, the loss compared to single-stage SLNR is only 3%. The network throughput with estimated covariance matrix is lower than the case covariance matrix is perfectly known. However, this loss decreases as M increases. Similar trends are observed in the user rate CDFs in Figure 2-7 (b).

2.1.4 Packaging for the 5G OS

The beamforming and UE power allocations for AADU are control plane functions. The AADU as a PNF is configured with the NETCONF server in it and a NETCONF client at the CU. The service descriptor includes RF transmit power, number of carriers used, number of turned ON antenna elements (RSUs), compute and storage capability of AADU.

2.2 Technical Component 2: Implementation of functional split using the OAI platform

2.2.1 Summary Description

As for the second technical component, we focus on the implementation of functional split using the OAI platform. Currently, following functional splits are now supportable in the OAI platform:

1. Low physical functional split: 3GPP functional split option 8 and option 7-1.
2. Functional split between physical layer (PHY) and medium access control (MAC) layers: 3GPP functional split option 6, also known as nFAPI defined by the small cell forum [19].
3. High level functional split between PDCP and RLC: 3GPP functional split option 2, also known as F1 functional split defined in [20][21].

Since we already detail our implementation work on the 3GPP split option 8, 7-1 and 6 in the deliverable D4.1 [3]; therefore, we here present the updated status only in terms of (a) the compression scheme for the low

physical functional split, and (2) the latest implementation on the F1 interface aligned with the recent standardisation activities by 3GPP.

2.2.2 Used programmable platforms and APIs

These functional splits are implemented over the OAI platform, through which the RAN functions can be executed in the x86 infrastructure with the connected Software-Defined Radio (SDR) in order to connect the commercial-of-the-shelf (COTS) user equipment (UE). To program the applied functional split, the corresponding configuration files for each functional split is used. Note that the deployment can either be split-specific or flexible-split deployment. The former one only deploys necessary functions at the corresponding entity, while the later one deploy all network functions to support a flexible change of functional split between RAN entities.

2.2.3 Function Design/Implementation/Evaluation

2.2.3.1 Functional split development update over F1 interface

Currently, 3GPP has standardised the F1 interface in terms of how to realise the disaggregated deployment. More specifically, this interface can be decomposed into two different interfaces: (1) F1 control plane (F1-C), and (2) F1 user plane (F1-U) interfaces.

The F1-C protocol relies on the F1AP (F1 application protocol) over the underlying stream control transmission protocol (SCTP) and Internet protocol (IP). It can provide several functionalities: (a) interface management (e.g., F1 setup, reset, error indication, gNB-DU/CU configuration update), (b) system information management, (c) UE context management (e.g., UE context setup, release, modification and so forth), (d) RRC message transfer, (e) paging, (f) warning message transfer. On the other hand, the F1-U uses the GPRS tunnelling protocol user plane interface (GTP-U) over user datagram protocol (UDP) and IP in order to transfer user data or do the flow control. In the following, we show how the F1 interface is used to establish the UE initial access in Figure 2-8. We can see that the gNB-DU and gNB-CU will exchange the information in between to establish the RRC connection. In particular, the RRC message will be encapsulated and exchanged in both downstream and upstream directions (cf. steps 2, 3, 6, 13, 14 and 17), and the UE context can be setup using the steps 9 and 11.

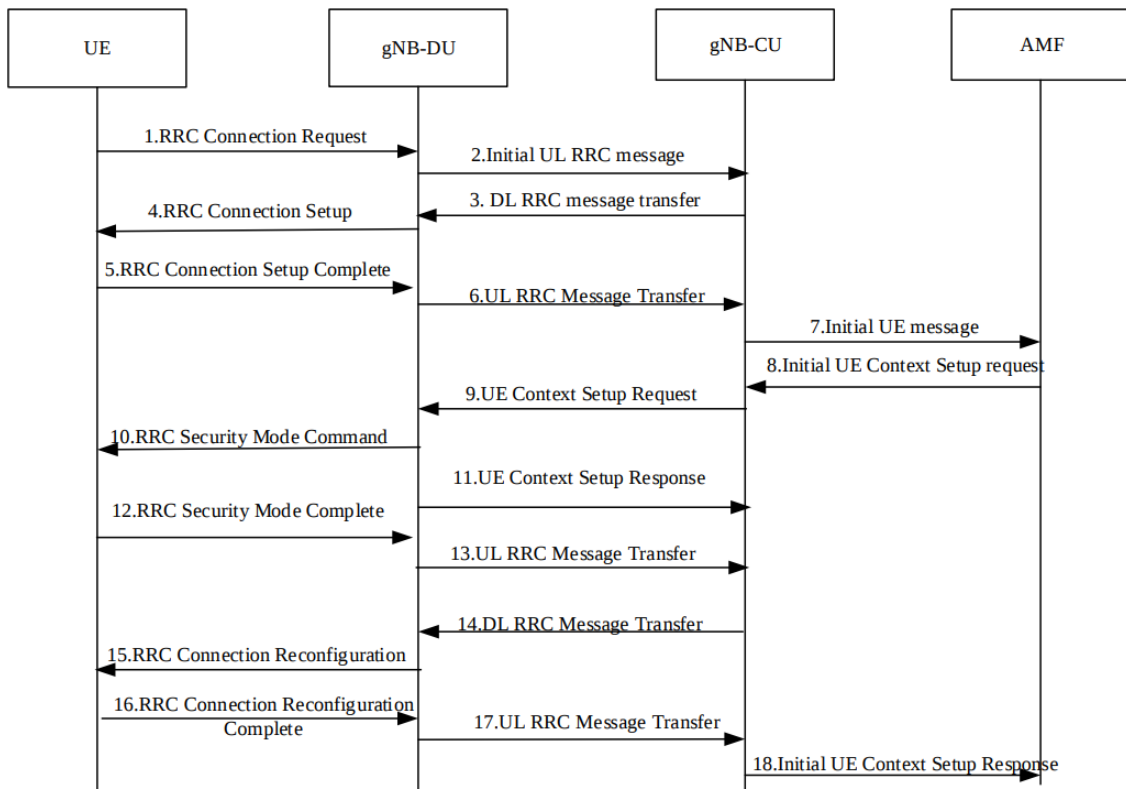


Figure 2-8: UE initial access procedure from [21].

To show our implementation outcome, the results of the corresponding UE initial access steps is shown in Figure 2-9 using the Wireshark trace gathered by the deployed CU entity. We can observe that after F1SetupRequest and F1SetupResponse messages, the corresponding RRC message transfers are initiated between CU and DU (i.e., InitialULRRCCMessageTransfer, DLRRCCMessageTransfer and ULRRCCMessageTransfer) to setup the RRC connection between the user and the CN. And then the RRC messages can be exchanged between CU and DU consecutively. To sum up, the UE can now use our implemented F1 interface to establish the UE connection, as the legacy monolithic BS.

As the next step, we plan to add some further extensions to our current implementation work:

- Current implemented F1 interface is done over the LTE system. According to the current study by 3GPP in [22], the extra V1 interface will be introduced to the LTE system.
- Some messages over the F1 interface will be further implemented according to the latest standardisation activities.
- Current F1-U interface is implemented over the protobuf messages over the UDP/IP protocol, and we expect future development to include the standardised GTP-U transportation over the top.

To this end, in the deliverable D4.3, we will explore more results for such functional split implementation over the OAI platform.

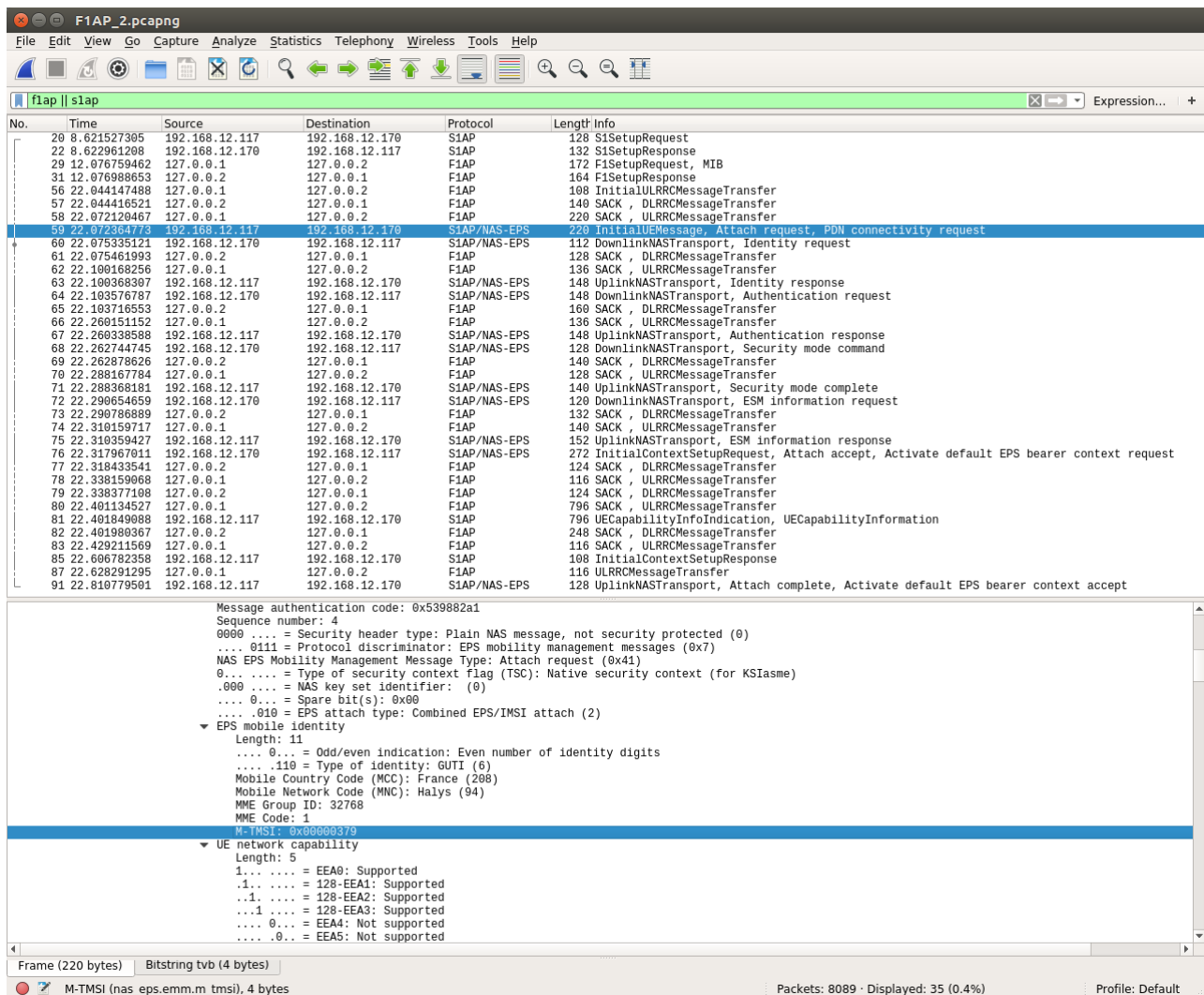


Figure 2-9: Implementation results of the F1-C interface over the OAI platform.

2.2.3.2 Data sample compression over the low physical functional split

Moreover, we explore the enhancement of the A-law compression for the low physical functional split. As we previously explained in deliverable D4.1, the data sample compression can bring significant gains to reduce the FH data rate; however, it will also introduce extra impact on the user plane performance. In practice, such A-law compression approach will compress the incoming 16-bit sample into an 8-bit output (1 sign bit, 3 exponent bits and 4 mantissa bits), and then expand it back to the original 16-bit level. However, it will introduce severe expansion error when the input signal amplitude is large as summarised in Table 2-1. For instance, when the input signal is either from -2^{15} to $-2^{14}-1$ or from 2^{14} to $2^{15}-1$, the largest expansion error will generate $512/2^{15} = 1.56\%$ signal distortion. Although this high-amplitude input signal rarely appears in the average case; the associated distorted value will impact all other sub-tones significantly due to the spreading effect introduced by the (inverse) digital Fourier transform (DFT/IDFT) operations. Hence, all symbols of the same time index will be influenced by this large expansion error. What is worst is that such high-amplitude inputs will show up more frequently due to the aforementioned high peak-to-average power ratio (PAPR) characteristic for the orthogonal frequency division multiplexing (OFDM). To this end, we investigate some possible approaches that can be applied for the improvements, while still maintaining the benefits brought by the FH data sample compression scheme, e.g. FH throughput reduction and few compression/decompression time.

Table 2-1: Original A-law compression scheme.

Input (16 bits)		Output (8 bits)			Expansion error	
From	To	Sign (1 bit)	Exponent (3 bits)	Mantissa (4 bits)	Minimum	Maximum
-2^{15}	$-2^{14}-1$	1	7	From 0 to 15	-511	512
-2^{14}	$-2^{13}-1$		6		-255	256
-2^{13}	$-2^{12}-1$		5		-127	128
-2^{12}	$-2^{11}-1$		4		-63	64
-2^{11}	$-2^{10}-1$		3		-31	32
-2^{10}	-2^9-1		2		-15	16
-2^9	-2^8-1		1		-7	8
-2^8	-1		0		-7	8
0	2^8-1	0	0		-7	8
2^8	2^9-1		1		-7	8
2^9	$2^{10}-1$		2		-15	16
2^{10}	$2^{11}-1$		3		-31	32
2^{11}	$2^{12}-1$		4		-63	64
2^{12}	$2^{13}-1$		5		-127	128
2^{13}	$2^{14}-1$		6		-255	256
2^{14}	$2^{15}-1$		7		-511	512

The first applicable approach is to dynamically quantise the incoming data samples when receiving a chunk of input samples, e.g., each OFDM symbol or each time slot, based on the provisioned FH link capacity. That is to say, we can replace the (de-)compression unit with the FH (de-)quantisation unit for this purpose. This approach will further include the quantisation control information to be packed together with the sub-header to facilitate the data sample recovery. Nevertheless, the optimal quantisation approach may take too much time to find the best solution, and it might take even longer time than the data sample (de-)compression time. To this end, we only consider the use of a uniform linear quantizer that can be finished in a much shorter time period.

As the second approach, we can improve the original A-law compression via adding the extra bits for the mantissa part, while reducing the number of bits used for the exponent part. The reason behind this is that the 4-bit mantissa part in Table 2-1 is not enough for the high-amplitude inputs, which will generate significant distortion to all sub-tones. Note that reducing the exponent part may also decrease the resolution especially when the input signal amplitude is small. Hence, our aim here is to find the trade-off between the signal distortion for high- and low-amplitude inputs and we propose two possible candidates as summarised in Table 2-2 and Table 2-3, respectively. The first candidate directly uses one more bit for the mantissa part from the original exponent part in order to decrease the expansion error for a wide range of high-amplitude inputs (i.e., from -2^{15} to $-2^{12}-1$ or from 2^{12} to $2^{15}-1$); however, the low-amplitude inputs will suffer significantly. On the

other hand, the second candidate strikes a balance in this trade-off via only reducing the expansion error for a smaller but critical region (i.e., from -2^{15} to $-2^{13}-1$ or from 2^{13} to $2^{15}-1$) and giving fewer side-effects to the low-amplitude inputs.

Table 2-2: Scheme of the first candidate to improve original A-law compression.

Input (16 bits)		Output (8 bits)			Expansion error	
From	To	Sign (1bit)	Exponent (2bits)	Mantissa (5bits)	Minimum	Maximum
-2^{15}	$-2^{14}-1$	1	3	From 0 to 31	-255	256
-2^{14}	$-2^{13}-1$		2		-127	128
-2^{13}	$-2^{12}-1$		1		-63	64
-2^{12}	-1		0		-63	64
0	$2^{12}-1$	0	0		-63	64
2^{12}	$2^{13}-1$		1		-63	64
2^{13}	$2^{14}-1$		2		-127	128
2^{14}	$2^{15}-1$		3		-255	256

Table 2-3: Scheme of the second candidate to improve original A-law compression.

Input (16 bits)		Output (8 bits)			Expansion error	
From	To	Sign (1bit)	Exponent (2bits)	Mantissa (5bits)	Minimum	Maximum
-2^{15}	$-2^{14}-1$	1	3	From 0 to 31	-255	256
-2^{14}	$-2^{13}-1$		2	From 0 to 31	-127	128
-2^{13}	$-2^{12}-1$		1	From 16 to 31	-127	128
-2^{12}	$-2^{11}-1$		1	From 0 to 15	-63	64
-2^{11}	$-2^{10}-1$		0	From 16 to 31	-63	64
-2^{10}	-1		0	From 0 to 15	-31	32
0	$2^{10}-1$	0	0	From 0 to 15	-31	32
2^{10}	$2^{11}-1$		0	From 16 to 31	-63	64
2^{11}	$2^{12}-1$		1	From 0 to 15	-63	64
2^{12}	$2^{13}-1$		1	From 16 to 31	-127	128
2^{13}	$2^{14}-1$		2	From 0 to 31	-127	128
2^{14}	$2^{15}-1$		3	From 0 to 31	-255	256

A visual comparison of these two candidates with the original scheme in terms of the output expansion error is shown in Figure 2-10. We can observe that both candidates can reduce the largest error as $256/2^{15} = 0.78\%$, while the second candidate exhibits two-times smaller error when the input signal is from -2^{11} to $2^{11}-1$. Finally, one may be curious to know whether we can continually increase the mantissa part by 1 bit to further reduce the largest error for the peak input signal. Nevertheless, adding one more bits for the mantissa part will lead to a constant expansion level between $[-127, 128]$ for all inputs, and thus it will become a linear uniform quantizer, which will be examined in the first approach.

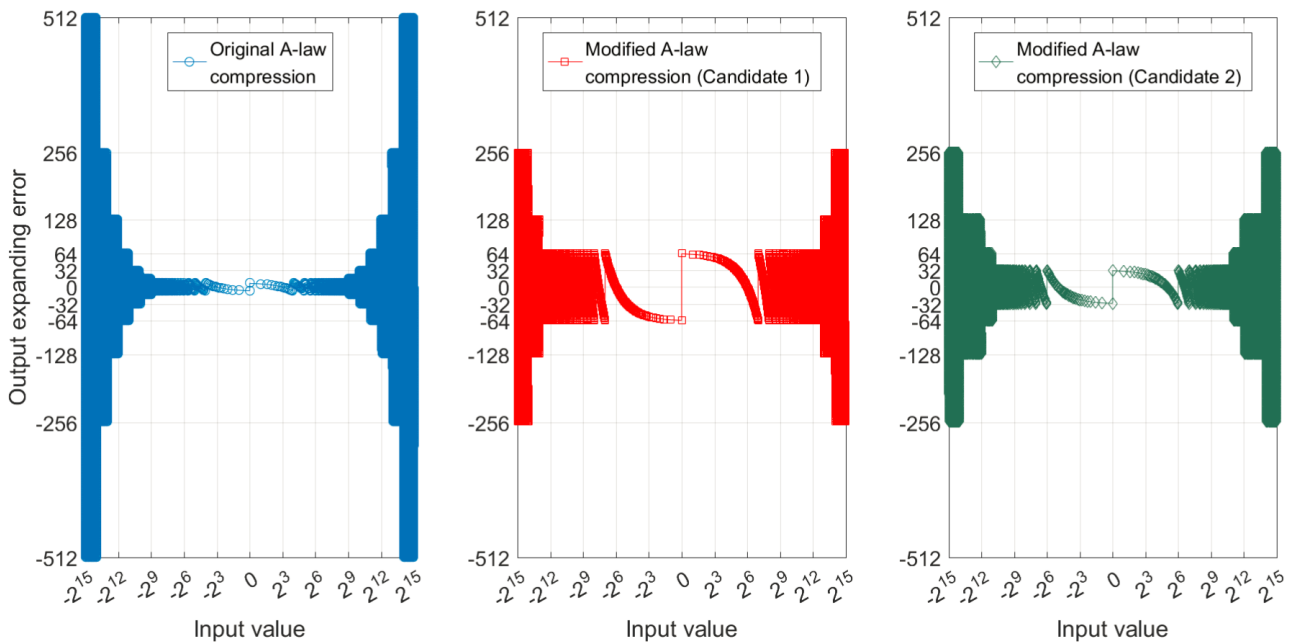


Figure 2-10: Output distortion comparisons between original and two modified A-law scheme.

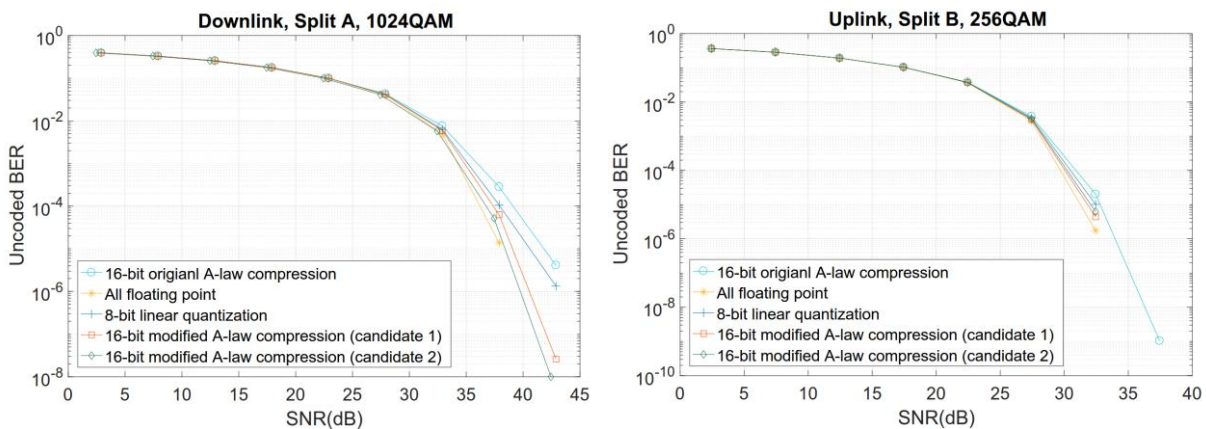


Figure 2-11: Performance comparison of different A-law compression scheme enhancements.

In the following, we show the performance of these two approaches over two extreme high-order modulation cases: (1) the downlink (DL) direction using 3GPP split option 8 with 1024QAM, and (2) the uplink (UL) direction using 3GPP split option 7-1 with 256QAM. These high-order modulations make the expansion error become a performance dominating factor. In Figure 2-11, we can see that the original 16-bit A-law compression scheme has a large gap toward the all floating point receiver, i.e., using 16 bits without any compression. Specifically, the gap is approximately 2.9 dB and 1.2 dB when we see the level with 10^{-4} uncoded bit error rate. If we apply the first approach, i.e., using an 8-bit linear uniform quantizer as the replacement, it can reduce the gap by 1.1 dB and 0.4 dB, respectively for two cases. Further, if we examine the second approach via modifying the original A-law compression scheme, both candidates can show significant performance enhancement. Specifically, when compared with the original A-law compression scheme, the first candidate can bring 1.7 dB and 0.8 dB gain, and the second candidate will generate 2.3 dB and 0.7 dB gain, respectively. Such gains not only justify the claim that the performance bottleneck of the original A-law compression scheme is at the high-amplitude inputs, but also indicate a better performance can be achieved via properly trade-off between high- and low-amplitude expansion errors. To conclude, these possible approaches can replace the original A-law compression scheme and still maintain the aforementioned the compression benefits.

2.2.4 Packaging for the 5G OS

The functional split can be configured by the RAN-domain controllers according to the optimal functional split decided by the technical components of Section 2.1 and the capability of underlying RAN entity. To control the functional split, the 5G OS shall configure the necessary configuration for both CU and DU entities. For instance, the example configuration files of CU and DU, when applying the F1 interface, are depicted in Figure 2-12. Note that, in this example, both RU, DU and CU are deployed in the same physical machine and different configurations can be applied when deploying them at different machines.

<pre>Active_eNBs = ("eNB-CU-Eurecom-LTEBox"); eNBs = ({ ////////// Identification parameters: eNB_ID = 0xe00; cell_type = "CELL_MACRO_ENB"; eNB_name = "eNB-CU-Eurecom-LTEBox"; tr_s_preference = "f1" local_s_if_name = "lo"; remote_s_address = "127.0.0.1"; local_s_address = "127.0.0.2"; local_s_portc = 60001; remote_s_portc = 60000; local_s_portd = 60011; remote_s_portd = 60010; ////////// MME parameters: mme_ip_address = ({ ipv4 "192.168.12.170"; ipv6 "192:168:30::17"; active "yes"; preference "ipv4"; }); NETWORK_INTERFACES : { ENB_INTERFACE_NAME_FOR_S1_MME = "en01"; ENB_IPV4_ADDRESS_FOR_S1_MME = "192.168.12.117/24"; ENB_INTERFACE_NAME_FOR_S1U = "en01"; ENB_IPV4_ADDRESS_FOR_S1U = "192.168.12.117/24"; ENB_PORT_FOR_S1U = 2152; # Spec 2152 }; };);</pre>	<pre>Active_eNBs = ("eNB-Eurecom-DU"); eNBs = ({ ////////// Identification parameters: eNB_CU_ID = 0xe00; eNB_name = "eNB-Eurecom-DU"; }); MACRLCs = ({ num_cc = 1; tr_s_preference = "local_L1"; tr_n_preference = "f1"; local_n_if_name = "lo"; remote_n_address = "127.0.0.2"; local_n_address = "127.0.0.1"; local_n_portc = 60000; remote_n_portc = 60001; local_n_portd = 60010; remote_n_portd = 60011; });); RUs = ({ local_if_name = "lo"; remote_address = "127.0.0.2"; local_address = "127.0.0.1"; local_portc = 50000; remote_portc = 50000; local_portd = 50001; remote_portd = 50001; local_rf = "yes" nb_tx = 1 nb_rx = 1 att_tx = 0 att_rx = 0; eNB_instances = [0]; }););</pre>
--	--

Figure 2-12: Example configuration files for CU (left) and DU (right) when applying F1 interface.

2.3 Technical Component 3: Flexible Functional Splits

2.3.1 Summary Description

As for this technical component, its aim is to flexibly compose the logical BS from different applied RAN functional splits between disaggregated RAN entities. This logical BS contains all necessary functionality of a BS, e.g., LTE eNB or NR gNB, and it can be further virtualised for multiple tenants in a customised manner for each network slice.

2.3.2 Used programmable platforms and APIs

To enable such flexible functional split, we exploit the FlexRAN+ platform provided in WP3, as an extension from the original FlexRAN platform [23] that only supports monolithic RAN deployment. In this sense, the extension considers also support for disaggregated RAN deployment. Moreover, such FlexRAN+ will support the changing of functional splits via using the out-band control scheme. Such FlexRAN+ control framework includes the FlexRAN+ controller (as the controller) as well as the RAN runtime (as the agent) on top of each RAN entity.

2.3.3 Function Design/Implementation/Evaluation

Based on the aforementioned FlexRAN+ platform, we show how it can compose the logical BS (IBS) from underlying RAN entities and then provide a customised form of the virtualised BS (vBS) in Figure 2-13. Toward this two-level abstraction scheme, the different functional splits can be applied by the FlexRAN+ without changing the composition of the IBS. For instance, in Figure 2-13, the physical deployments span different radio access technologies (Wi-Fi, 4G, 5G) and deployments (monolithic BS, two- and three-tier C-RAN entities) and they can be controlled and abstracted as a number of IBSs that comprise the necessary BS protocol and baseband processing. Each IBS will be further abstracted into vBSs for each network slice, e.g., 4/3/2 virtualised BSs are viewed by the slice 1/2/3, for the slice-specific control purposes. These vBSs can possess the slice-specific information, such as states, resources and processing capabilities.

First, the underlying heterogeneous physical RAN entities host a number of RAN PNFs/VNFs for CP/UP processing. These PNFs/VNFs can be chained by the split-aware RAN runtime to compose the RAN processing, e.g., the DU that comprises both MAC and RLC processing. Moreover, they can be either shared by several network slices or customised for slice-specific purposes, e.g., customised PDCP processing for low-latency purposes. The RAN runtime provides the underlying CP/UP processing information to the FlexRAN+ controller. However, some passive RAN entities do not possess the local RAN runtime due to their limited processing capabilities and they therefore rely on the in-band control through the remote RAN runtime on top of other entities. For instance, as can be seen in Figure 2-13, RUs and remote radio unit (RRU) rely on the in-band control through DU and baseband unit (BBU). To this end, the operating functionalities and the relation toward other RAN entities for these RUs/RRUs are maintained explicitly by the connecting DU/BBU.

Afterwards, each IBS can be formed by the FlexRAN+ controller via (1) chaining the necessary functions of a single BS across RAN entities, (2) abstracting the information such as the applied functional split and physical deployments, and (3) unifying the status and capability information to be further consumed by vBSs. This formation of an IBS is based on the required BS functionality among respective air interfaces and/or RATs. Moreover, the FlexRAN+ controller can apply the specific network operator control logic, e.g., the flexible change of the functional split between RAN entities, together with the underlying RAN runtime in an out-band control approach. Such out-band control approach relies on the control framework comprising FlexRAN+ controller and RAN runtime to maintain the service continuity.

To take one step further, the FlexRAN+ controller can virtualise each IBS into several vBSs based on the network slice description to reveal the information (e.g., capability, configuration and state) for each slice owner. This vBS is revealed in an as-a-resource manner to each network slice, e.g., virtualised/physical radio resources, shared/dedicated RAN processing, and slice-specific users' information and their associations, to apply the slice-specific control logic. Note that these multiple vBSs are isolated from each other. For instance, in Figure 2-13, the three vBSs (i.e., $vBS_{1,1}$, $vBS_{1,2}$ and $vBS_{1,3}$) over a single IBS (i.e., IBS_1) can apply the slice-specific radio resource management control logic toward the respective associated users without impacting other vBSs. Last but not least, a slice can retain its service requirements via controlling its vBSs and the associated users, e.g., slice 1 can balance the load among its 4 vBSs by handing over users to $vBS_{3,1}$ according to the vBSs' load information.

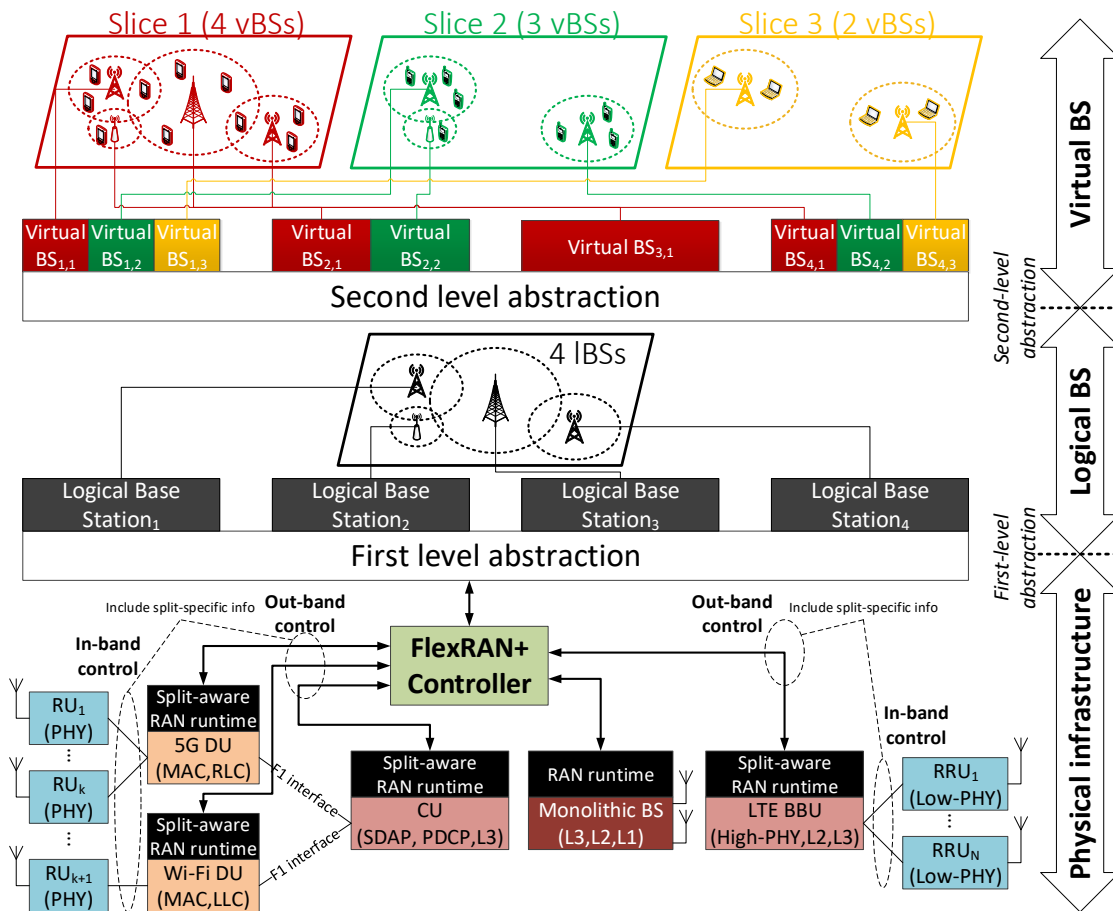


Figure 2-13: Two-level abstraction provided via FlexRAN+ architecture framework.

First of all, the resource manager is responsible to compose the IBS via interacting with the underlying RAN runtime instances through the south-bound interface. For instance, it can merge the underlying RAN entities' capabilities to form a logical 4G BS comprising the baseband and protocol processing ranging from physical layer, MAC, RLC, PDCP and radio resource control (RRC) for both monolithic and disaggregated deployments. Note that the controller will withhold this capability information once a IBS cannot be composed and wait for the corresponding RAN runtime to connect. The formed IBSs will then be stored in the LBS information base and can be updated on-the-fly according to the dynamics of physical infrastructures and their capabilities. Further, it can support the change of functional split of the underlying RAN entities without changing the formed IBSs.

Moreover, these IBSs can be treated as resources to be further abstracted as vBSs based on the needs of the slice owner. The second-level abstraction is performed by the virtualisation manager. For example, one slice may request RAT and deployment specific vBSs, and thus the particular IBSs with the corresponding deployment and RAT information are provided for such slice. In contrast, another slice may only needs vBSs of specific RAT and, therefore, the corresponding IBSs with the RAT specific information (e.g., LTE duplex mode and transmission mode) are provided. Further, another slice does not request technology specific vBS; therefore, the virtualisation manager will provide only performance indicators (e.g., average user-plane latency and throughput). These different levels of virtualisation are based on the slice data that contains the slice context, such as the slice service level agreement (SLA) and customised level.

2.3.4 Packaging for the 5G OS

This technical component aims to bring the vBSs as the customised resources for each network slice based on the formed IBS.

2.4 Technical Component 4: Disaggregated Heterogeneous Base Station functionality

2.4.1 Summary Description

Base station disaggregation will be a ground-breaking feature of the upcoming 5G networks, adding up to the network flexibility and manageability. To this aim, in the standardisation efforts of 5G-NR, the integration of heterogeneous technologies has been included by means of integrating heterogeneous DUs to a single CU [24]. The technical architecture of the standardised interface for the standardised functional split (3GPP option 2 split) is shown in Figure 2-14. This split is addressing the division of the base station into two elements, the CU and DU. A CU includes the processes of the PDCP layer and upwards, able to control multiple DUs incorporating the RLC layer and downwards.

3GPP standardised the interface for the communication between the CU and DUs through the introduction of the F1 interface and the F1AP protocol [20]. Different types of DUs shall be supported, including 5G-NR, legacy LTE and Wi-Fi. A single CU should be able to serve multiple DUs (1:n relationship), whereas each DU is served from a single CU (1:1 relationship). The data plane traffic (payload traffic forwarded to the network UEs) is transported over the F1-U interface, encapsulating the traffic with GPRS Tunneling Protocol (GTP) headers over UDP/IP, whereas the control plane (e.g., RRC signalling) is using the F1-C interface, running over SCTP/IP. Since this disaggregation of the base station functionality takes place at a higher layer, it allows lower layer splits to be incorporated, thus creating a multi-tier disaggregated architecture.

Based on this functionality, the UTH team has engaged in developing support for heterogeneous DUs in the OAI platform [25]. This means that for incorporating new DU functionality to the system (e.g., a Wi-Fi device), it should be augmented with the respective functions for handling the communication with the respective CU side, for both control and user plane as well as monitoring functions. As not all of the standards are yet finalised, regarding the F1 interface setup, we will adopt the following methodology in the developments of the VNFs for CU and DU operation:

- Initially we will provide an IP interface for the communication between the CU/DU interface, for both control plane and user plane traffic. This interface is hereby mentioned as F1 over IP (F1oIP).
- Later in the project, we will develop the 3GPP compatible interface.

The first solution relies on the creation of an IP interface for the logical connection between the CU and DUs, whereas the traffic can be transferred using any of the common transport protocols on top (UDP/TCP/SCTP). For successful communication of the components, we have defined a new protocol for addressing the CUs and DUs, and to piggyback information that is currently being used by the lower layers of the stack, residing at the DU, for controlling the scheduling of the transmissions in the RAN. As the implementation is based on the OAI platform, the processes for the CU (RRC, PDCP) and DU (RLC, MAC, PHY) operation already exist. We hereafter focus on the new elements of the network, being the F1oIP exchange protocol, and the Wi-Fi DUs. The detailed packet format that is employed to carry the data plane traffic between the CU and DUs is shown in Figure 2-15.

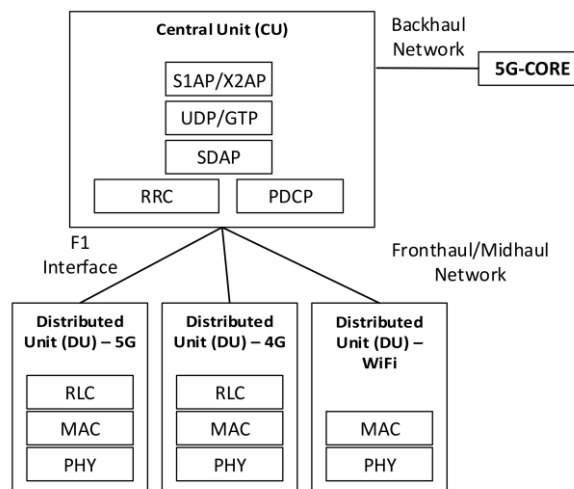


Figure 2-14: 5G RAN architecture for CU/DU operation.

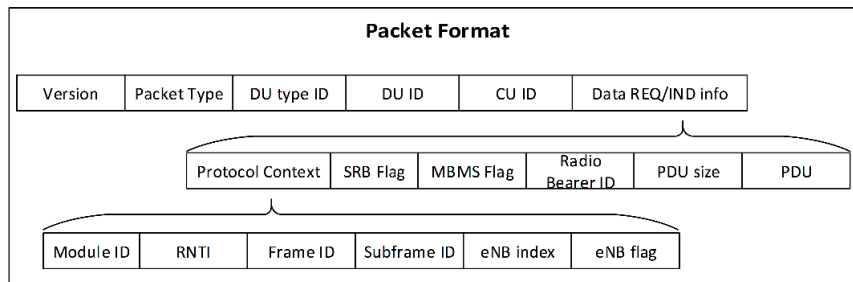


Figure 2-15: F1oIP packet format for sending data to the heterogeneous DUs.

2.4.2 Used programmable platforms and APIs

For the development of the aforementioned functionality, the OAI platform has been employed. OAI is able to run on any off-the-shelf GPP, with a compatible RF front-end. Although acceleration of the OAI features is currently supported in OAI through the Single Instruction, Multiple Data (SIMD) instruction set, no such requirements are needed to execute this technical component. Nevertheless, they can be used in order to yield performance benefits. Similarly, no formal requirements are needed for running the OAI platform over the accelerator solution that the UTH team implements in WP3 (porting OAI over the Zynq platform [4]). This porting will allow the OAI instance to reach new levels of channel bandwidth, and can be used complementary to the existing solution for providing the RAN as a VNF, to achieve better performance on the DU side.

2.4.3 Function Design/Implementation/Evaluation

In this section, we detail the implementation process and the different components that are developed in OAI and the respective DU software, and evaluate the solution in terms of performance. For the implementation and evaluation of the designed solution, we employ the NITOS testbed [26]. NITOS is used in the context of WP6 for the early validation of the project’s solutions.

2.4.3.1 Logical Connection between CU and DUs

The first solution relies on the creation of an IP-based interface for the logical connection between the CU and DUs. For the transferring of traffic between the different components, the common protocols are used (UDP/TCP/SCTP).

2.4.3.2 Communication Setup and Message Exchange

The CU and DU units are able to discover each other upon the system start, using predefined capabilities and a configuration file with the locations of the different modules. In the configuration file, information about the address and port of the DU side is provided. The communication is based on the client-server paradigm, with the DU running the server side and the CU the clients. Upon system startup, the PDCP layer spawns new thread processes running the clients that are associated with each DU. It is worth to mention that this capability allows us to further tailor the transport protocol based on how each DU better performs; for example, a single CU may maintain concurrent connections to an LTE and a WiFi DU, with the transport being carried over TCP for the former and over SCTP for the latter.

Upon this initial configuration phase, and through the exchange of “Hello” messages, capabilities exchange messages follow. Each DU send to the CU a message indicating the type of DU that is being handled, along with its current configuration. From this point, the exchange of the user-destined data taking place either on the DL or the UL channels, is being carried out through the F1oIP functions. Since both ends need to be informed of all the values needed for carrying out any computations at each receiving end (e.g., hash tables with the network users), extra fields shall be allocated at each packet piggy-backing all the needed information.

Each packet should include fields for packet type, DU type, and addresses each side through the DU ID and the CU ID. Different types may be supported for the same DU, as a single unit may incorporate functionality for both technologies, whereas the selection of the interface is made by the CU. This relies on a small module implemented on the OAI platform, which selects the DU that the traffic shall be forwarded to. Preliminary results show only three indicative policies for benchmarking purposes (DU aggregation, Round Robin, single technology) but can be further tailored by defining percentages between the different DUs, even during the base station execution through the utilisation of the FlexRAN implementation. The overall overhead posed by this header, along with the current status in the size of the respective variables that shall be used and exchanged for OAI is measured to be 80 bytes long.

2.4.3.3 Wi-Fi DUs

As the Wi-Fi stack significantly differs from the mobile networking stack in terms of the supported procedures, different processes need to take place upon the reception of the F1oIP packets for transmitting the payload to the network UE, or sending the data back to the CU. These processes include the reception of the F1oIP packets transmitted from the CU, unpacking and stripping off the respective headers, and subsequently delivering the payload to the wireless driver running on the DU device. For the UL data flow, payload traffic shall be encapsulated in the respective headers for the PDCP instance running on the CU. This includes dedicated processes for assigning new sequence numbers for the packets sent to the CU, as well as packet compression. Information that does not exist in the Wi-Fi operation (e.g., protocol context, data bearer ID) needs to be integrated on the F1oIP messages to allow the transparent handling of the packet reception at the CU side. Thus, a mapping between the Wi-Fi end-points with the related protocol context takes place in the DU. This process requires that the initial packet transmission (HELLO) happens from the CU to the DU, in order to keep this information, and updates are sent from the CU side in the case of a new UE. Through this process, all the heterogeneous DUs maintain a mapping between actual IP addresses used to reach each client of the network and the RNTIs that are used by the CU side to distinguish traffic between each different UE. In the case where the end-client side uses a similar joint PDCP procedure, such as a 5G-NR/LTE instance, this process can be omitted.

2.4.4 Evaluation Methodology and experiment setup

The described functionality has been executed over the NITOS testbed. NITOS is a heterogeneous testbed located in the premises of the University of Thessaly, in Greece. It offers a very rich experimentation environment with resources spanning from commercial LTE, to WiFi and Software Defined Radio (SDR) platforms that suits our experimentation needs.

For the development of the messaging exchange scheme, we employed Google's Protocol Buffers Library and the C language binding [27]. By formatting the message header through the *protobuf* library, the overall header size of our communication solution, along with the piggybacked information, is 80 bytes, which is exchanged between the CU and DUs or vice-versa whenever a packet is transmitted over the network.

The development of the CU/DU functionality has been written as a separate module inside the Layer 2 functionality of the OAI code. As the transport protocol between the CU and DUs, we use an asynchronous TCP or UDP interface. The current configuration of the CU enables the utilisation of different transport channels per each DU, thus allowing them to run with different settings (e.g. TCP for the LTE-DU and UDP for WiFi-DU).

The utilisation of the *protobuf* library provides the opportunity for applications of different languages to use the same message definitions. Therefore, for the development of the WiFi DU we used a Python based agent. This agent is capable of receiving the CU messages, retrieving the payload and injecting it to the WiFi device that is configured as an Access Point. The injection is being handled by the *Scapy* Python module [28], which provides bindings for creating packets and injecting them to a network interface. The topology used for our experimentation process is given in Figure 2-16. Since the current version of F1oIP is only overriding the data plane communication between the CU and the LTE DU, the production of two different binary files is not possible. However, we emulate this type of behaviour by injecting delay between the network interfaces that are used for this communication between the CU and DU, equal to 0,250 ms. The delay that we inject is performed using the *netem* application and is equal to the mean delay that we measure over the FH between the CU and the WiFi DU.

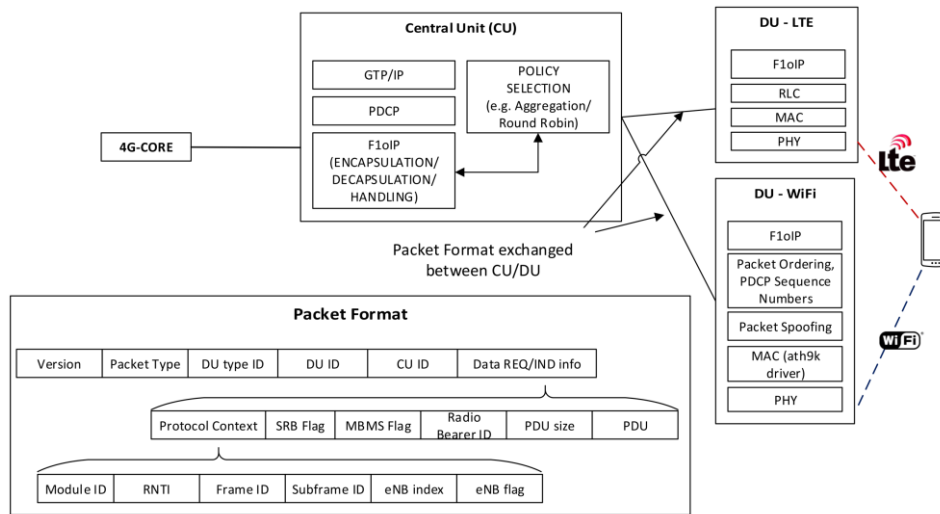


Figure 2-16: Experiment setup for heterogeneous DUs in NITOS.

Figure 2-16 provides our experimental setup for the NITOS case, considering a multi-homed UE with two wireless interfaces, a COTS LTE and a WiFi device. The experimental evaluation of our scheme is organised in two subsections: 1) Initial benchmarking of the platform for the different policies for network selection, and in terms of Cloud resource consumption as the number of DUs increases, and 2) evaluation based on the delay over the FH interface.

We measure the single-unit *vanilla* OAI eNB to achieve 34.4 Mb/s goodput for the DL channel for the under-test configuration. The measurements are carried out at the UE, by injecting traffic from the Core Network. Subsequently we measure the performance of OAI including our additions, for either UDP or TCP based FH (see Figure 2-17). We evaluate the process for three different policies of forwarding traffic from the CU side to the DUs, as follows:

- Aggregation Mode, where the traffic is sent to all the available DUs.
- Round Robin, where each DU is selected with a Round Robin manner. For the cases of two DUs, the total injected traffic is split to 50% per each DU.
- Single DU, where only one of the DUs is selected for transmitting the data traffic.

We see that for the Aggregation mode, in which the CU is forwarding traffic to all available DUs, the achieved performance for the LTE network is close to the *vanilla* setup. Likewise, the single network selection policy produces similar results. This is due to the configuration of F1oIP that exchanges signalling messages between the CU and the DU only during the initial setup phase. For the Round-Robin configuration we observed slightly lower performance for both DUs, caused by the extra delay induced in the system by the respective processes that determine the DU selection.

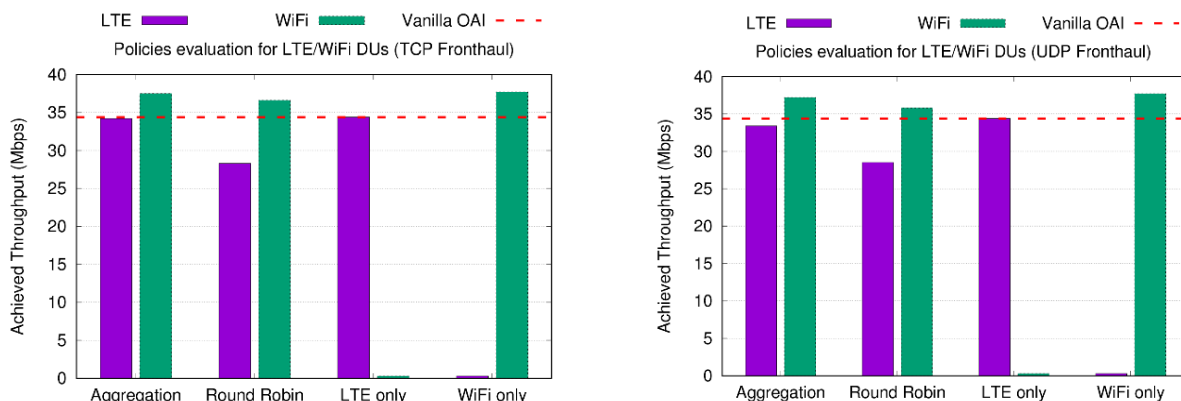


Figure 2-17: Policies Evaluation for different FH transport.

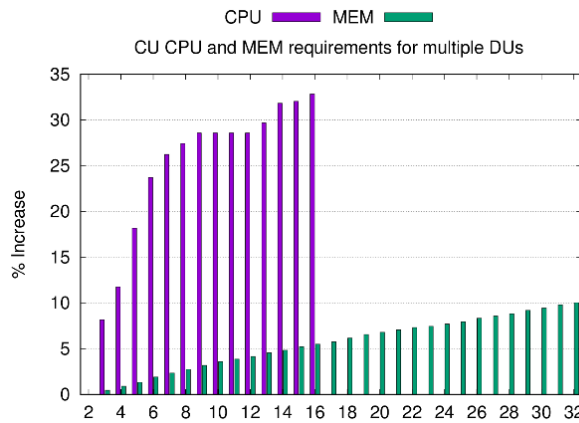


Figure 2-18: CPU and MEM utilisation on the CU side for varying number of managed DUs.

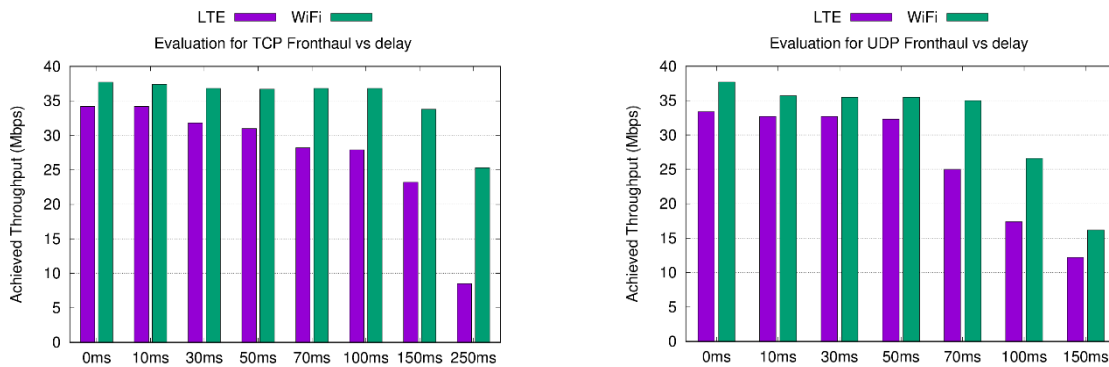


Figure 2-19: Policies Evaluation for varying delay on the FH.

As a second benchmarking evaluation we measure the requirements for the CU in processing power and memory, when varying in the number of DUs deployed. Figure 2-18 depicts our experimental findings in terms of measured overhead for each new DU introduced to the system, compared to an initial setup with 2 DUs. We use only WiFi DUs for this type of experiment. We measure the resource requirements for up to 16 DUs in the system, as at that point we determine that the CPU of the machine running the CU software is exhausted. As illustrated, the processing resources needed to run the CU for up to 8 DUs requires approximately 25% more processing power compared to the 2 DUs scenario. For supporting the remaining set of the DUs (up to 16 DUs) we require about 34% more processing power. For the memory usage we observe a near linear increase as new DUs are added to the system. Approximately, from the CU side, each new DU consumes additionally about 30MB of memory for its efficient operation. This performance metric needs to be taken into consideration when selecting to deploy the function as a VNF, as the respective processing and memory resources shall be allocated to the host machine.

As a second set of experiments, we measure the delivered goodput and Round Trip Time (RTT) for varying delay on the FH link. The latency on both the LTE and WiFi links is measured to be the same. We use the netem application to set delay on the FH link. We use the aggregation policy for these experiments, as this is the policy that produced higher results in the initial benchmarking experiments in the previous section. Figure 2-19 illustrates the results for either UDP or TCP F1oIP protocol configuration.

For both cases, we see that the performance starts to drop at around 70 ms of MH latency. Nevertheless, the respective RTT (see Figure 2-20) for the same interfaces seems to be growing by the double delay and a fixed amount added by the wireless access. Based on our results, we can incur that if the FH interface is realised over a fibre-based Ethernet link, the CU will be able to serve distributed DUs located at 200 km away without any decrease in the provisioned service at the end-client in terms of throughput performance. Of course, in such environments, we need to investigate further on how to differentiate the paths that low latency applications take in order to minimise the impact on the user's QoE.

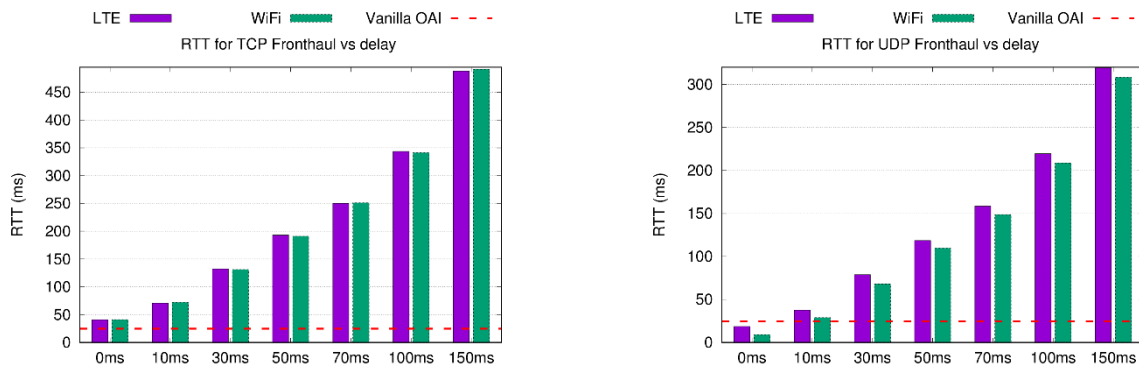


Figure 2-20: UE RTT times for different technologies for varying delay on the FH.

2.4.5 Packaging for the 5G-OS

The provided functionality is currently being wrapped to be provided as a set of VNFs that will be deployed through the 5G OS being developed in WP5. Four different VNFs will be developed for deploying the solution:

- A cellular DU (LTE or 5G-NR) VNF.
- A WiFi DU VNF.
- A CU VNF.
- Core Network VNF(s).

As the core network may be broken up to three different VNFs, further disaggregation of the core network VNF to three more is possible. For the initial bootstrap of the VNFs, the APIs developed in WP3 [4] will be used in order to allow the transparent configuration of the VNFs. The APIs setup a REST-based agent software, provisioned through Cloud-Init [29] along with the actual VNF, in charge of discovering the interconnected VNFs (e.g. DU with CU) and setting up the initial configuration files.

As the environment for deploying and instantiating the VNFs, we are using the OpenSourceMANO (OSM) orchestrator, which is compliant with the NFV-MANO architecture, managing the NITOS testbed as the NFVI. In order to accommodate functionality for deploying VNFs over heterogeneous networks, the VIMs is a slightly altered version of OpenVIM and OpenStack. Through these developments, we can stitch the VNFs to wireless interfaces instead of the traditional Ethernet interfaces that these VIMs support.

Below is a Network Service Description (NSD) of instantiating the Core Network, CU, LTE and WiFi DUs over the NITOS testbed. The NSD will be made available to the 5G OS repository of services.


```

nsd:nsd-catalog:
  nsd:
  - id: disaggregated-het-dus-ns
    name: disaggregated-het-dus-ns
    short-name: oai-ns
    description: NSD to create a software based disaggregated base station system (RAN + EPC) based on OpenAirInterface. Consists of 3 VNFs for the RAN and the EPC
    version: '1.0'
    version: '1.0'
    logo: osm.png
    constituent-vnfd:
    - vnfd-id-ref: oai_cu-vnf
      member-vnf-index: '1'
    - vnfd-id-ref: oai_du-vnf
      member-vnf-index: '2'
    - vnfd-id-ref: wifi_du-vnf
      member-vnf-index: '3'
    - vnfd-id-ref: oai_cn-vnf
      member-vnf-index: '4'
  vid:
  - id: mgmtnet
    name: mgmtnet
    short-name: mgmtnet
    type: ELAN
    vim-network-name: provider
    vnfd-connection-point-ref:
    - vnfd-id-ref: oai_cu-vnf
      member-vnf-index-ref: '1'
      vnfd-connection-point-ref: vnf-mgmt
    - vnfd-id-ref: oai_du-vnf
      member-vnf-index-ref: '2'
      vnfd-connection-point-ref: vnf-mgmt
    - vnfd-id-ref: wifi_du-vnf
      member-vnf-index-ref: '3'
      vnfd-connection-point-ref: vnf-mgmt
    - vnfd-id-ref: oai_cn-vnf
      member-vnf-index-ref: '4'
      vnfd-connection-point-ref: vnf-mgmt
  - id: datanet
    name: datanet
    short-name: datanet
    type: ELAN
    vim-network-name: provider2
    mgmt-network: false
    vnfd-connection-point-ref:
    - vnfd-id-ref: oai_cu-vnf
      member-vnf-index-ref: '1'
      vnfd-connection-point-ref: vnf-data
    - vnfd-id-ref: oai_du-vnf
      member-vnf-index-ref: '2'
      vnfd-connection-point-ref: vnf-data
    - vnfd-id-ref: wifi_du-vnf
      member-vnf-index-ref: '3'
      vnfd-connection-point-ref: vnf-data
    - vnfd-id-ref: oai_cn-vnf
      member-vnf-index-ref: '4'
      vnfd-connection-point-ref: vnf-data
  
```

Figure 2-21: NSD description for supporting disaggregated heterogeneous RAN VNFs in OSM.

2.5 Technical Component 5: Wireless Transport Technologies with Functional Split Support

2.5.1 Summary Description

The goal of this technical component is to study under what conditions Sub-6 wireless backhaul technologies could be used to transport some of the functional splits defined by the eCPRI standard. Figure 6 included in deliverable D4.1 [3] depicts the functional splits defined by 3GPP and eCPRI, where the lower the functional split (i.e. closer to the RF front-end), the higher the potential centralisation gains.

For our study, we discard functional splits that require the transport of digitised radio samples, and instead focus our attention to eCPRI split [9] that is based on the transport of frequency domain samples between the Distributed Unit (CU) and the Remote Unit (RU).

The eCPRI functional split is supported by the OAI format, where is referred to as the IF4p5 split [30], with an architecture illustrated in Figure 2-22.

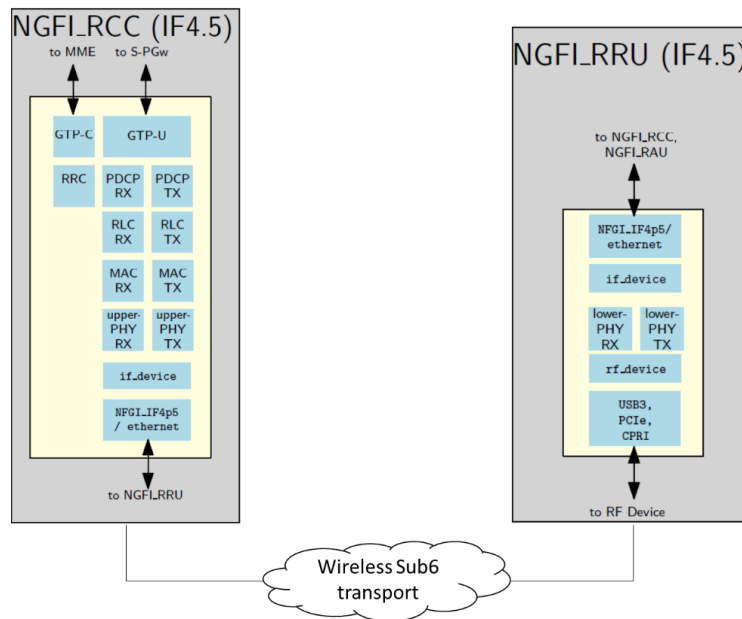


Figure 2-22: OAI IF4p5 functional split over Sub-6 wireless transport.

Table 2-4: OAI IF4p5 required transport capacity.

Carrier bandwidth	OAI IF4p5 capacity
5 MHz	138 Mb/s
10 MHz	276 Mb/s
20 MHz	553 Mb/s

The bandwidth required to transport the OAI IF4p5 interface depends on the number of antennas, and the bandwidth of the LTE wireless carrier. Our interest in this study is on low data rate Radio Access Networks (RANs), hence we focus on SISO systems with 5, 10 and 20 MHz of wireless carrier bandwidth. In this settings we experimentally measure the following required capacities in the OAI IF4p5 interface:

As a reference Sub-6 wireless transport we use the wireless BH technology being developed by I2CAT in WP3. These BH nodes are based on a Single Board Computer (SBC) manufactured by Gateworks running Ubuntu Linux as OS (kernel version 4.9.65), which may connect multiple wireless modems through a mini-PCIe interface. The selected board features QCA9888 modems from Qualcomm Atheros implementing the wave 1 of the IEEE 802.11ac standard. This radio technology is capable of operating with an 80 MHz carrier bandwidth, and delivers application layer data rates between 200 Mb/s and 300 Mb/s in practical deployments [31].

2.5.2 Used programmable platforms and APIs

This technical component is based on the joint access-backhaul SBC developed as part of WP3, and reported in deliverable D3.2 section 3.2 [5].

2.5.3 Function Design/Implementation/Evaluation

Looking at the OAI IF4p5 data rates included in Table 2-4, we can see that: i) a 5 MHz carrier should be readily supported by the considered Sub-6 wireless transport, ii) a 10 MHz carrier bandwidth is pushing the current limits of a single wave 1 IEEE 802.11ac modem, and iii) a 20 MHz is not supported by one modem.

With this a-priori knowledge we as ourselves:

- Q1. What is the maximum number of hops that we can use to BH a 5 MHz carrier using a IF4p5 interface?
- Q2. Can we bond, using SW, multiple wave 1 IEEE 802.11ac modems in order to aggregate capacities and serve the 10 MHz and 20 MHz carriers?

In order to bond multiple IEEE 802.11ac wireless modems using software we propose the architecture depicted in Figure 2-23.

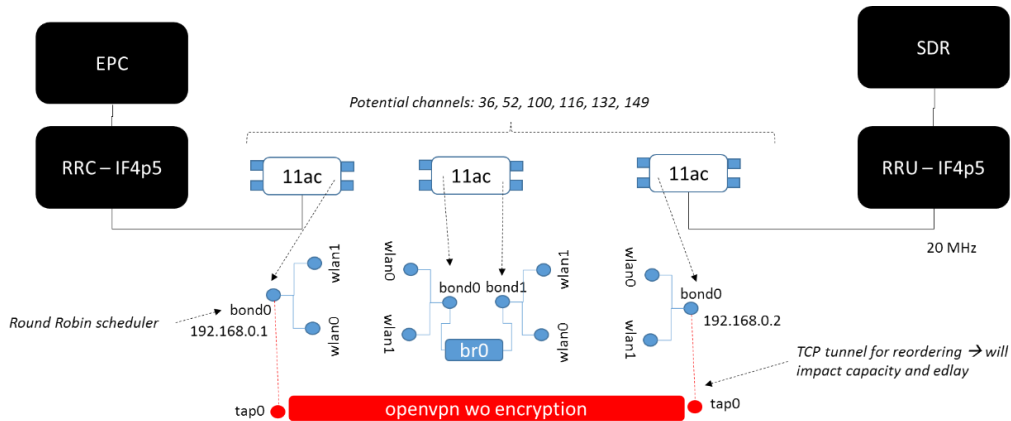


Figure 2-23: Software based NIC bonding architecture.

The challenge in SW-based bonding is to deliver the packets in order at the other side of the bond. Hence, we adopt the following design:

1. Individual mac80211 based wireless NICs are pairwise connected between a pair of nodes, forming separate wireless networks operating at different channels to minimize interference. See in the figure the potential IEEE 802.11 channels that could be considered for operation at 80 MHz mode.
2. At each node the various wireless interfaces (*wlanX*) facing a given neighbor, are bonded together using an existing Linux tool known as a bond interface (e.g. *bond0*)¹. A bond interface aggregates multiple network interfaces into a single abstracted software interface presented to the OS, which supports various schedulers to distribute incoming traffic between the various wireless interfaces. In our case, since we want to increase capacity, we use a Round Robin (RR) scheduler.
3. Finally, noticing that the RR scheduler at the bonding interfaces could introduce packet reordering, we need to introduce buffers that allow the reordering of packets if they are delivered out of order. For this purpose we need a tunneling technology that offers a layer two abstraction to the OS, while using TCP as underlying technology. We use *openvpn*² disabling encryption to avoid increased latencies.

2.5.4 Evaluation Methodology and experiment setup

To evaluate the throughput achievable by aggregating multiple wireless links, we set-up a 1-hop double link connection between 2 SBCs as shown in Figure 2-24.

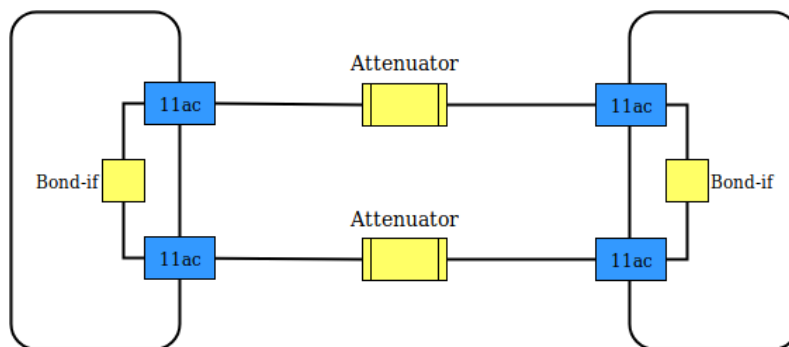


Figure 2-24: Single-hop set-up used to perform the bonding tests. Each IEEE 802.11ac WiFi card is MIMO 2x2.

¹ <https://wiki.linuxfoundation.org/networking/bonding>

² <https://openvpn.net/>

The premises where the tests took place are overfull of active WiFi networks sharing the same spectrum that can considerably compromise the accuracy of the results. For such a reason we decided to connect the WiFi NICs forming a single wireless network via a cable. To simulate the channel attenuation, RF attenuators have been used. Even under this condition we measured a certain degree of interference from the active WiFi networks whose RSSI has been measured between -70 and -85 dBm. Since the channel attenuation has been calculated to provide an average RSSI at the receiving interface of -45 dBm, the interference from external WiFi sources can be considered not relevant enough to cause an appreciable performance degradation.

As showed in Figure 2-24, each SBC is equipped with 2 IEEE 802.11ac WiFi card each of them 2x2 MIMO capable. The SBC is ARMv7 based running a custom Linux Kernel version 4.9.65. The tests have been performed using the iperf and iperf³ tools to evaluate the end-to-end throughput using UDP and TCP as transport protocol.

The bonding functionality has been added to the Linux kernel compiling the *bonding.ko* module. Other than the bonding module, it was required to add the teaming module necessary to combine the interface together and the specific *team_mode* module to be used. For these experiments the *team_mode_roundrobin.ko* has been compiled. The round-robin mode transmits packets in a sequential order from the first available slave interface (the WiFi interface) through the last. If two interfaces are slaves in the bond and two packets arrive to the bond interface, the first packet will be transmitted on the first slave and the second one will be transmitted on the second slave. A third packet will be sent on the first slave and so on.

Parameters of the bonding configuration have been adjusted in the attempt of maximising the throughput, however evaluations and tests are still in progress, and additional progress will be reported as part of deliverable D4.3.

We measured end-to-end throughput in two different configurations: i) sending one packet per WiFi slave interface, and ii) sending 32 packets per WiFi slave interface. The choice of how many packets to send over a WiFi interface affects (at least in theory) the aggregation and the system queueing process, hence the overall performance. The results in Figure 2-25 depict that the WiFi interfaces transmitting alone (not bonded together) are able to provide a UDP throughput around 270 Mb/s in average and a TCP throughput around 360 Mb/s in average.

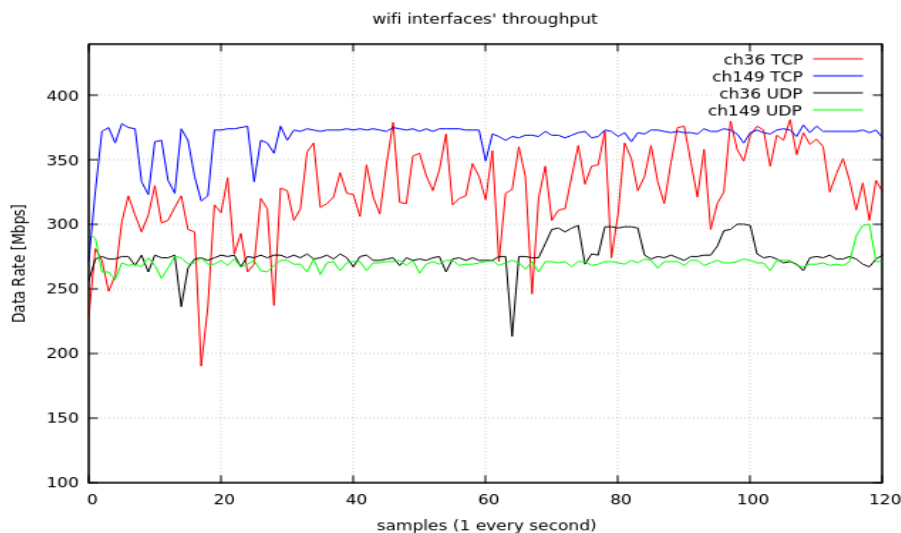


Figure 2-25: Throughput of each WiFi interface transmitting alone.

Bonding 2 WiFi interfaces together performing a simple round-robin scheduling between them brought to an achieved throughput that has been measured around 470 Mb/s in average for UDP traffic, but of only 250 Mb/s in average for TCP, as depicted in Figure 2-26 .

³ <https://iperf.fr/>

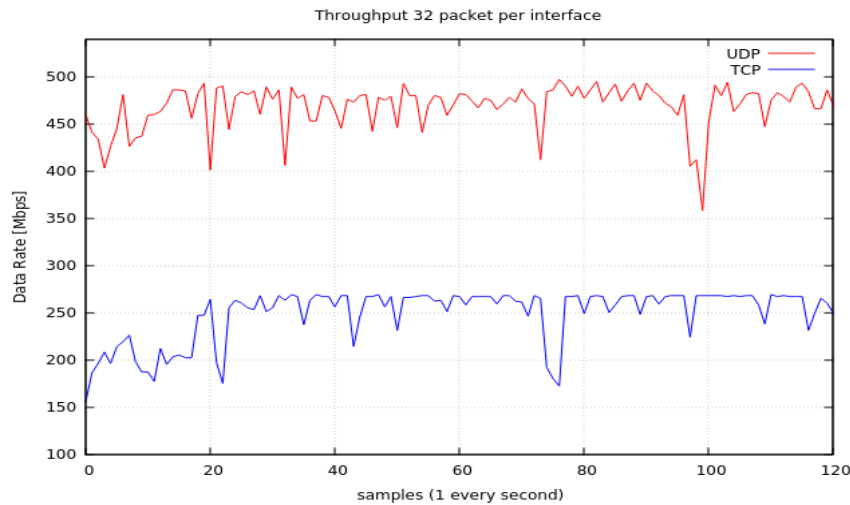


Figure 2-26: UDP and TCP throughput achieved by bonding two interface together.

While a performance close to 500 Mb/s is achieved when bonding two wireless interfaces together, and transmitting UDP traffic, surprisingly the bonding performance decreases in the case of TCP. The reason why a TCP performance is severely affected is due to an interaction between the round-robin mechanism, the AMPDU aggregation used by the IEEE 802.11ac NICs, and the decrease in congestion window triggered by TCP when receiving three consecutive duplicate ACKs. To understand this effect, consider the following situation:

1. TCP packets are delivered to the two wireless NICs in a round robin fashion, hence the wlan0 interface receives packets #1, #3, #5, ..., and the wlan1 interface receives packets #2, #4, #6, ...
2. wlan0 is the first interface to transmit over the air and when doing so creates an AMPDU aggregate with packets #1, #3, #5, ...
3. Immediately upon receiving these packets the TCP sink generates three ACKs all requesting the packet with sequence number #2, which is sitting in the queues of wlan1 interface in the transmitter.
4. Upon receiving the three duplicate ACKs the TCP source decreases the congestion window, hence the throughput is impacted.

This is a problem in our attempt to transmit a FH interface over the wireless link, because the FH interface cannot tolerate reordered packets. Hence additional research is required, for which we consider the following potential paths that will be studied in deliverable D4.3:

- Introducing an additional virtual interface between the bonded interface and the TCP streams that reorders incoming packets. Such interface could be developed using Click-router modules.
- Using MP-TCP as a mechanism to aggregate capacity at the higher layer. The advantage of MP-TCP is that a separate congestion is kept for each path, and hence uneven path capacities could be efficiently aggregated.

In addition, we observed a CPU performance issue while running UDP throughput tests in the proposed architecture. The high queueing activity at the bonding interface receiving end continuously generates software interrupts overloading the CPU capacity. Such issue brought to instability of the receiving system as observed during an intensive test reported in Figure 2-27.



Figure 2-27: UDP throughput degradation due to high queuing interrupt activity that overload the CPU capacity.

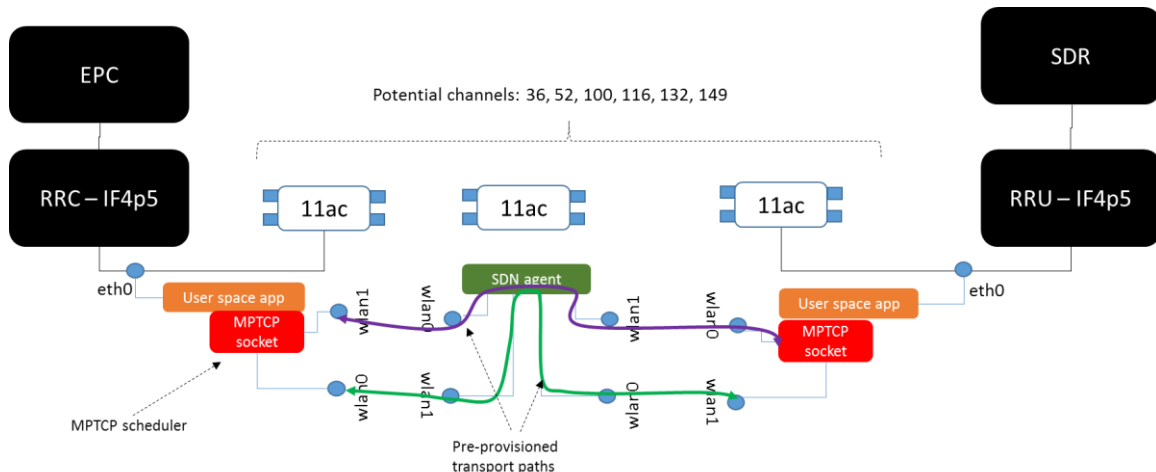


Figure 2-28: Bonding architecture based on MP-TCP.

Based on the problems we discovered in our original architecture based on the kernel bonding module, we turn our attention to an alternative architecture based on Multipath TCP (MP-TCP), which is depicted in Figure 2-28.

MP-TCP is a transport protocol that replaces TCP and allows the aggregation multiple IP interfaces, representing different paths within a single transport session. MP-TCP automatically balances the data to be transmitted among the available paths (IP addresses), according to its built-in scheduling and congestion control mechanisms. Thus, we propose an architecture where the edge transport devices encapsulate incoming packets into an MP-TCP session that can aggregate capacity among transport paths (c.f. purple and green paths in Figure 2-28). In our architecture the end-to-end transport paths across the wireless transport should have been pre-provisioned before the actual traffic flow begins. The advantage of our architecture is that the transport paths can be dynamically updated in a manner that is transparent to MP-TCP.

Taking into account these considerations, we tested the Linux Kernel implementation of Multipath TCP (MP-TCP)⁴. MP-TCP needs a modified Linux Kernel to be implemented. For this reason, a custom kernel version 4.14.79+ has been compiled. The MP-TCP implementation in Linux is modular and is able to incorporate

⁴ <https://multipath-tcp.org/>

different congestion control mechanisms. For our evaluation we study the following congestion control modules:

- Linked Increase Algorithm (LIA) [32].
- Opportunistic Linked Increase Algorithm (OLIA) [32].
- The Balanced Linked Adaptation Congestion Control Algorithm (BALIA) [33].

The results obtained are shown in Figure 2-29. Under the same conditions described in the beginning of section 2.5.4, MP-TCP proved to be able to aggregate the throughput of the two IEEE 802.11ac interfaces providing a stable TCP throughput around 500 Mb/s in average. The choice of the TCP congestion control algorithm does not make a substantial difference as depicted in the lower part of Figure 2-29, as long as it is one of the recommended algorithms. However, we measured a performance decrease using Cubic (the default Linux Congestion Control algorithm). A slight throughput improvement, with respect to the provided results, has been measured classifying transport traffic as AC_VO in IEEE 802.11ac, which uses the smallest contention window.

From the CPU usage point of view, MP-TCP is less demanding than bonding. The CPU usage never overcame 40% overall, and the queueing interrupt process contribute for at most 24% of the total CPU capacity.

After this evaluation we have not been able to reach the 553 Mb/s required to transport OAI IF4p5 traffic for a 20 MHz cell, but we can easily FH 5 MHz and 10 MHz cells. However, the performance delivered by MP-TCP is very close (approximately 500 Mb/s) to the level required to FH 20 MHz cells. Hence, we believe that with additional research on FH compression and the use of one 160 MHz channel in one of the IEEE 802.11ac NICs the 20 MHz target is on sight.

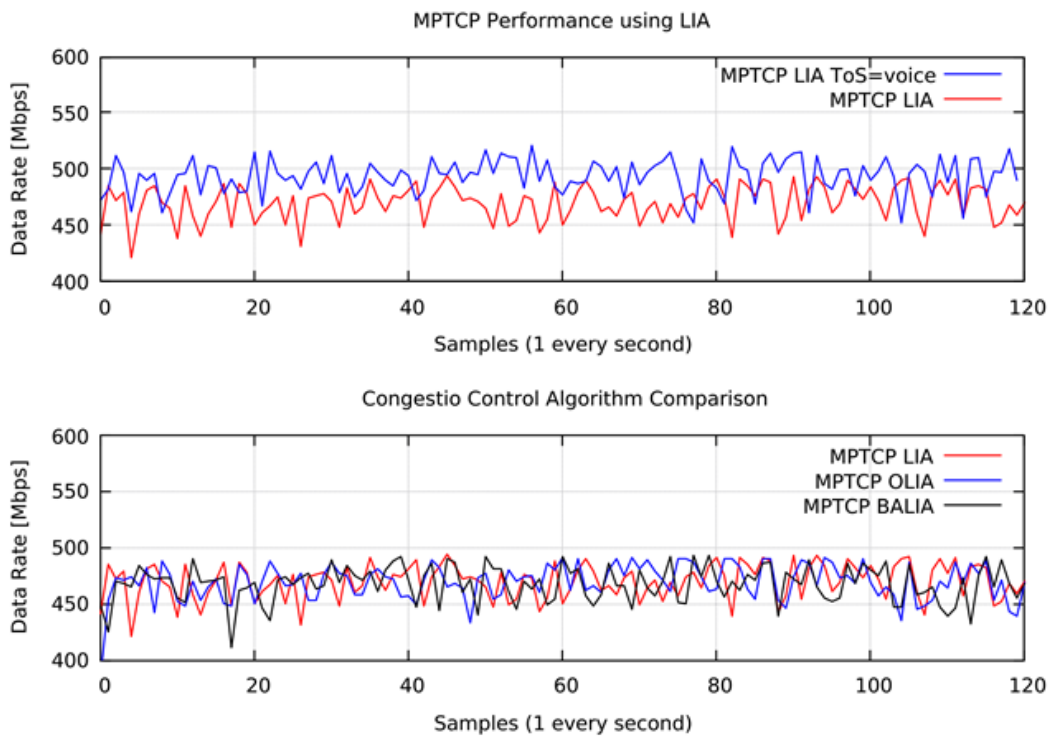


Figure 2-29: MPTCP throughput measured using different congestion control algorithms and ToS.

2.5.5 Packaging for the 5G OS

Does not apply.

3 Transport Slicing for converged wired-wireless FH/BH networks and integration with 5G-PICTURE orchestrator

3.1 Introduction

Under the umbrella of transport networking, BH and FH networks offer the underpinning for connectivity services for the case of disaggregated RAN with multiple functional splits. In order to couple with the strain raised and satisfy the strict requirements of 5G systems, significant changes have been undergoing in transport networking. In this section we analyse the technical components for the following three categories of transport networks: optical, Ethernet/IP and wireless networks. As described also in deliverable D4.1 [3] the reason for selecting this classification is that unlike layer 2 and layer 3 networks, optical network resources and transport format are different due to their analogue nature. Similarly, when using wireless technologies these are subject to issues like interference or channel variations that we do not meet in optical networks.

3.2 Technical Component 1: Time Shared Optical Network (TSON)

3.2.1 Summary Description

The TSON network brings flexibility and scalability in the optical transport networks. The TSON node using flexible TDM, allows allocation of time slots. It also permits to configure the size of a time slot. The TSON edge node is able to classify the ingress traffic according to the VLAN tag where each network slice is identified by a VLAN ID.

To benefit the advantages of TSON, an SDN controller is used to configure the TSON nodes. In this section, we focus on the description of the SDN controller used to configure the TSON nodes in order to provide end-to-end connectivity through the TSON optical transport network.

The TSON SDN controller is based on the open source OpenDayLight (ODL) platform. It is composed essentially of three important modules which are:

- Topology Manager.
- Open Source Mano (OSM) Northbound module.
- Extended OpenFlow modules.

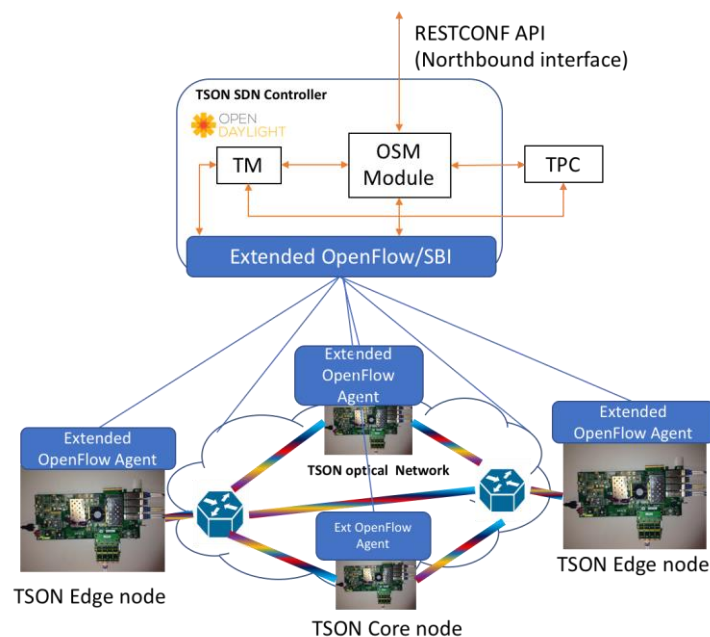


Figure 3-1. TSON ODL SDN controller.

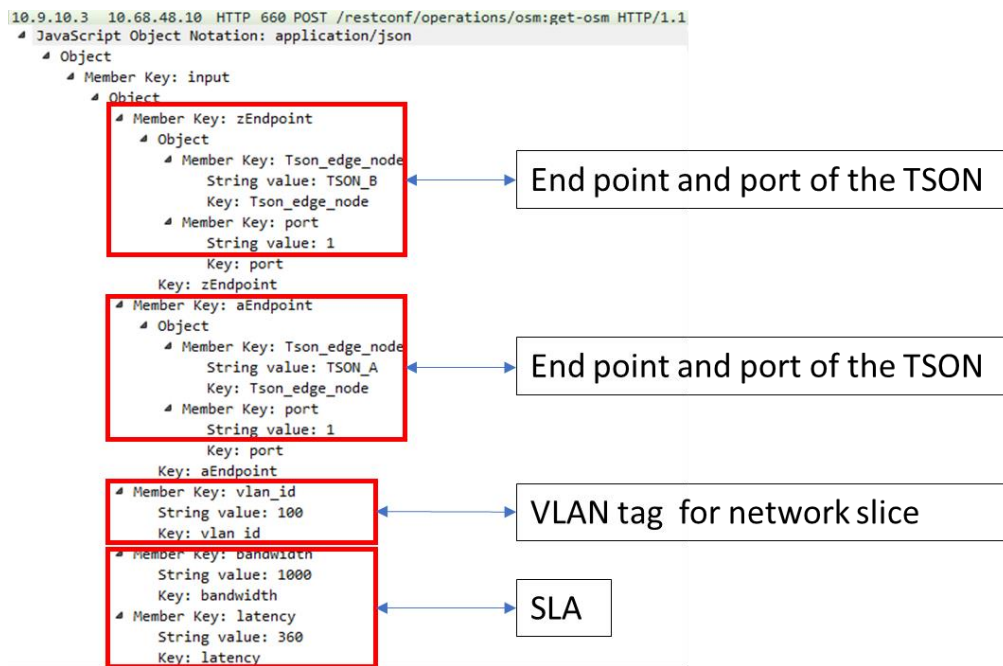


Figure 3-2 Restconf request getting by the OSM module.

The Topology Manager (TM) is the module which exposes an abstraction of the TSON network topology. It provides the inputs to the other modules. For example, it provides the information about the different nodes to the OSM module. The OSM northbound module permits to get the request from the northbound interface and initiate the process to configure the TSON node according to the received request.

The OSM module RESTCONF API is able to receive the request to establish the end-to-end connectivity and expose the TSON edge node topology to the northbound. Figure 3-2 shows the json frame representing the end-to-end request through the TSON network received by the OSM module. This request defines the end points and the port toward the client entity. It also defines the VLAN tag to separate the network slices and the required SLA of the transmission.

The extended OpenFlow is designed and implemented within ODL to configure the TSON optical network. The extended OpenFlow modules are OpenFlowJava module and OpenFlowPlugin. They handle the OpenFlow messages to configure the TSON nodes and the optical elements (optical cross connection switch, Wavelength Selective Switch). The current extension permits to set the number of the time slices in the TSON frame. These modules are going to be extended to allow setting up the size of the frame, the size of the time slice and the size of the overhead. Also, since the OpenFlow protocol has relatively poor flexibility in terms of extension and implementation in ODL, studies are going on to evaluate the use of the NETCONF protocol to configure the TSON nodes.

In addition, an external module to ODL, written in python, is developed to compute the path in the TSON network. This module is called TSON Path Computation (TPC). The calculation of the path is based on bandwidth and latency.

When the OSM module receives the request to establish the end-to-end connectivity through the TSON network. The request contains the TSON endpoints, the ingress port, the bandwidth, the latency and the VLAN tag which permits to identify the network slice. The request is forwarded to the TPC. The TPC based on the acquired abstraction of the topology and empirical model, determines the path through the TSON network by using shortest path algorithm. The TPC module contains a list where the status of the TSON nodes and the different elements in the TSON network such as the optical cross connection switch are stored. The result of the calculation path is sent to the OSM module, containing the different nodes and the ports involved in the transmission, the bandwidth and latency. The OSM module, according to the received result, uses the extended OpenFlowJava and plugin APIs to configure the appropriate nodes.

3.2.2 Used programmable platforms and APIs

Section 2 of deliverable D3.1 [4] describes the different programmable platform designed and developed to transport and handle the heterogeneous traffic. The first used platform implements the TSON developed program in the Xilinx VC709 platform. This FPGA can be implemented as a standalone or in the test server machine to run as a TSON node [4]. Figure 3-4 shows the standalone TSON node. The current implementation provides 2 inputs as clients and 2 outputs towards the TSON network providing the TSON frames. It also contains the Look Up Table (LUT) port supported by the FMC card plugged in the Xilinx VC709, to allow reprogramming the parameters of the TSON node.

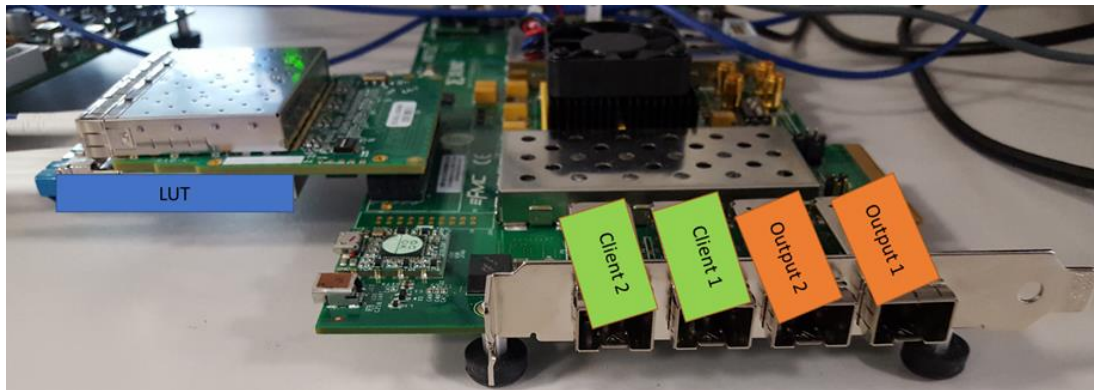


Figure 3-3 TSON node implemented in Xilinx VC709 FPGA platform.

Front view of the server

Side view of the server

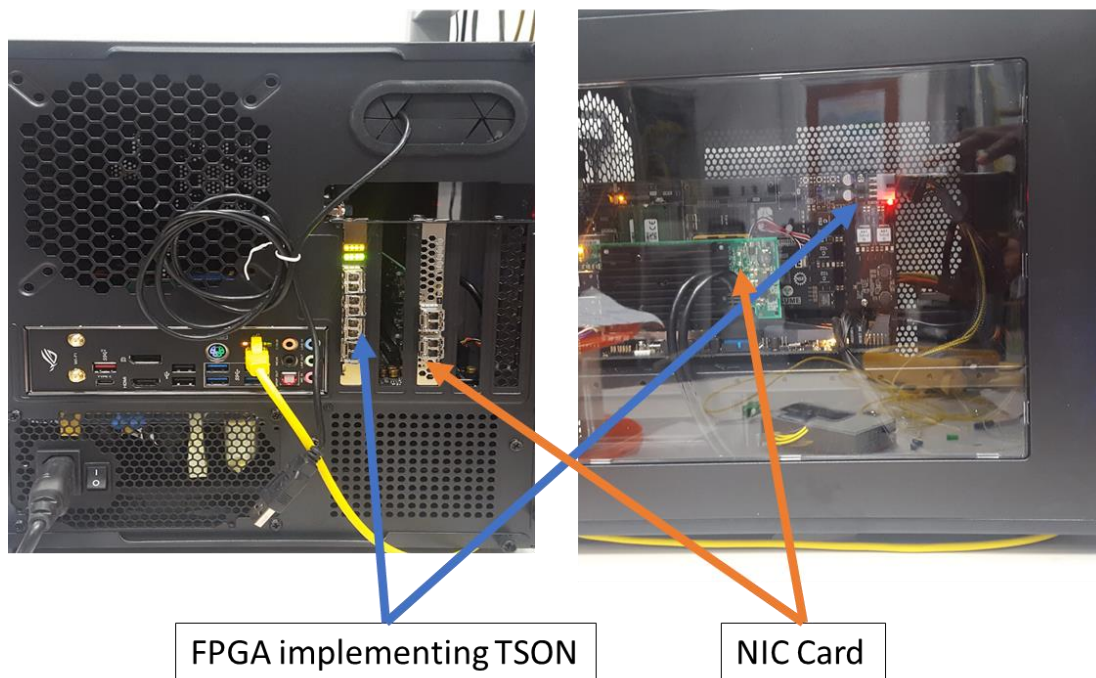


Figure 3-4 TSON implementation in FPGA embedded in the server.

The current implementation is able to classify 2 VLANs tags. Moreover, it is possible to extend the number of VLAN tags up to 4 by using the other ports of the FMC card. An OpenFlow SDN agent is designed and developed to allow the FPGA to receive the instruction from the SDN controller. This OpenFlow SDN agent is implemented in the server and communicates with the FPGA using the LUT port.

Figure 3-5 illustrates the first results regarding the implementation of the TSON node in the test server machine. The NIC Card is used to communicate with the SDN controller. The developed OpenFlow API agent

is implemented in the server containing the FPGA. The LUT configuration of the Implemented TSON is performed via the PCIe interface instead of to use SFP LUT port in the standalone implementation case.

3.2.2.1 OpenFlow protocol Agent

The programmable platform within TSON FPGA is described in deliverable D3.1. This platform contains the API called *SDN agent*, which is the interface between TSON node platform and the SDN controller. The API contains the extended library of OpenFlow to handle the configuration of TSON node. It contains 2 interfaces, one of them is connected to the SDN controller using a TCP/IP connection and the other one is connected to the TSON node via the PCIe or the raw socket to communicate via the SFP port.

Figure 3-6 shows the supported action messages by the SDN agent. These action messages are extended to support TSON parameters. This API receives the request from the SDN controller to configure the TSON node. It gets the parameters from the OpenFlow messages in order to build the Ethernet Frame to set the Look Up Table (LUT) of the FPGA to set up the TSON node. The SDN controller is able to get the status of the current configuration of the TSON node.

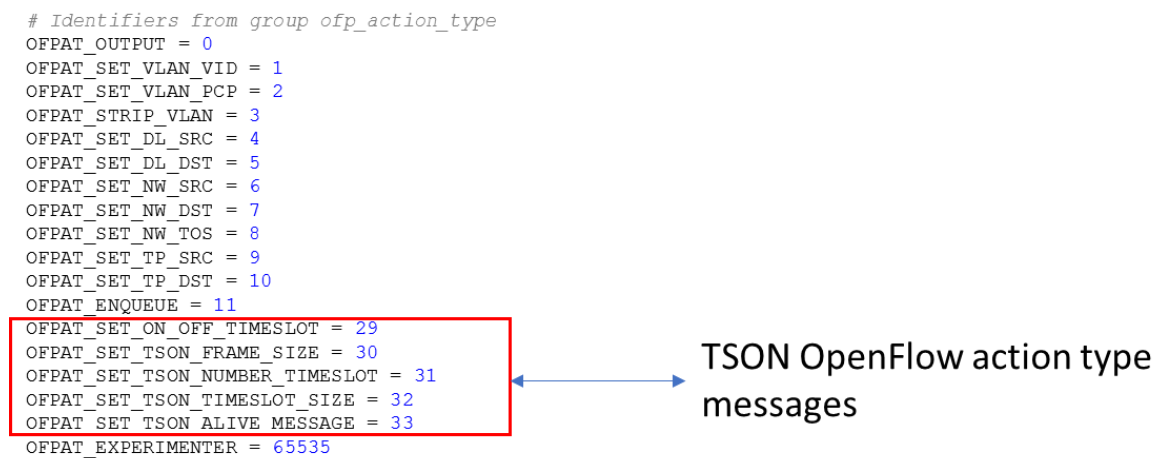


Figure 3-5 Extended OpenFlow message for TSON.

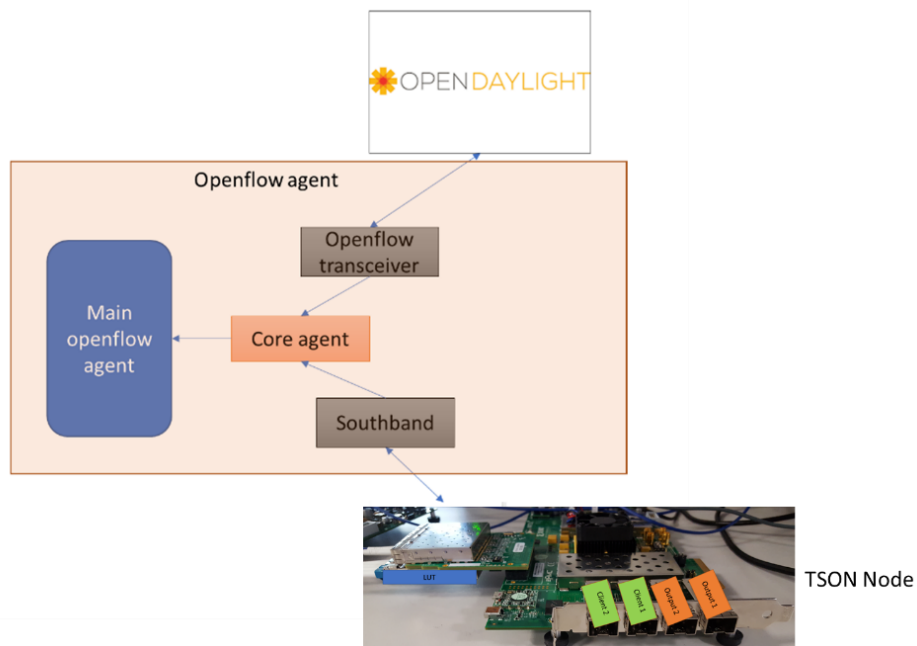


Figure 3-6: General SDN agent design to control a standalone TSON implementation.

The SDN agent is divided into 2 main parts. The entry point is the main agent, which calls the second important component: the core agent (Figure 3-7). The core agent runs two threads:

- OpenFlow Transceiver.
- Southband Thread.

The *OpenFlow Transceiver* permits to the agent to communicate with the ODL controller. It is composed of two methods within *OfTransceiver Class*. It creates a TCP/IP socket with the IP address and the port of the controller. In addition, it selects the right version of the OpenFlow requested by the user.

The Southbound communication is ensured by two classes, which are the *NetworkDevice* and the *NetworkDeviceTransceiver*. The *NetworkDeviceTransceiver* class is the most important class to communicate with the FPGA. It contains the method to initiate the connection between the python *openflow* agent and the FPGA and the methods to build the read the Ethernet LUT frames to configure the TSON parameters. It contains the methods required to send the built Ethernet LUT Frame to configure the FPGA, and the methods that read the messages from the TSON Node.

In the case of the implementation in the test server machine the southbound is designed to communicate with the FPGA via the PCIe.

3.2.2.2 TSON Path Computation (TPC) API

The TPC API is a python API developed to calculate the path in the TSON network. Two parameters are considered. These are bandwidth and latency. The shortest path algorithm is used to determine the suitable paths according to the received request. The latency on the TSON network depends on the bandwidth and the distribution of the active time slices and the size of the packet. The lowest latency, according to the bandwidth, is achieved when all the available time slots are allocated. The estimation of the latency used by the TPC module is based on an empirical model. The latter is obtained by measuring the latency according to bandwidth. The size of the generated packets to measure the latency is random in order to emulate the real data traffic. The range of the random size is from 64 bytes to the 1500 bytes. The minimum, the maximum and the mean latency values are measured but for the estimation we consider only the mean value. Figure 3-7 summarises the results of the empirical experience and depicts the mean values of the latency.

The figure above shows that the latency decreases in function of the data rate. The high latencies for the low data rates are less than 1 ms. The different weights of each node are obtained based on Figure 3-7, where each latency value corresponds to a specific weight. In addition, the different paths of the different connection points have a specific fixed weight in function of the delay introduced by the link. According to data rate, the current topology and the available resources, a weight is attributed to the TSON node if it exists one or several paths provided by the shortest path algorithm satisfying the requested latency. The path corresponding to the lowest weight is transmitted to the OSM module. If the requested latency is not achieved the bandwidth is increased gradually to reduce the latency of the TSON node. For each step, the shortest path algorithm is applied until an available path is found.

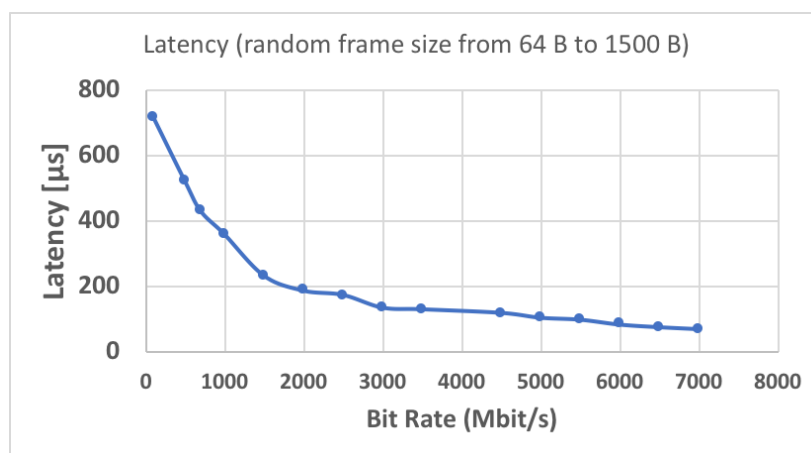


Figure 3-7 Estimation of the latency in the TSON network.

3.2.3 Function Design/Implementation/Evaluation

To evaluate the integration of the control plane and the influence of the TSON network on the data plane, we present the implemented testbed for the integration of MANO with TSON as shown in Figure 3-8. Indeed, two VNFs of the same Virtual Network Service (VNS) located in two different datacentres (DCs) can communicate via TSON network where the standalone TSON node are implemented and where each VNS uses one slice in the optical transport network identified by the VLAN tag.

OSM is used for orchestration and an ODL SDN controller is used to manage the TSON network. OSM manages the deployment of different VNSs where the VNFs are hosted in DC managed by OpenStack. In this test two VNSs are deployed.

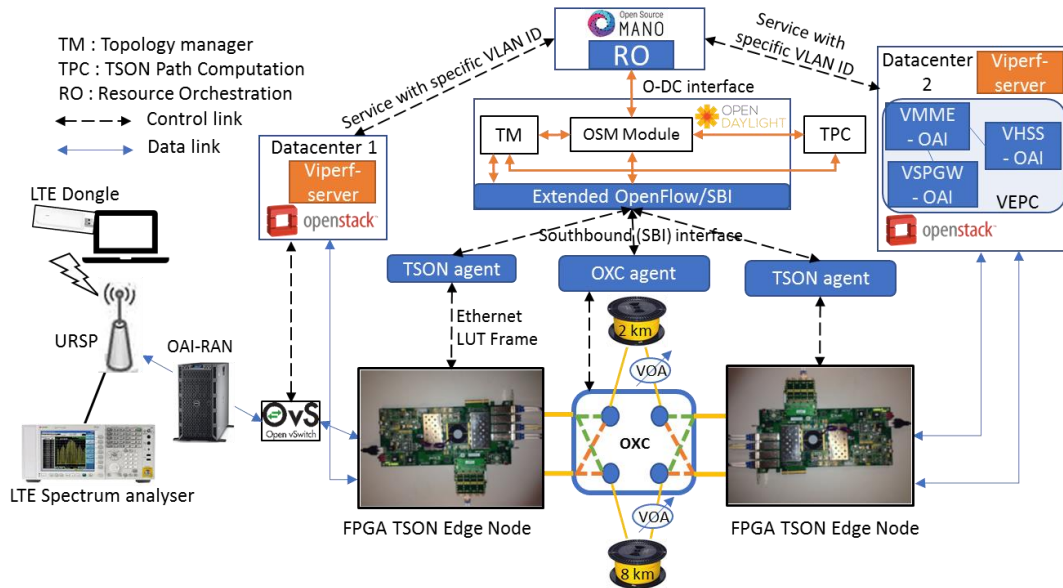


Figure 3-8. Evaluation of the DO DC and data plane in TSON network.

The first VNS is composed of 2 VNFs, where each VNF has the “*Iperf*” tool installed (to measure the performance of the transmission between two VMs over TSON); and both VNFs are in two different DCs. Also, an LTE network, as the second VNS, is implemented in parallel to emulate the BH traffic. The different components of EPC for LTE are virtualised using OAI, which is an open source LTE emulator. OAI combined with a Universal Software Radio Peripheral (USRP) is used here to emulate a RAN. An LTE dongle is connected to a computer to emulate the User Equipment (UE). The interface between the orchestrator and the ODL controller is a RESTCONF interface, where OSM sends a request to configure the end-to-end connectivity in the TSON network to the ODL controller. This request specifies the information about the TSON edge node, the entry points of the TSON node (input ports), the VLAN tag, the bandwidth, and the latency. The deployment of a VNS is carried out by OSM; this involves OSM first deploying the VNFs in the DCs. Once the VNFs are deployed in the DCs, they send the VLAN information about the network connected to the VNFs to OSM. OSM forwards the TSON relevant information along with the received VLAN IDs to the ODL SDN controller as shown in the Wireshark capture in Figure 3-2.

In ODL, an API called *OSM Module*, which receives the request to configure the optical transport nodes, has been developed. This module gets the TSON network topology from the Topology Manager (TM) to configure them. The TPC calculates the path in the TSON using the shortest path algorithm, utilizing the optical transport topology obtained from TM. The result of the path computation provides the outputs of the TSON nodes and the ports of the optical cross connection (OXC) switch to configure. The TPC sends the results to the OSM module that configures the different nodes by using an extended OpenFlow protocol developed to set the parameters of TSON node and the OXC switch. Concerning the southbound interface in Figure 3-8, the TSON agent and the OXC agent have been developed to receive the OpenFlow messages from ODL to translate them for configuring the TSON node and the OXC.

To evaluate the performance of data plane, the LTE VNS is deployed from OSM which has a 1 Gb/s bandwidth, 360 μ s latency and VLAN ID 100 where the input TSON port is 1. The second VNS for “*iperf*” has 1 Gb/s

bandwidth, 500 μ s latency and VLAN ID 101 where the input TSON port is 2. 2 km of optical fibre is allocated to the LTE VNS and 8 km of optical fibre is allocated to iperf VNS. Also, the UE in the LTE network service uses iperf tool to measure the performance of the mobile BH network based on TSON.

To show the impact of TSON, the result for the case without TSON where the two ends of the DCs are connected only via OXC are also computed and compared with the case where TSON is used. Figure 3-9 and Figure 3-10 show the performance of the data plane vs. the optical budget of the lost packets and jitter respectively. The average of the difference between the result with and without using TSON is about 2% for the “iperf” VNS and about 1.1% for the LTE VNS for the lost packets. The average difference in case of jitter is about 0.1 ms for the “iperf” VNS and about 0.9 ms for the LTE VNS. This slight increment of the lost packet and jitter is due to the data processing performed by the FPGA.

The radio performance is measured by connecting the Radio Frequency (RF) output port of the USRP to an LTE spectrum analyzer. The TSON network does not have any influence on the quality of the radio signal. Indeed, for an LTE Bandwidth of 10 MHz the data transmission rate is constant and equal to 12 Mb/s for the UE. In addition, the Error Vector magnitude (EVM) is 1.2 % on the RF downlink and 1.8 % on the RF uplink.

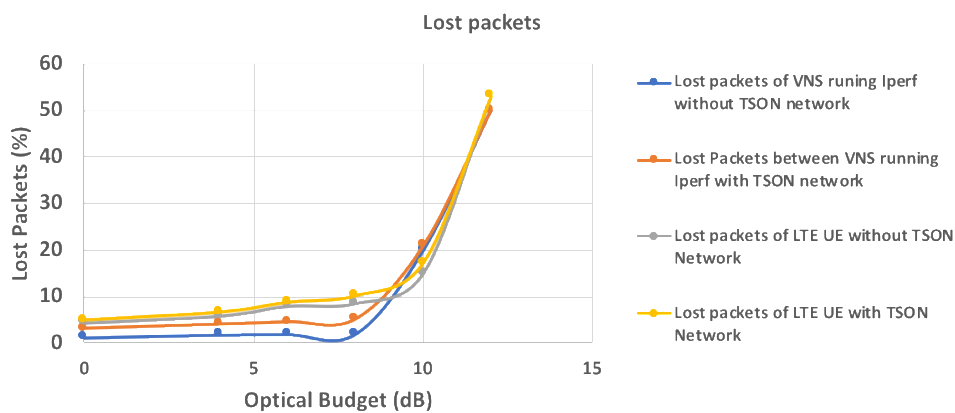


Figure 3-9: End-to-end packets lost.

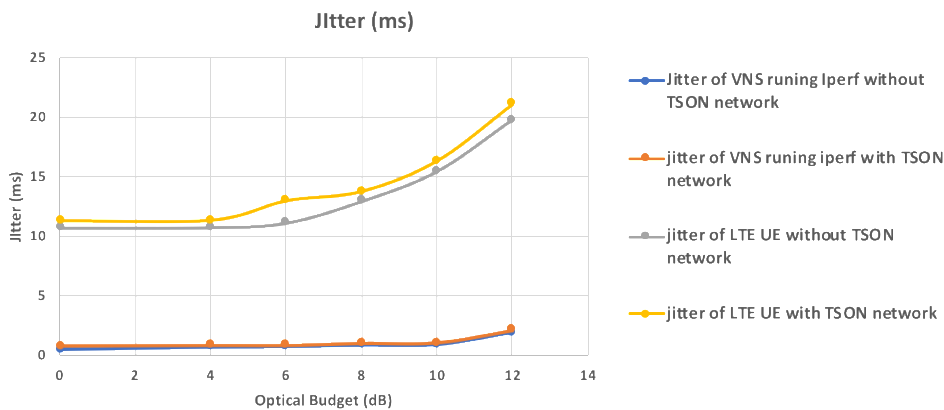


Figure 3-10: End-to-end measured Jitter.

3.2.4 Packaging for the 5G-OS

The Integration of the TSON control plane in the 5G OS is performed by the definition of the TSON Domain Orchestrator (DO) (Figure 3-11). Indeed, the TSON DO is composed by the OSM. The OSM has been chosen since it is able to manage the TSON connectivity and the Network Function Virtualised. The OSM uses the RESTCONF API to expose the TSON edge node topology to the Multi Domain Orchestrator (MDO). It also gets the request from the MDO indicating the endpoints, the VLAN tags, and the QoS corresponding to the bandwidth and the latency. The WAN infrastructure Manager (WIM) connector in the Resource Orchestrator (RO) is implemented to send the RESTCONF request to the OSM module in the TSON Domain Controller.

Based on the TSON topology edge node exposed by the TSON DO, the MDO is able to split the request and sends the right configuration parameter to the TSON DO, the TSON DO initiates the deployment of the NFVs if necessary and sends the parameters to configure the TSON network to the TSON DC. The OSM module get the request and sends it to the TPC, which has an overview of global topology of the TSON network. The latter sends back to the OSM module the different nodes, ports and parameters to configure the end-to-end-path. The OSM module uses the OpenFlow APIs to send the appropriate OpenFlow message to the right SDN agent to configure the TSON node.

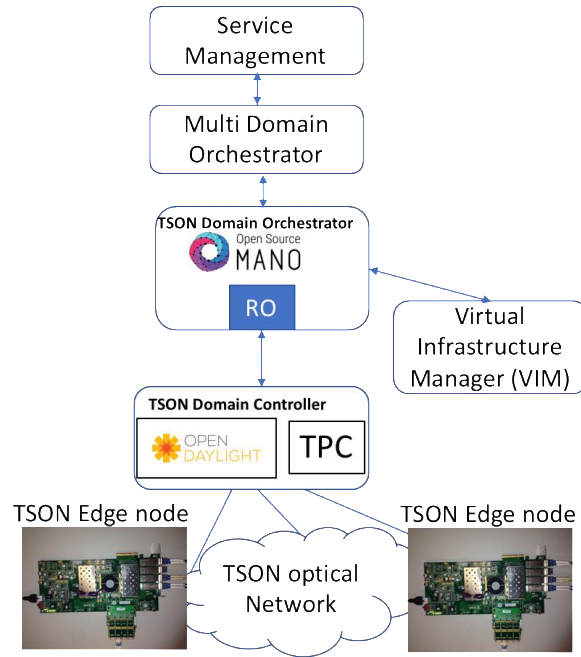


Figure 3-11. TSON network integrated in the 5G OS.

3.3 Technical Component 2: Flex-E

3.3.1 Summary Description

Flex-E interface technology [34] introduces the notion of “hard” slicing in the transport network, realising the concept of logically isolated Ethernet flows operating on common links but avoid influencing negatively the performance of each other in case of congestion. Furthermore, Flex-E can utilize fully the capacity of Network Processing Units (NPU) without waiting for future Ethernet rates to be standardised, while it supports a variety of Ethernet MAC rates independently of the Ethernet PHY rate being utilised. In this section, we provide technology primitives and functionalities while, the performance of a Flex-E demonstrator is evaluated in terms of achieved traffic protection and isolation, average throughput and latency.

3.3.2 Used programmable platforms and APIs

The Flex-E solution is demonstrated using PTN990 series of Huawei Optical Routers. PTN devices are primarily used on bearer networks that carry various services, such as mobile communication, enterprise users’ services. In more detail the OptiX PTN 990 is primarily used at the access or aggregation layers of a metropolitan transport network. It transports packet services on the network and converges them to an IP/MPLS backbone network. The Flex-E testbed description is provided in deliverable D3.1 [4]. The VRP 8.130 software is used V100R008C10, with a special patch that enables the Flex-E functionality. Note that Flex-E only runs under diagnose mode and still is not available for commercial use.

3.3.3 Function Design/Implementation/Evaluation

Flex-E technology is introduced as a thin layer, known as Flex-Shim, being able to support data rates out of the conventional range offered by current Ethernet standards. The main idea behind Flex-E is to decouple the actual PHY layer speed from the MAC layer speed of a client. Flex-E is based on a time-division multiplexing mechanism that is able to drive the asynchronous Ethernet flows over a synchronous schedule over multiple PHY layers. The main operational components of Flex-E include the following:

- Flex-E Client is an Ethernet flow based on a MAC data rate that may or may not correspond to any Ethernet PHY rate. The MAC rates currently supported are 10, 40, and $m \times 25$ Gb/s.
- Flex-E Group is a group of Ethernet PHYs that are bonded together. The Optical Internetworking Forum (OIF) supports Flex-E groups composed of one or more bonded 100GBASE-R PHYs. Higher rates like 400 GbE are under development in the IEEE P802.3bs project and will be supported in future Flex-E releases.
- Flex-E Shim is the layer that maps or de-maps the Flex-E clients over a Flex-E group. This procedure relies on a calendar-based slot scheduling. Essentially, a set of slots are assigned to each client, according to the MAC layer speed and group participation.

As we described in deliverable D4.1, there are three operational scenarios supported by Flex-E, which relate in a different way the MAC layer speed with the corresponding PHY speed (higher or lower), allowing a distinct manner for multiplexing clients in time. These are a) bonding that allows a MAC layer speed higher than a single PHY by grouping multiple PHYs to serve a flow (e.g. support a 200G MAC over two bonded 100GBASE-R PHYs); b) Sub-rating where MAC layer speed is less than the actual PHY; and c) channelisation which enables multiple Flex-E clients over a shared single PHY or bounded PHY.

Hybrids of these scenarios are also possible, for instance a sub-rate of a bonded PHY supporting 250G MAC over three bonded 100GBASE-R PHYs. These options allow increased resource flexibility for 5G and fine-tuning the offered rate depending on the usage. In this demonstrator, we will describe and evaluate the case of channelisation of a single 100G PHY.

Demonstrator Description

Flex-E operation

Flex-E introduces a Shim layer responsible for the mapping of Flex-E clients (i.e. Ethernet flows) to groups of PHYs. The Flex-E Shim layer is positioned between the Ethernet MAC and the Physical Coding Sublayer (PCS) of the PHY layer, as depicted in Figure 3-12. Each layer supports:

- Data Link Layer: a) Logical Link Control (LLC) for multiplexing network protocols over the same MAC, b) MAC Sublayer for addressing and channel access control mechanisms, and c) Reconciliation Sublayer (RS) that processes PHY local/remote fault messages.
- PHY Layer: a) PCS performs auto-negotiation and coding, b) Physical Medium Attachment (PMA) sublayer performs framing, octet synchronisation/ detection, scrambling/descrambling, and c) Physical Medium Dependent Sublayer (PMD) is the transceiver that is physical medium depended.

Each Flex-E client has its own separate MAC and RS above the Flex-E Shim, which operate at the client rate. The layers below the PCS are used intact as specified for Ethernet. As a first step in every Flex-E client flow, a 64b/66b encoding is performed to facilitate synchronisation procedures and allow a clock recovery and alignment of the data stream at the receiver. Then a procedure of idle insert/delete is performed. This step is necessary for all Flex-E clients in order to be rate-adapted, matching the clock of the Flex-E group according to IEEE 802.3. The rate of the adapted signal is slightly less than the rate of the Flex-E client in order to allow alignment markers on the PHYs of the Flex-E group. Then all the 66b blocks from each Flex-E client are distributed sequentially into the Flex-E group calendar where the multiplexing is performed.

Flex-E calendar (Figure 3-12): For each Flex-E group, a calendar is responsible to assign 66b block positions on sub-calendars on each PHY, to each of the Flex-E clients. The calendar has a length of 20 slots per 100G of Flex-E group and a bandwidth allocation of 5 Gb/s granularity, where a client may have any combination of slots in a group. To facilitate the demux process, the calendar is communicated along with the data. There are two calendar configurations for each PHY of the Flex-E group: the A calendar configuration (encoded as 0) and the B calendar configuration (encoded as 1). The two calendars are used to facilitate reconfiguration. Calendar slot are logically interleaved. A link failure is generated towards all Flex-E clients in the group once any PHY of the group fails. A control function manages the calendar slot allocation for each Flex-E client and inserts or extracts the Flex-E overhead on each Flex-E PHY in the transmit/receive direction. Calendar scheduling between the PHYs is currently performed on a Round Robin fashion. The calendar scheduling mechanism and the ability to adjust the slot allocation for guaranteed user performance, enables Flex-E to precisely "slice" the transport network.

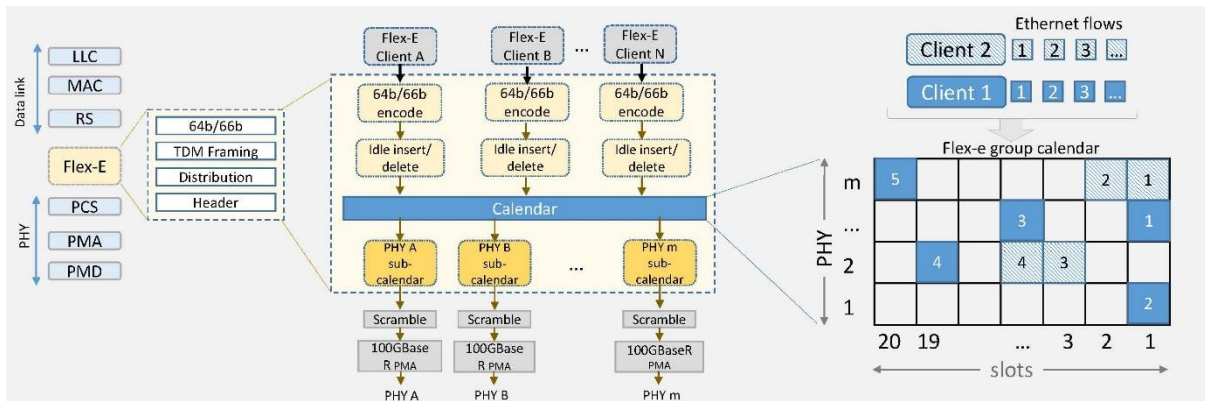


Figure 3-12: Flex-E layer between Ethernet MAC and PCS showing the FlexE Shim distribute/aggregate sub-layer in PCS/PMD.

System and Experiment Description

We demonstrate and evaluate Flex-E technology, while verifying the theoretical framework proposed by OIF. The system demonstrator under test, is depicted in Figure 3-13 where for the implementation of the testbed two Huawei Optix PTN 990 are used. A software patch on VRP V100R008C10, supports the necessary Flex-E functionality.

In principle channelisation enables multiple Flex-E clients over a shared single PHY or bounded PHY via the means of time division multiplexing in the Flex-Shim (e.g. support 150G and a 50 MAC over two bonded 100GBASE-R PHYs).

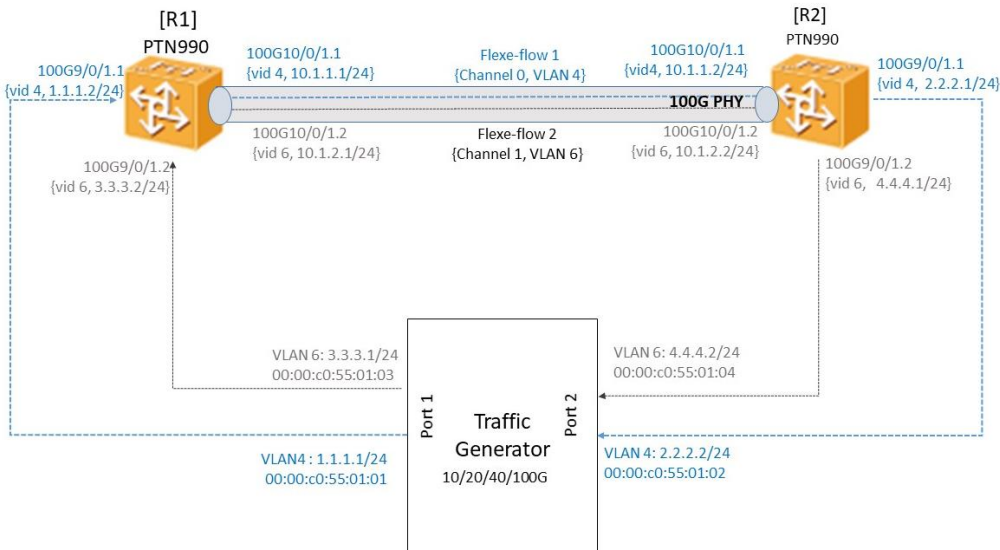


Figure 3-13: Flex-E demonstrator experimental setup.

```
[~R1]diagnose
Warning: Enter diagnose view, return user view with Ctrl+Z.
Info: The diagnose view is used to debug system hardware and software. Misuse of
certain commands in this view may affect system performance. Therefore, use the
se commands with the guidance of Huawei engineers.
[~R1-diagnose]
[~R1-diagnose]display pic slot 17 card 10 flex-ethernet all
All channel/Timeslot
channel [0]: 0 1 2 3 4 10 11 12 13 14 15 16 17 18 19
channel [1]:
channel [2]: 5 6 7 8 9
channel [3]:
```

Figure 3-14: Flex-E allocated slots per channel (4 channels supported).

In this demonstrator one 100G link between the two routers is “splitted” using the Flex-E technology. The splitting is made by creating Flex-E channels. The Flex-E clients are identified using VLAN technology and each VLAN is mapped to a specific Flex-E channel.

As presented in Figure 3-14 for the demonstrator we allocate time slots in two Flex-E channels, namely channel 0 and Channel 2. The mapping of channels in Flex-E flows is made on a VLAN basis where for the current implementation the lower two bits of the channel ID and VLAN ID correspond to each other (e.g., VLAN 4 maps to channel 0 and VLAN 5 to channel 1 and VLAN 6 in channel 2 and so on).

In Figure 3-14 a sample channel configuration is presented for router 1 (R1 prompt) while a similar configuration also exists for router 2. In the example depicted, slots 5 to 9 are allocated in channel 2, while all the other slots (1-4 and 6-20) are allocated in channel 0. With this allocation channel 0 receives 15/20=3/4 capacity (translated to scheduling opportunities) from the 100G interface, channel 2 receives 1/4 from the link capacity, while channels 1 and 3 are blocked. Because of the specific flow mapping implementation, channel 0 will support not only VLAN 4 but also all VLAN-ids where the ending bits are 00 (like 0b000, which is VLAN 8) and so on.

To create multiple Ethernet flows with the necessary VLAN identification, traffic has been generated by using a Huawei 100GE traffic generator, which was controlled using Tesegine 2.0 V300R006C10B410 software.

Using the Tesegine 2.0 traffic generator software, we were able to generate multiple concurrent flows with different configuration options of all the frames/packets fields (like src/dst MAC address, VLAN-id, src/dst IP address etc.). An example configuration is depicted in Figure 3-15.

As a baseline experiment, we created two flows: Flow-1 (VLAN-id 4) that could be critical traffic and Flow-2 (VLAN -id 6) being the background traffic.

The goal of the demonstrator was to showcase the ability of Flex-E channelisation to provide precise capacity shares between the two competing channels/flows. Channel statistics were obtained using interface counters that were updated to report per channel reports like the one presented in Figure 3-16.

Isolation guarantees: From all the experiments performed, perfect isolation characteristics were obtained were not only the capacity ratios were respected but also channels 1 and 3 were blocking the traffic not mapped to “open” Flex-E channels. Even though we only present here a representative subset of results, our conclusions apply to a wide range of network parameters tuning and specifications.

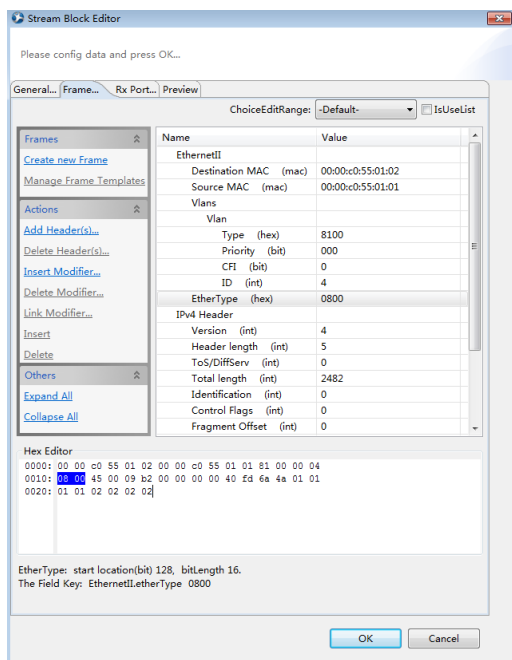


Figure 3-15: Tesegine traffic generator configuration example.

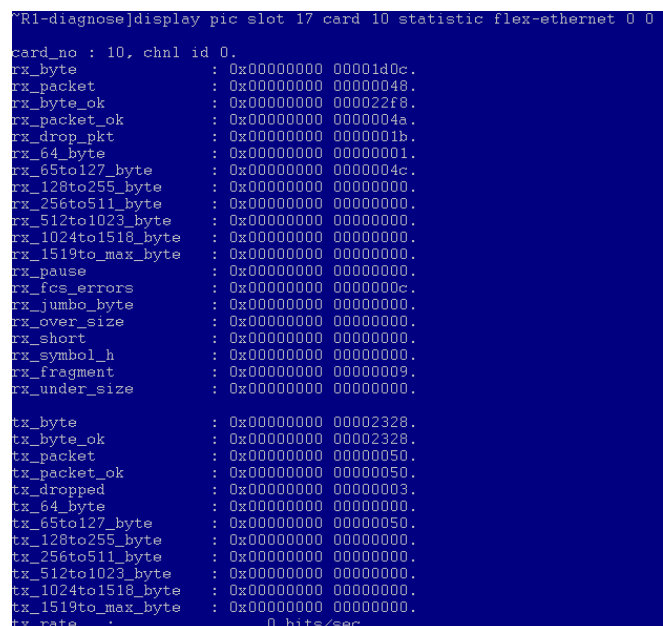


Figure 3-16: Per Flex-E channel statistics reports.

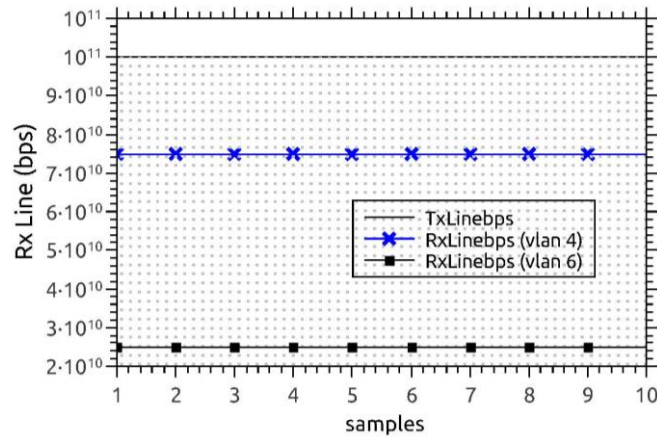


Figure 3-17: Network slicing performance using Flex-E channels over 100G PHY link for vlan4:15slots and vlan 6: 5 slots out of 20.

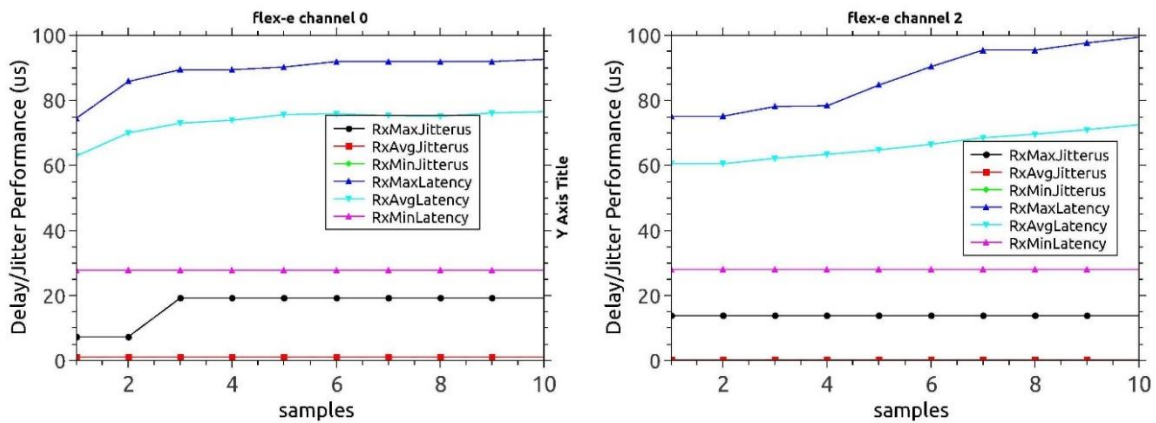


Figure 3-18: Delay and jitter results for two different Flex-E channels.

In Figure 3-17 we present the evaluation results of the experiment described above with two Flex-e channels serving to Ethernet flows that are identified using their VLAN-id. As we can see, the 100G link is precisely sliced exploiting time scheduling on the Flex-Calendar. Flex-E channel 0 throughput is 75 Gb/s on average while channel 2 throughput is 25 Gb/s on average (5 slots out of 25).

An important observation however is that although flex-e technology is able to provide precise throughput guarantees per flow using very low level slicing, is not able to differentiate delay and jitter per channel for each Flex-E client. This phenomenon was however expected and is depicted in Figure 3-18 where, as we can observe, both flows experience similar delay and jitter performance.

3.3.4 Packaging for the 5G OS

Currently, the control plane for the end-to-end provisioning of a Flex-E pipe is an open issue, yet to be specified. A GMPLS signalling through RSVP-TE approach is proposed in [12], while a software defined network (SDN) control with out-of-band signalling can be a potential alternative candidate. In both GMPLS and SDN cases, new data models need to be devised that expose the Flex-E information and functionalities to the control plane. Although the design of YANG models is possible over RSVP, new YANG models are expected to emerge specialised for Flex-E. As in all control plane models, the design primitives for the Flex-E control plane are security, scalability and fast convergence.

To on-board on 5G OS a possible service descriptor should include:

- 1) Flex-E Group provisioning, configuration and instantiation operations: Routers must advertise the type of Flex-E support that they offer. The current calendar allocation and information like link delay and node delay. Regarding capabilities exposure auto-negotiation procedures also need to be defined.
- 2) Flex-E calendar scheduling: The control plane must be able to provide an efficient mechanism for the

optimal assignment of PHYs to a specific group, while also consider for the optimal slot allocation in the group calendar for each Flex-E client.

- 3) Establishment of Flex-E multi-hop paths: Existing solutions consider a pre-configured Command-Line Interface (CLI) based Flex-E group configuration and client assignment. Note that the most important functionality in order to have a functional Flex-E setup is that for each PHY the mux and demux share the same sub-calendar. Otherwise, it would be impossible to decode the slot information to a specific Flex-E client. In a multi-hop setup this information sharing can be challenging.
- 4) Dynamic calendar switching configurations: Control plane must support dynamic switching between calendar configurations (A or B) and allow modifying the configuration of Flex-E clients into calendar slots, based on SLAs and performance criteria.

3.4 Technical Component 3: X-Ethernet

3.4.1 Summary Description

X-Ethernet is a Huawei proprietary technology, where X stands for extended distance, expanded granularity and extremely low latency. X-Ethernet introduces Ethernet PCS switching based on the interface offered by Flexible-Ethernet. The switch device will redirect Flex-E Clients (64B/66B block streams) from its inbound port to its outbound port without waiting for the arrival of the whole Ethernet frame for FCS checksum and forwarding decision with table lookup. Therefore, all the time consuming procedure, such as encapsulation/decapsulation, queuing and table lookup can be removed.

3.4.2 Used programmable platforms and APIs (WP3)

The X-Ethernet physical testbed description together with X-Ethernet technology primitives of operation are provided in deliverable D3.2. For each X-Ethernet switch the solution is based on FPGA board, six 100G CFP2 optical module slots, two 10G SFP+ optical module slots, one Ethernet interface slot and one RS232 interface. The RS232 is for device management and control. In the following, we provide experimentation results from the execution of three test scenarios that were carried out and showcase the ability of X-Ethernet technology to satisfy challenging switching requirements for the integrated 5G-PICTURE network.

3.4.3 Function Design/Implementation/Evaluation

Experiment Environment

The experiment environment is shown in Figure 3-19. Three X-Ethernet prototypes are connected to each other and formed a network. A controller (PC) configures each of the devices via the RS232 port on each device. A CPRI tester is used to generate CPRI option 7 traffic [35]. A network performance tester is used to generate Ethernet traffic with 100 Gb/s maximum bit rate. CPRI is injected into the XE No.1 device and transport to XE No.2, then loop back to the CPRI tester. The Ethernet traffic pass through three XE devices one by one and loop back to the network performance tester. It should be noted that the connection between XE devices is through 100G links. Particularly, CPRI and Ethernet traffic share the same 100G link between XE No.1 and XE No.2. Moreover, the connection between Network performance tester and XE devices are also 100G link. Network performance tester generate an Ethernet traffic that has an effective data rate ranges from 0 to 100 Gb/s.

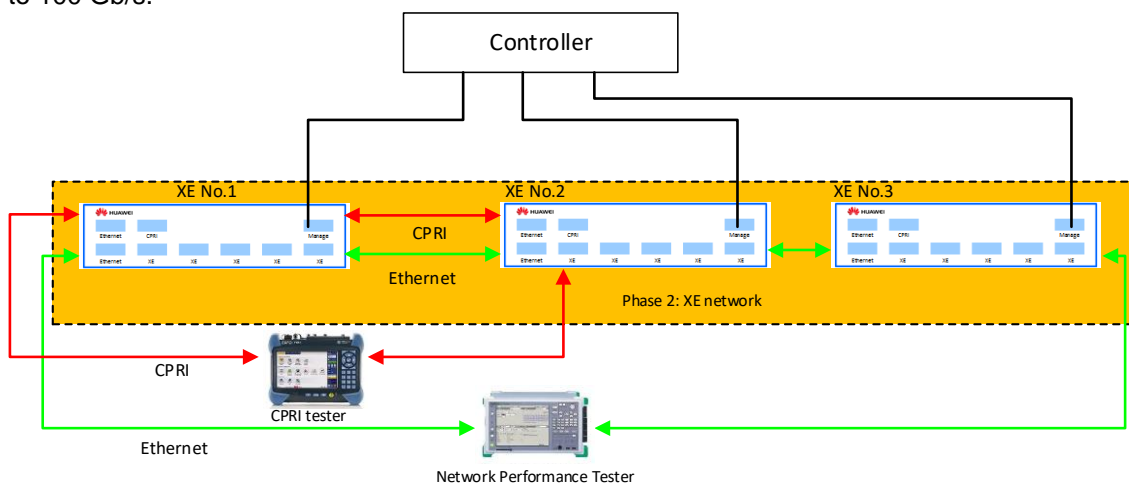
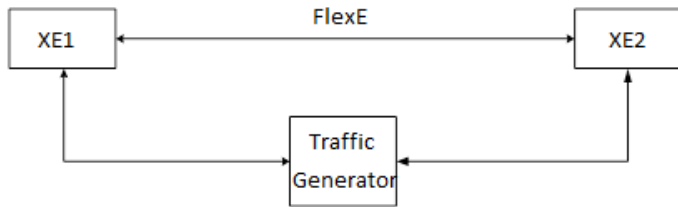
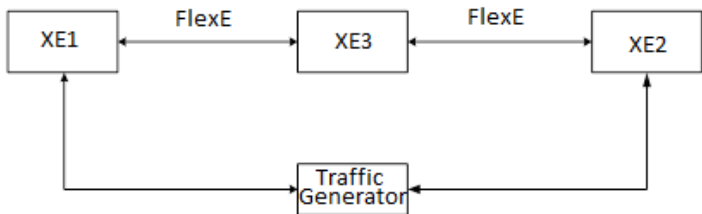


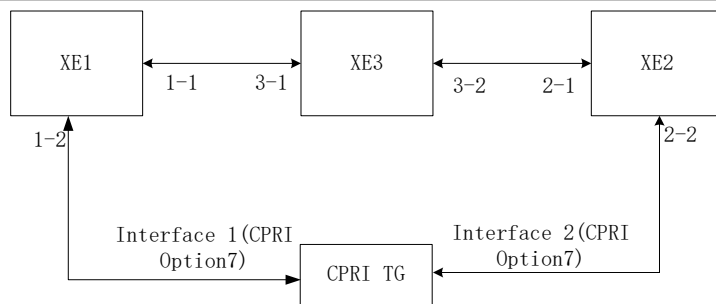
Figure 3-19: Experiment topology.

Test case 1: Ultra-low Latency Test for X-Ethernet Device as P node.

Objective	Testing latency of X-Ethernet device as P node					
Test instrument	Traffic Generator					
Configuration	<div style="text-align: center;">  <p>Figure 3-20: Test environment 1.</p>  <p>Figure 3-21: Test environment 2.</p> </div>					
Test Step	<p>a) Setup the test environment as shown in Figure 3-20.</p> <p>b) Configure end-to-end Ethernet traffic flow between XE1 and XE4 (Flex-E tunnel allocated 2/10 slots), the path is XE1-XE3. The TestCenter generate and send packets of length 128, record traffic latency T1.</p> <p>c) Setup the test environment as shown in Figure 3-21.</p> <p>d) Configure end-to-end Ethernet traffic flow between XE1 and XE4 through XE2, the path is XE1-XE3-XE2. The TestCenter generate and send packets of length 128, record traffic latency T2.</p> <p>e) Modify traffic packet length to 1518byte in sequence, and repeat the above operation, then record traffic latency.</p>					
Test Success Criteria	<p>a) The latency of P nodes can be calculated by the following formula: $(T2 - T1)$.</p> <p>b) Calculate and record forwarding latency of P nodes.</p>					
Test Result						
	Slot number	Packet length	T1 (μs)	T2 (μs)	Delay of P nodes (μs)	
	2	128 byte	2.988	3.539	0.551	
		1518 byte	4.309	4.903	0.594	
	10	128 byte	2.116	2.683	0.567	
		1518 byte	2.59	3.156	0.566	

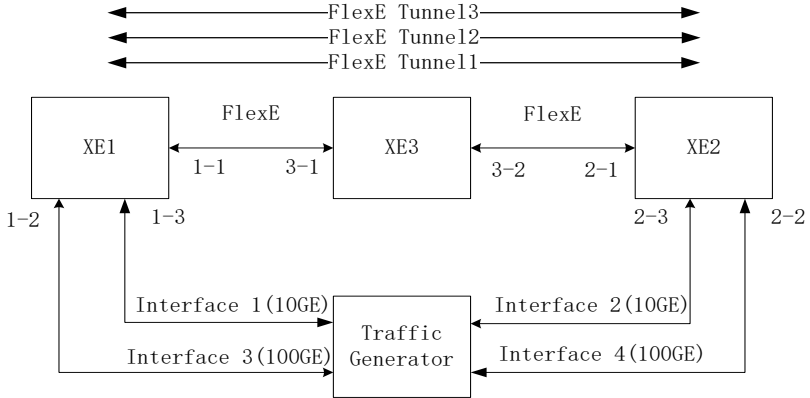
For the packet streams of different packet length at different rates, X-Ethernet exhibits its ultra-low latency forwarding capability at around 0.5 μs for P node. Compared to the classic router/switch (30 μs ~ 200 ms), X-Ethernet has a huge advantage in carrying latency sensitive services. Thanks to its L1.5 Non Stop Switching mechanism, all the time consuming procedures like table lookup, queuing, buffering, etc., are removed. What's more, due to its TDM like mechanism, it provides end-to-end hard isolation pipe, which guarantees a nanosecond level jitter in the transmission.

Test Case 2: Test Transport Capacity of CPRI over X-Ethernet Device.

Objective	Verify FlexE client number of the X-Ethernet device
Test instrument	CPRI traffic generator
Configuration	 <p style="text-align: center;">Figure 3-22: Test environment.</p>
Test Step	<ul style="list-style-type: none"> a) Setup the test environment as shown in Figure 3-22. b) Create FlexE Tunnel 1 and FlexE Tunnel 2 between the NNI(FlexE) of XE1 and XE2. Every FlexE Tunnel has 2 slot. c) Create the Ethernet traffic flow 2 (10GE) between the interface 1-2 of XE1 and the interface 2-2 of XE2, which is carried by the Tunnel 2. Start the TestCenter. d) Create the CPRI traffic flow 1(CPRI Option7) between the interface 1-3 of XE1 and the interface 2-2 of XE3, which is carried by the Tunnel 1. Start the MST5800.
Test Success Criteria	No alarm in CPRI traffic generator.
Test Result	No alarm in CPRI traffic generator.

The Common Public Radio Interface (CPRI) is the major FH traffic in 4G era. Many legacy devices still depend on CPRI. It also has strict requirement on the transport technology, like latency and frequency synchronisation. The test results shows that CPRI traffic can carried over X-Ethernet network. Combined with previous test case, X-Ethernet can carry CPRI and Ethernet/Packet traffic at the same time. Thus X-Ethernet can realize carrying FH and backhaul traffic on an integrated network.

Test Case 3: Multi-service Isolation Function Test of the X-Ethernet Device

Objective	Testing FlexE multi-service isolation function of the X-Ethernet device
Test instrument	Traffic Generator
Configuration	 <p style="text-align: center;">Figure 3-23: Test environment.</p>
Test Step	<ol style="list-style-type: none"> Setup the test environment as shown in Figure 3-23. Create three FlexE Tunnel 1/2/3 between XE1 and XE2, which allocated 2/4/4 slots. Create 3 Ethernet traffic flow 1/2/3 between the XE1 and the XE2, which are carried by the above three FlexE tunnel 1/2/3 respectively. Traffic flow 1: the nterface 1-3 to the interface 2-3; traffic flow2: the interface 1-2(VLAN100) to the interface 2-2(VLAN100); traffic flow3: the interface 1-2(VLAN200) to the interface 2-2(VLAN200). The TestCenter creates three bidirectional Ethernet traffic flow described above, traffic rates are 10G/20G/20G respectively. The TestCenter generates three bidirectional traffic flow described above, and then check whether the TextCenter can receive traffic normally. The TestCenter generates and send three traffic flow described above, and increases traffic flow 2 rate to 30G, and then check whether the TextCenter can receive traffic normally.
Test Success Criteria	<ol style="list-style-type: none"> The TestCenter can receive the three traffic flow normally, and no packet loss. There same packet loss in traffic flow 2. The traffic flow 1/3 are normally, no packet loss.
Test Result	No packet loss in the test.

The test result shows that overloaded in one X-Ethernet channel will not affect other traffic streams in other X-Ethernet channels. Thus, the hard isolation features of X-Ethernet can be demonstrated. For practical deployments, the network with X-Ethernet technology can carry different kinds of services, e.g. mobile transport, residential access and leased line services. Each of these services can be hard isolated, and will not be affected by each other.

3.4.4 Packaging for the 5G OS

To enable programmability of the X-Ethernet solution a new southbound protocol should be designed or exploit/device a new NETCONF protocol-based solution [36]. In the case of NETCONF protocol, a NETCONF server should operate on the switching device to interpret messages send from a remote SDN controller (implementing the NETCONF client). NETCONF is connected oriented using TCP while messages are encoded in XML and encrypted by SSH. An SDN controller like ODL or RYU could be used to implement this functionality and be directly connected to the rest of the 5G OS solution.

Table 3-1: NETCONF protocol operations.

Operation	Description
<get>	Retrieve running configuration and device state information
<get-config>	Retrieve all or part of a specified configuration datastore
<edit-config>	Edit a config. datastore by creating, deleting, merging or replacing content
<copy-config>	Copy an entire configuration datastore to another configuration datastore
<delete-config>	Delete a configuration datastore
<lock>	Lock an entire configuration datastore of a device
<unlock>	Release a configuration datastore lock
<close-session>	Request graceful termination of a NETCONF session
<kill-session>	Force the termination of a NETCONF session

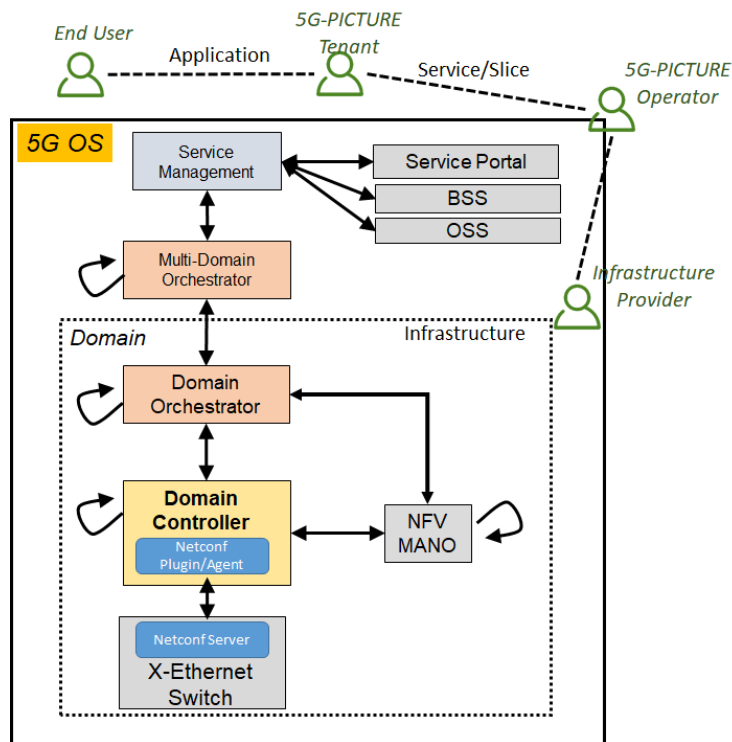


Figure 3-24: NETCONF-based programmability of X-Ethernet switches.

A simplified version of the 5G OS integrated with a NETCONF based solution for the programmability of the X-Ethernet solution is depicted in Figure 3-24. Depending on the NETCONF approach with the appropriate YANG models RESTCONF API can also be exploited. RESTCONF is a REST like protocol running over HTTP for accessing data defined in YANG using datastores defined in NETCONF. RESTCONF is an IETF draft that describes how to map a YANG specification to a RESTful interface.

Note however that as X-Ethernet is a Huawei proprietary experimental solution and the focus in the development phase is on the dataplane operations. No YANG models are currently available or under development. However, these are planned to be implemented the following period, exploiting NETCONF protocol enabling integration with the 5G OS.

For an orchestration and management system like 5G-OS a possible X-Ethernet service descriptor should include: VLAN ID, Flex-E client bandwidth, Flex-E group ID, PHY ID, Flex-E calendar slot number, performance monitoring information, system-type. data mode, client signal type.

3.5 Technical Component 4: Segment Routing

3.5.1 Summary Description

In 5G-PICTURE TSON, Flex-E and X-Ethernet are the key enabling technologies that are used to provide transport network slicing functionality for the wired network. These can be used for all types of transport network (namely FH, MH and BH as described in 3GPP [37]) and support connectivity between the disaggregated RAN functions. However, although these technologies are able to provide throughput guarantees for each slice of the transport network, they are not able to meet other objectives like for example delay guarantees or fast routing protocol convergence times. In 5G-PICTURE we exploit segment routing as a technology that is able to provide service guarantees and support advanced functionalities for the virtualised network in Layer 3.

Segment routing (SR) is a new protocol designed to forward data packets on a network based on the source based routing paradigm. However, instead of performing routing based on a node to node basis, segment routing divides a network path into several segments. Each forwarding path is constructed based on sequentially arranged segment list. A segment may be associated with a service instruction, with a node, a link or a path. Segment routing is defined in IETF RFC 8402 [38] and more information can be also found in [39] and [40]. Segment routing is expected to play a key role in deterministic networks like the ones defined by IETF in the Detnet working group [41] and networks where “plain” VPN solution are not enough, since besides encryption by means of performance existing VPN solutions are actually best effort and are not able to provide service guarantees to the virtual “sliced” network. Even when MPLS-TE solutions are deployed the end-to-end network performance is subject to the routing protocol behaviour and the policy used. The need for enhanced VPN solutions is described by IETF in [42].

Segment routing offers a number of benefits like simplification of the control plane of MPLS type of networks, efficient topology independent-loop-free alternate fast re-routing protection, higher network capacity expansion capabilities, smooth integration of SDN technology while it can also be used as an enabling technology for deterministic networking.

For this technical component analysis, a segment routing testbed was built based on Linux container technology and virtual router infrastructure. In the following subsections we provide background information for the basic segment routing functionality and terminology used, we analyse the testbed built, together with the configuration primitives. Relevant information for the functionality of the testbed, messaging and the protocol signalling are also presented.

3.5.2 Used programmable platforms and APIs

For the goals of this technical component analysis, no technology specific HW platform was required and all the infrastructure has been built using virtualisation technology. In more detail, a single Ubuntu Linux 16.04 Virtual Machine has been used to host a LXN hypervisor to create and to manage a number of LXC containers. LXC is an operating-system-level virtualisation method for running multiple isolated Linux systems on a control host using a single Linux kernel⁵. Table 3-2 summarizes the basic software tools used to implement the virtualised segment routing testbed. The VM was running over the VirtualBox hypervisor system over a commercial hardware equipped with Intel Core i7-6600U, 2.6 GHz with 16GB memory and 64bit OS.

Virtual routers based on the FRRouting (FRR) software package⁶ were deployed in LXC containers (a single LXC container hosts a single instance of a FRR router) and connectivity between routers was achieved using Linux bridging. FRRouting (FRR) is an IP routing protocol suite for Linux and Unix platforms which includes protocol daemons for BGP, IS-IS, LDP, OSPF, PIM, and RIP. FRR’s seamless integration with the native Linux/Unix IP networking stacks makes it applicable to a wide variety of use cases including connecting hosts/VMs/containers to the network, advertising network services, LAN switching and routing, Internet access routers, and Internet peering. In the context of the H2020-ICT-2014 project 5GEx an experimental support of Segment Routing for the MPLS dataplane in FRR was devised, which is the one we also exploit for this demonstrator.

⁵ <https://linuxcontainers.org/lxc/> and <https://linuxcontainers.org/lxd/>

⁶ <https://frrouting.org/>

Table 3-2: Basic software components used to build a virtual segment routing testbed.

Key software component	Description
LXD https://github.com/lxc/lxd-pkg-ubuntu	LXD is a lightweight container hypervisor. LXD uses LXC under the covers for some container management tasks. However, it keeps its own container configuration information and has its own conventions, so that it is best not to use classic LXC commands by hand with LXD containers.
LXC https://github.com/lxc/lxc https://www.sdxcentral.com/cloud/containers/definitions/containers-vs-vms/	LXC (Linux Containers) is an operating-system-level virtualisation method for running multiple isolated Linux systems (containers) on a control host using a single Linux kernel. A daemon that maintains the compartmentalisation between containers, while connecting their workloads to the kernel.
FRR routing (software router) https://frrouting.org/	FRR is a Quagga fork and is Linux routing software package that provides TCP/IP based routing services with routing protocols support such as RIPv1, RIPv2, RIPv6, OSPFv2, OSPFv3, IS-IS, BGP-4, and BGP-4+. FRR also supports special BGP Route Reflector and Route Server behavior. In addition to traditional IPv4 routing protocols, FRR also supports IPv6 routing protocols and new features like Segment routing.

3.5.3 Function Design/Implementation/Evaluation

3.5.3.1 Primitives of Segment Routing Technology

Segment routing divides a network path into several segments and assigns a segment ID to each segment and network forwarding node. The segments and nodes are sequentially arranged (segment list) to form a forwarding path. A summary table with the basic terminology used in segment routing is depicted in Table 3-3.

Segment routing encodes the segment list identifying a forwarding path into a data packet header. The segment ID is transmitted along with the packet. After receiving the data packet, the receive end parses the segment list. If the top segment ID in the segment list identifies the local node, the node removes the segment ID and proceeds with the follow-up procedure. If the top segment ID does not identify the local node, the node uses the Equal Cost Multiple Path (ECMP) algorithm to forward the packet to a next node.

A prefix segment indicates a destination address, and an adjacency segment indicates a link over which data packets travel. The prefix and adjacency segments are similar to the destination IP address and outbound interface, respectively, in conventional IP forwarding. Combining prefix (node) SIDs and adjacency SIDs in sequence can construct any network path. Every hop on a path identifies a next hop based on the segment information on the top of the label stack.

How SR information is distributed in the network

The SR architecture supports distributed, centralised, or a hybrid control plane. In the centralised case, an SDN solution could be exploited where for example BGP can be used to distribute SR information from a SDN controller, through Network Configuration Protocol (NETCONF), Path Computation Element Communication Protocol (PCEP) or BGP. In the decentralised case, segments are allocated and signalled by IGP protocols like IS-IS or OSPF. Segment routing uses an IGP to advertise topology information, prefix information, a segment routing global block (SRGB), and label information. To complete the preceding functions, the IGP extends some TLVs of protocol packets.

Over which dataplane technologies segment routing can operate?

For the moment an SR solution could smoothly be integrated and operate over two data-plane technologies, namely SR over MPLS (SR-MPLS) and SR over IPv6 (SRv6). In this demonstrator we will describe the case of SR over MPLS, where SR information is distributed using the OSPF routing protocol.

Table 3-3: Segment routing basic terminology.

Term	Description
Segment	an instruction a node executes on the incoming packet (e.g. forward packet according to shortest path to destination, or, forward packet through a specific interface, or, deliver the packet to a given application/service instance). Is represented by a Segment Identifier (SID).
SR Path	Connects an SR ingress to an SR egress, Can be different from the least cost path and contains one or more SR Segments.
Segment Identifier (SID)	Identifies the path fragment that the packet follows, while it can have node-local or domain-wide (a.k.a., global) significance.
SR domain	is a collection of SR capable devices.
SR-MPLS SID	an MPLS label or an index value into an MPLS label space explicitly associated with the segment.
SRv6 SID	an IPv6 address explicitly associated with the segment.
Active Segment	the segment that is used by the receiving router to process the packet. In the MPLS data plane, it is the top label. In the IPv6 data plane, it is the destination address.
PUSH	the operation consisting of the insertion of a segment at the top of the segment list. In SR-MPLS, the top of the segment list is the topmost (outer) label of the label stack. In SRv6, the top of the segment list is represented by the first segment in the Segment Routing Header.
NEXT	when the active segment is completed, NEXT is the operation consisting of the inspection of the next segment.
CONTINUE	the active segment is not completed; hence, it remains active. In SR-MPLS, the CONTINUE operation is implemented as a SWAP of the top label [RFC3031]. In SRv6, this is the plain IPv6 forwarding action of a regular IPv6 packet according to its destination address.
SR Global Block (SRGB)	Segment Routing Global Block (SRGB) is the range of labels reserved for segment routing. SRGB is local property of a segment routing node. In MPLS, architecture, SRGB is the set of local labels reserved for global segments. In segment routing, each node can be configured with a different SRGB value and hence the absolute SID value associated to an IGP Prefix Segment can change from node to node.
SR Local Block (SRLB)	local property of an SR node. If a node participates in multiple SR domains, there is one SRLB for each SR domain. In SR-MPLS, SRLB is a set of local labels reserved for local segments.
Global Segment	The instruction associated with the segment is defined at the SR domain level. A topological shortest-path segment to a given destination within an SR domain is a typical example of a global segment.
Local Segment	In SR-MPLS, this is a local label outside the SRGB. It may be part of the explicitly advertised SRLB. In SRv6, this can be any IPv6 address, i.e., the address may be part of the SRGB, but used such that it has local significance. The instruction associated with the segment is defined at the node level.
IGP Segment	the generic name for a segment attached to a piece of information advertised by a link-state IGP, e.g., an IGP prefix or an IGP adjacency.
IGP-Node Segment	an IGP-Node segment is an IGP-Prefix segment that identifies a specific router (e.g., a loopback). Also referred to as "Node Segment".
IGP-Prefix Segment	an IGP-Prefix segment is an IGP segment representing an IGP prefix. When an IGP-Prefix segment is global within the SR IGP instance/topology, it identifies an instruction to forward the packet along the path computed using the routing algorithm specified in the algorithm field, in the topology, and in the IGP instance where it is advertised. Also referred to as "prefix segment".

3.5.3.2 Testbed description

In Figure 3-25 the testbed built to present segment routing solution feasibility is presented. In more detail the testbed is comprised by 4 routers (labelled as R1, R2, R3 and R4) and 2 Hosts (H1, H2). Both the routers and the hosts were deployed in Linux LXC containers, using LXD hypervisor while, to bridge the different network namespaces, Linux bridging was used. For example in order to connect eth1 interface from host H1 with the eth1 interface of R1 and so on. The goal of this demonstrator is to verify the proper functionality of the segment routing solution, so it can be exploited in overall network sliced based solutions. The SR system demonstrated here is based on the extended FRR routing open source solution ⁷.

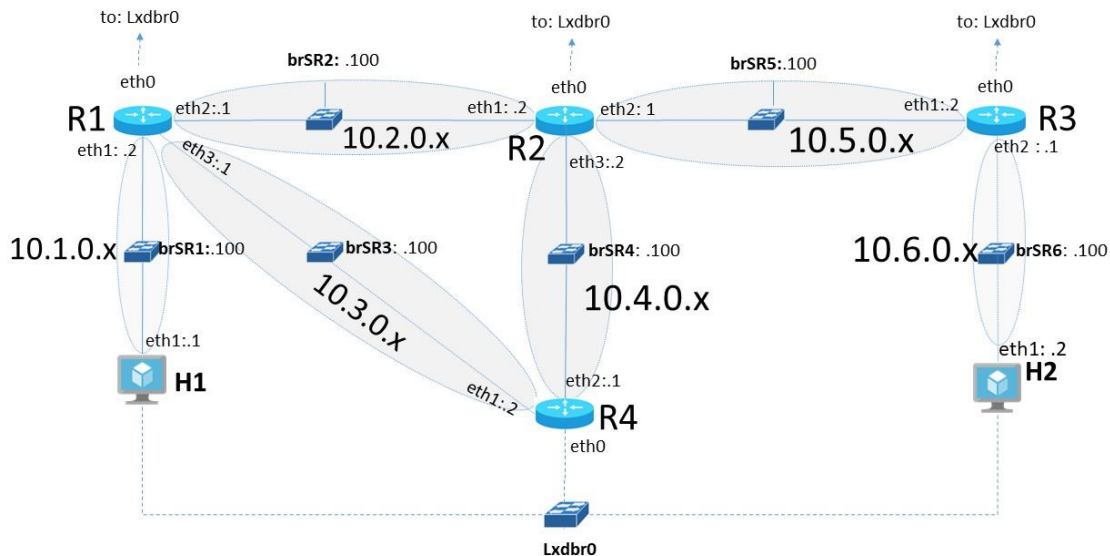


Figure 3-25: Segment routing testbed.

3.5.3.3 Experimentation results

In the data plane the solution we are using operates on top of MPLS. In Linux specific kernel modules, need to be enabled to support MPLS functionality. In principle SR can operate directly over MPLS with the main forwarding mechanism, where SIDs are encoded as MPLS labels. The segment list is encoded as a label stack and the segment to be processed is at the stack top. For the distribution of the SR segment information, OSPF link-state routing protocol is used that is extended according to [43].

In each router besides the normal network interfaces we also configured the loopback interface that was used to support the node-SID. In router R1 the loopback was 1.1.1.1, for router R2 the loopback was 2.2.2.2 and so on. In Figure 3-26 the OSPF configuration is depicted for router R1. Global-block field is used to configure the SRGB, node-msd is used for the maximum SID (labels) that can be stacked and the prefix is the SR Prefix Segment Identifier (with an index used for backward compatibility) and is mapped to the loopback.

Figure 3-26: Router R1 OSPF configuration.

```
!
! Zebra configuration saved from vty
! 2018/11/03 00:56:43
!
frr version 6.0
frr defaults traditional
!
hostname R1
log syslog informational
!
line vty
```

⁷ <https://github.com/FRRouting/frr/blob/master/doc/developer/ospf-sr.rst>

```

!
!
interface lo
 ip ospf area 0.0.0.0
!
interface eth1
 ip ospf area 0.0.0.0
!
interface eth2
 ip ospf area 0.0.0.0
!
interface eth3
 ip ospf area 0.0.0.0
!
router ospf
 ospf router-id 1.1.1.1
 capability opaque
 router-info area 0.0.0.0
 segment-routing on
 segment-routing node-msd 16
 segment-routing global-block 20000 29999
 segment-routing prefix 1.1.1.1/32 index 100 no-php-flag
!

```

Segment Routing requires some additional capabilities of the router to be advertised to other routers in the area. These SR capabilities are advertised in Router Information Opaque LSA message. In Figure 3-27 a sample LSA message sent by R1 is depicted presenting the relevant information after a network change. As we can see the relevant SR information is correctly included for Adj-SID sub TLV.

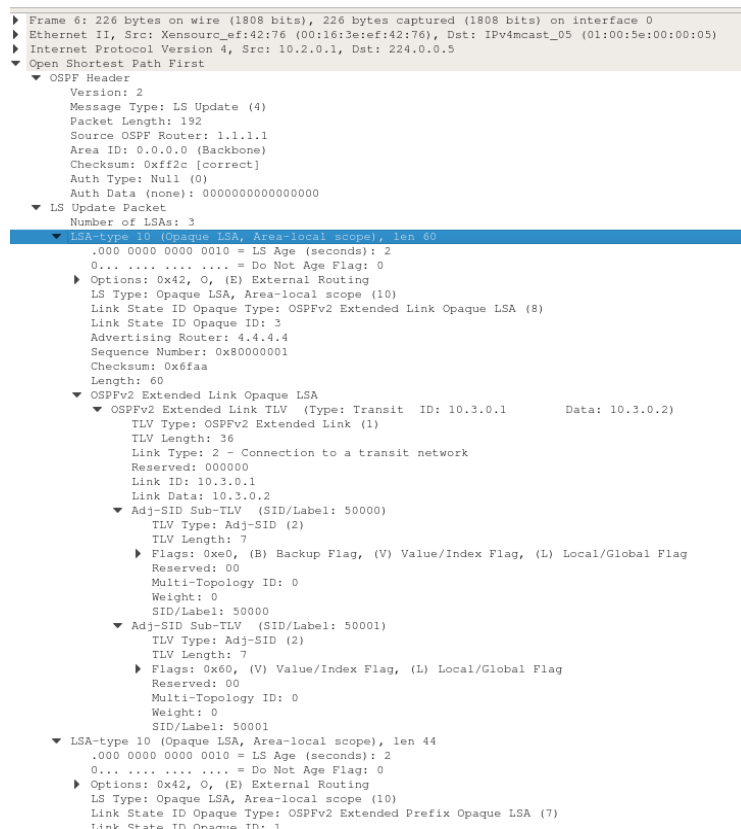


Figure 3-27: OSPF-SR LSA message.

Based on the OSPF message advertisements in all the routers the correct routing tables are constructed like also the relevant MPLS tables. In Figure 3-28 the IP routing table (left hand-side) and MPLS table (right hand-side) are presented. As we can see all the routing information for all the networks depicted in Figure 3-25 is correctly advertised through OSPF. The same holds for the Node-SIDs and Adj-SIDs that are depicted as SR type in the MPLS table of the router.

```
R4# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, E - EIGRP, N - NHRP,
       T - Table, v - VNC, V - VNC-Direct, A - Babel, D - SHARP,
       F - PBR,
       > - selected route, * - FIB route

K>* 0.0.0.0/0 [0/0] via 10.28.74.1, eth0, 00:00:30
O>* 1.1.1.1/32 [110/1] via 10.3.0.1, eth1, label 20100, 00:00:23
O>* 2.2.2.2/32 [110/1] via 10.4.0.2, eth2, label 20200, 00:00:23
O>* 3.3.3.3/32 [110/1] via 10.4.0.2, eth2, label 20300, 00:00:23
O 4.4.4.4/32 [110/0] is directly connected, lo, 00:00:30
C>* 4.4.4.4/32 is directly connected, lo, 00:00:30
O>* 10.1.0.0/24 [110/20] via 10.3.0.1, eth1, 00:00:23
O>* 10.2.0.0/24 [110/20] via 10.3.0.1, eth1, 00:00:23
   * via 10.4.0.2, eth2, 00:00:23
O 10.3.0.0/24 [110/10] is directly connected, eth1, 00:00:28
C>* 10.3.0.0/24 is directly connected, eth1, 00:00:30
O 10.4.0.0/24 [110/10] is directly connected, eth2, 00:00:30
C>* 10.4.0.0/24 is directly connected, eth2, 00:00:30
O>* 10.5.0.0/24 [110/20] via 10.4.0.2, eth2, 00:00:23
O>* 10.6.0.0/24 [110/30] via 10.4.0.2, eth2, 00:00:23
C>* 10.28.74.0/24 is directly connected, eth0, 00:00:30
```

```
R4# show mpls table
Inbound      Type      Nexthop      Outbound
Label        Type      Nexthop      Label
-----
20100        SR        10.3.0.1     20100
20200        SR        10.4.0.2     20200
20300        SR        10.4.0.2     20300
20400        SR        4.4.4.4     implicit-null
50000        SR        10.3.0.1     implicit-null
50001        SR        10.3.0.1     implicit-null
50002        SR        10.4.0.2     implicit-null
50003        SR        10.4.0.2     implicit-null
```

Figure 3-28: R4 routing table (left) - R4 MPLS table (right).

In all tests, to generate MPLS tagged traffic from the hosts, we used the open source Scaby Python library ⁸. Using the command line of FRR we can also verify that the correct SR information is advertised in the network. In Figure 3-29 the relevant SR information is depicted as stored in the R4 router using the `show ip ospf database segment-routing json` command.

Figure 3-29: SR database information for router R4.

```
{
  "20100":{
    "inLabel":20100,
    "nexthops":[
      {
        "type":"SR",
        "outLabel":20100,
        "distance":150,
        "nexthop":"10.3.0.1"
      }
    ]
  },
  "20200":{
    "inLabel":20200,
    "nexthops":[
      {
        "type":"SR",
        "outLabel":20200,
        "distance":150,
        "nexthop":"10.4.0.2"
      }
    ]
  },
  "20300":{
    "inLabel":20300,
    "nexthops":[
      {
        "type":"SR",
        "outLabel":20300,
        "distance":150,
        "nexthop":"10.4.0.2"
      }
    ]
  }
}
```

⁸ <https://github.com/secdev/scapy>

```

    ]
  },
  "20400":{
    "inLabel":20400,
    "nexthops":[
      {
        "type":"SR",
        "outLabel":3,
        "distance":150,
        "nexthop":"4.4.4.4"
      }
    ]
  },
  "50000":{
    "inLabel":50000,
    "nexthops":[
      {
        "type":"SR",
        "outLabel":3,
        "distance":150,
        "nexthop":"10.3.0.1"
      }
    ]
  },
  "50001":{
    "inLabel":50001,
    "nexthops":[
      {
        "type":"SR",
        "outLabel":3,
        "distance":150,
        "nexthop":"10.3.0.1"
      }
    ]
  },
  "50002":{
    "inLabel":50002,
    "nexthops":[
      {
        "type":"SR",
        "outLabel":3,
        "distance":150,
        "nexthop":"10.4.0.2"
      }
    ]
  },
  "50003":{
    "inLabel":50003,
    "nexthops":[
      {
        "type":"SR",
        "outLabel":3,
        "distance":150,
        "nexthop":"10.4.0.2"
      }
    ]
  }
}

```

We highlight that, since few SR implementations exist, this is an experimental support for SR in FRR, more details for the limitations are described in [44]. Although the analysis presented is representative for the issue, Segment Routing (SR) protocol and its ability to forward data packets based on source routing paradigm makes it as an appealing technology that is able to meet challenges in future transport networking.

3.5.4 Packaging for the 5G OS

In the case of SR, the interaction with the 5G OS depends on the control plane solution used. If an SDN solution is used to configure SR in each router of the network, this could be made through BGP or NETCONF. Information carried would be relevant with SR Node-SID, Adj-SIDs, loopback and interface network configuration etc., but also to the status of the network. In the case for example a link is down or a node is

down then the SDN controller needs to reconfigure everything. If, on the other hand, the SR control plane is exploiting an IGP solution like OSPF or IS-IS, then the 5G OS needs to act only on the management plane to enable the necessary OSPF features.

3.6 Technical Component 5: Open Packet Processing (OPP)

3.6.1 Summary Description

This section describes the functionalities provided programming the Open Packet Processor (OPP) engine described in deliverable 3.1 [4]. In particular, two different functionalities are described. The first is a stateful load balancer, while the second is a network function for handover management and will be used to provide session continuity during the railway demo.

3.6.2 Used programmable platforms and APIs

The functions presented in this section have been designed using as target platform the OPP presented in section 3.3 of deliverable D3.1 [4] and section 2.4 of deliverable D3.2 [5]. While now the design of the two functionalities is based on a table based configuration of the OPP pipeline, CNIT is currently developing a Domain Specific Language called XTRAlang (see section 3.4.3 of deliverable D3.2), which will facilitate the design of network functions to be executed by the OPP engine.

3.6.3 Function Design/Implementation/Evaluation

3.6.3.1 Load-driven forwarding load balancer

This network function implements a distributed load balancing algorithm designed for the widely used multi-rooted topologies (e.g. leaf-spine, fat trees). It uses special probes, which can be piggybacked on data or dedicated telemetry packets, which propagate utilisation information through the network. Probes frequency is adaptive and based on the estimation of flows' weight, i.e. lower the utilisation, and lower the probe frequency. This information is stored in the edge/aggregation switches, while the core switches only propagate the information enriching it with their estimations. In this way, every edge switch has its updated estimation of the best path for a given destination as its best-hop. Flows are divided in flow lets: bursts of packets belonging, for example, to the same 5-tuple, and divided by a significant amount of time (e.g. an inter-packet gap greater than $2 \cdot RTT$). By using flow let granularity, we can better utilise network resources especially in the presence of heavy hitter flows.

3.6.3.1.1 Design

Probes origin and replication. Each leaf sends probes to all the uplinks that connect it to the spines. Once the probes reach the spines, they add path information to the utilisation field and then replicate the packets sending them to all the leaves they see below, except the sender. When the leaves get the probe, they keep the information stored in and drop the packet.

Probe frequency. Probe frequency (F_p) is determined in an adaptive way. The leaves take a rate estimation in their up ports (used also for probing) and this estimation is quantised in such a way that if the rate is high, the probe frequency increases. F_p is packets dependent, i.e. it is measured in *incoming-packets/probe*, e.g. $F_p = 10$ means that a probe will be sent every 10 packets received.

Probe information. A probe utilisation header is composed of two fields:

1. **Leaf-id.** The leaf id is a number identifying the leaf which originated the packet;
2. **Utilisation.** The utilisation field contains the estimated *incoming* utilisation of the link.

Every switch maintains a link utilisation estimator per each port connected to the core network. The estimator is an Exponential Weighted Moving Average (EWMA) specified by this relation:

$$U_{now} = (1 - \mu) * P + \mu * U_{last},$$

where U_{now} is the actual estimator, U_{last} is the previous estimated value and P is the dimension in bytes of the packet. The μ parameter is a constant between 0 and 1 that adjusts the steepness of the exponential. Globally this estimator is a first order Infinite Impulse Response (IIR) filter with impulse response $h(P) = (1 - \mu) * \mu P$. The filter gives exponentially less weight to previous samples and it has a linear dependence with the delta of timestamps. In practice, if the packets entering the filter are close together (i.e. μ near to 1), the weight moves towards the previous samples of the estimator. On the contrary, if the packets are distant (i.e. μ near to 0), the weight moves towards the newest samples. In fact, the variation of the timestamps controls the steepness of

the exponential. These properties assure a rapid response to rate changes. Finally, in order to get the actual rate estimation, it is enough to divide the estimator by the delta of timestamps. Actually, making tests on the effective implementation of the filter, it appeared that the TCP bursts had a considerable variance, so instead of a single sample, it is provided to the filter an average on a certain amount of packets, e.g. 20.

Spines have these estimators as well. As opposed to leaves, they do not store the information about the best-hop for a given leaf, because when spines receive a packet belonging to a leaf, they can forward it through the only one link that connects to it. When a probe crosses a spine, the spine adds his utilisation estimation to the utilisation header, propagating the probe to the other leaves. For example, Figure 3-30 shows how the probe replication would be in a 3-to-2 leaf-spine topology.

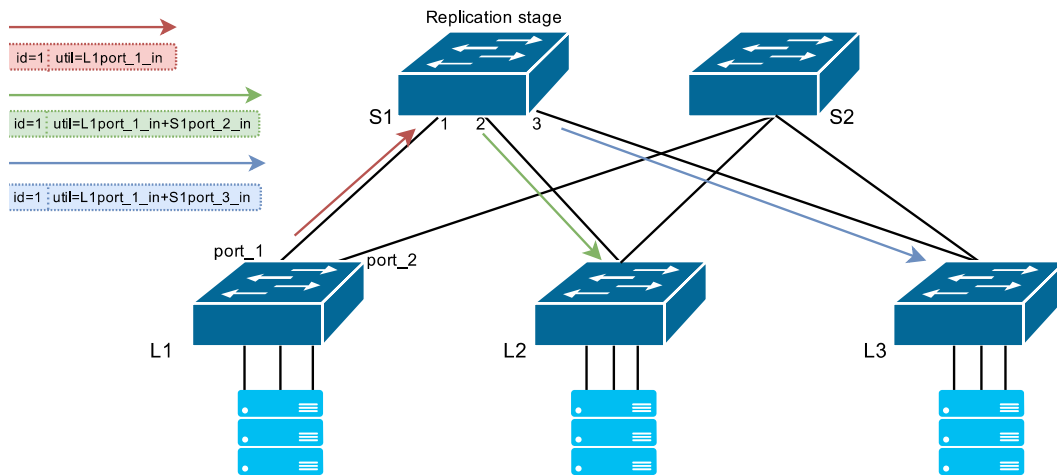


Figure 3-30: Probes logic.

3.6.3.1.2 Implementation

This use case requires the usage of four OPP stages to handle all the operations, even if the main logic of the load-driven forwarding algorithm is handled in the last stage. Figure 3-31 shows the pipeline describing the application.

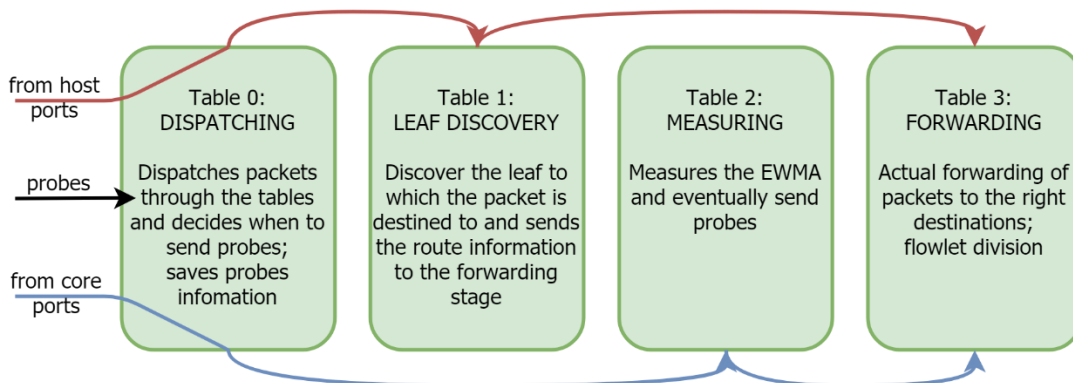


Figure 3-31: OPP pipeline.

There are three main paths between tables that a packet can follow:

1. *Packets from host ports*, after entering Table 0, go to Table 1 to discover to which leaf is intended the packet. Then they are directly sent to Table 3 for being forwarded.
2. *Probes* end their journey in Table 0: the utilisation information is stored in the appropriate registers and then the packet is dropped.
3. *Packets from network ports*, after entering table 0, go to Table 2 where they will be used as input for EWMA measuring. If Table 0 sends a special metadata, then it triggers a probe, which will be processed and sent in Table 2. In both cases the packets are sent to Table 3 and forwarded ordinarily to host ports.

3.6.3.2 Handover management for session continuity

One of the network functionalities that will be implemented using OPP is the handover management between the Typhoons transport nodes installed in the train and those installed in the towers for the 5G-PICTURE Railway demonstration. The OPP nodes will be used to guarantee the session continuity while the train is moving. Figure 3-32 shows a high-level picture of this network infrastructure.

The users travelling in the train can connect to the internet using the Access Points (APs) deployed in the train. The Various APs are connected to a switch that should forward the traffic to the two Typhoon nodes (T1 and T2). The T1 and T2 nodes are connected to some of the Typhoon i nodes deployed close to the railway. These nodes are connected using the TransPacket switches to a router via the OPP-2 node.

Since the connectivity of the T1 and T2 nodes change over time while the train is moving, a mechanism is needed to detect which of the two nodes is able to forward traffic at that time and forward the traffic only to the connected node. We planned to deploy an OPP node (OPP-1 in Figure 3-32) to monitor the connection state of the two typhoon nodes and to forward the traffic. As described in deliverable D3.1 [4] the OPP functionalities are designed as eXtended Finite State Machine (XFSM). In particular, for OPP-1 the designed functionalities are the following:

1. Periodically send probes packets (e.g. ICMP ping) toward the OPP-2 node to detect if the connection of a specific Typhoon node link is active; this functionality is performed by an XFSM that provides the active links as a metadata for the second XFSM.
2. The second XFSM performs the load balancing of the incoming traffic. When only one link is active, it sends all the traffic to the active link, otherwise it split the traffic between the two active connections.

Each of the Typhoon i nodes deployed close to the railway belong to a different VLAN network, thus allowing to the OPP-2 node to detect from which link the packet arrive. In more details, when the packets arrive to a specific Typhoon i nodes, the node insert a VLAN tag with $VLAN_ID=i$ and forward the packet toward the router. The OPP-2 node is therefore able to associate the MAC addresses of T1 and T2 to a specific VLAN, and send to this VLAN network the packets arriving from the router.

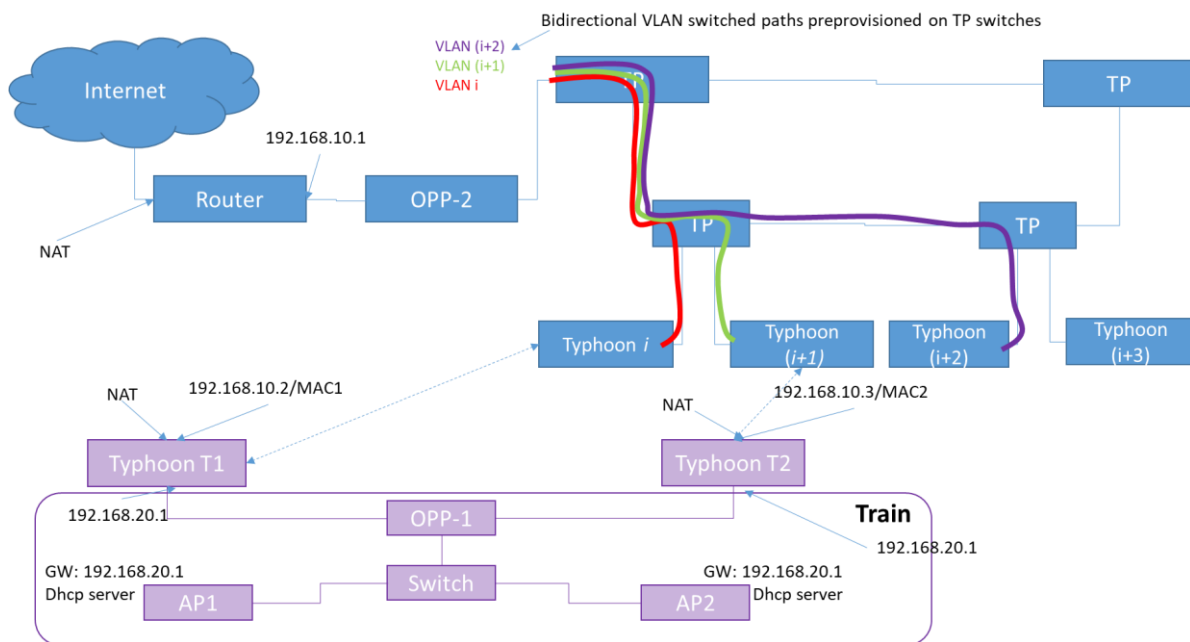


Figure 3-32: deployment of OPP nodes for handover management.

lookup_key: dst_MAC
update_key: src_MAC

Match	Action	Note
in_port=1, state=*	set_output_port=0, set_state(BIND), R1=VLAN_ID	Store the VLAN tag for the SRC_MAC address in R1
in_port=0, state=BIND	set_output_port=1, push_vlan(R1)	Push the VLAN tag for the DST_MAC address
in_port=0, state=*	set_output_port=1, push_vlan(BROADCAST)	Send the packet in broadcast

Figure 3-33: XFSM table of the OPP-2 node.

The details of the XFSM behaviour are depicted in Figure 3-33. The *in_port=0* represents the traffic coming from the router, and *in_port=1* represents the traffic coming from the Typhoon nodes. Setting the *update_key* as the source MAC allows learning the *VLAN_ID* associated to that specific source MAC. In particular, the XFSM stores in the R1 flow register *VLAN_ID*. On the other hand, the *lookup_key* search for the destination MAC and, if found, write the R1 value in the VLAN header. Otherwise, it sends the packet in broadcast to all Typhoon nodes in the network.

3.6.4 Packaging for the 5G-OS

Currently, the functions executed by the OPP engine can be stored as binary images in the 5G OS repository. The image can be downloaded into the OPP engine using a suitable agent that read the image and configure the OPP pipeline to execute the specific program. It is worth to notice that this description is platform dependent, i.e. if in the network there are different type of OPP engines (i.e. a software engine and a FPGA-based one), then different binary image must be stored into the repository.

However, CNIT is now developing the compiler for XTRAlang. The function described with XTRAlang will be first compiled to an intermediate target-independent representation, in form of a JSON file, to then be compiled with respect to a specific target (a specific HW platform). This improvement will allow to store into the 5G OS repository a descriptor containing the JSON file, which is target independent. In this case, the target dependent compilation will be performed by the agent when the JSON representation of the specific function to instantiate is sent to the agent.

3.7 Technical Component 6: Solution based on IEEE 802.11 technologies, both for access and BH (802.11ac modems)

3.7.1 Summary Description

This technical component is used to provide a wireless connectivity service over a distributed area, and it is instantiated over prototype Wi-Fi Small Cells being developed in WP3. Hereafter, we refer to this technical component as *SWAM: “SDN-based WiFi Small Cells with Joint Access-Backhaul and Multi-Tenant capabilities”*. In essence the services provided by SWAM can be divided in:

1. Service 1: Instantiate an access connectivity service composed of virtual APs over a set of physical Aps.
2. Service 2: Allocate a connection through the wireless backhaul, which transport the traffic from such access service until a fibre attachment point.

Figure 3-34, already included in deliverable D4.1 [3], describes an exemplary scenario where SWAM would be instantiated. In the figure we see a set of Wi-Fi Small Cells deployed outdoors, mounted on street furniture. These Small Cells offer both access, i.e. connection to the mobile devices of a specified Mobile Network Operator (MNO), and wireless BH, through a multi-hop mesh network that connects to various fibre attachment points. The technical component described in this section is a control and management system, that allows the 5G-PICTURE 5G OS to instantiate a connectivity service over the infrastructure described in Figure 3-34, whereby a given tenant specifies: i) the credentials of its access network connectivity service, ii) the Small Cells where it requires presence, and iii) a tunnel to its home network where traffic from that service should be delivered. The technical component at hand implements all the necessary plumbing to provision that service. In Figure 3-34 the mobile devices belonging to different MNOs are highlighted with red and blue bounding boxes.

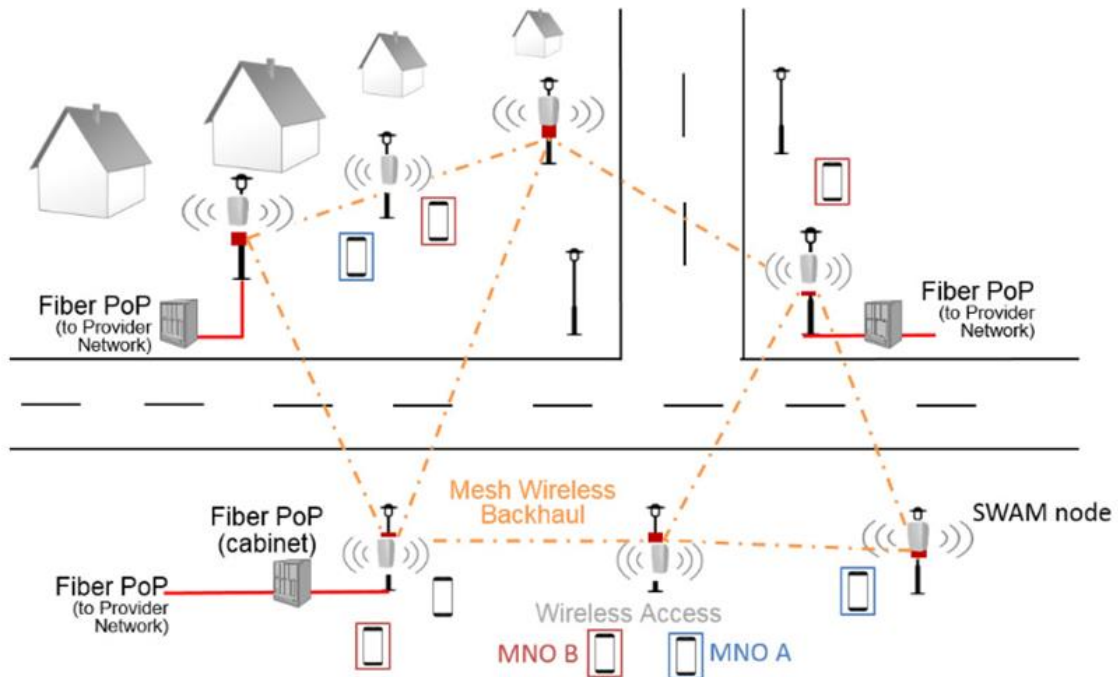


Figure 3-34. Deployment scenario for the joint access-backhaul function.

3.7.2 Used programmable platforms and APIs

This technical component uses is executed over the ARM based Gateworks SBCs introduced in deliverable D3.1 [4]. In addition, there are two software APIs that are used to control the previous devices:

1. A NETCONF based API, which is used by the management plane to manage the lifecycle of the virtual APs belonging to a specific connectivity service, i.e. deploy and delete them. This API has been developed as part of the work in WP3, and is described and evaluated in detail in deliverable D3.2 [5].
2. A second API used to deploy connections through the wireless backhaul, using OpenFlow and software switches. This API was developed as part of the 5G-XHaul project⁹, and is based on the Control and Orchestration Protocol (COP) [45].

3.7.3 Function Design/Implementation/Evaluation

3.7.3.1 SWAM design and implementation

Technically, SWAM is composed of the following components: i) the physical WiFi Small Cells featuring multiple interfaces (wireless Network Interface Cards – NICs); ii) a software-based data-path running on each physical Small Cell; and iii) the SWAM controller, featuring a BH module to instantiate paths over the wireless BH, a provisioning module, used to manage the lifecycle of virtual Access Points (vaps), and an access bridge module used to connect the vaps to the connections in the wireless BH. The left part of Figure 3-35 depicts the various components of the SWAM architecture.

⁹ <https://www.5g-xhaul-project.eu/>

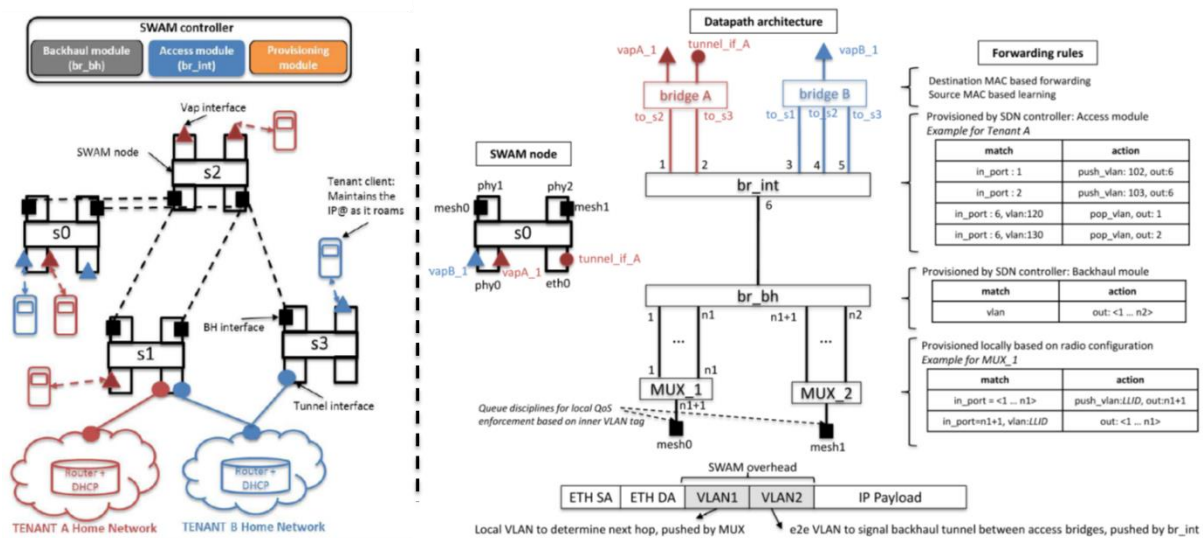


Figure 3-35. Design of the SWAM technical component.

The core of SWAM is the software based datapath depicted in the right part of Figure 3-35 where we can see an example of a SWAM node with three physical wireless interfaces and one Ethernet interface. One wireless interface is used to serve access traffic and instantiates two vap interfaces for tenant A and B, whereas the other two wireless interfaces are used for wireless backhaul and instantiate two mesh interfaces. The Ethernet interface connects to the wired network and instantiates a tunnel interface.

The goal of the SWAM datapath is to process packets coming from the tenants' customers (vap interfaces) and deliver them to the appropriate SWAM gateways through the wireless backhaul (mesh interfaces). A three level hierarchy of software switches is used for this purpose: i) Per-tenant access bridges, ii) the integration bridge (br_int), and iii) the backhaul bridge (br_bh).

The core idea behind the SWAM datapath is a logical separation between the access and the backhaul. The job of the wireless BH is to forward packets along a set of end-to-end tunnels, whereas the job of the access side is to match traffic coming from the tenants' vaps to the appropriate BH tunnels. In SWAM, a BH tunnel is defined using a VLAN tag, and provides a unidirectional connection between two interfaces of a per-tenant access bridge. The right part of Figure 3-35 depicts a sample SWAM datapath in node s0, where tenants A and B have instantiated vaps, along with their corresponding access bridges (brA and brB). Tenant A has also instantiated presence in SWAM nodes s2 and s3. Therefore, brA has two BH facing ports, representing respectively tunnels sA:0→1 and sA:0→2 for tenant A which, as depicted in the right part of Figure 3-35, they map to VLANs 102 and 103 in the integration bridge. Notice that BH tunnels are unidirectional, hence the reverse tunnels for tenant A, i.e. sA:1→0 and sA:2→0 map to different VLANs (120 and 130 in the right part of Figure 3-35). The interested reader is referred to [46] for a detailed description of the SWAM datapath.

In addition to the datapath, SWAM also features a specific control plane that for each virtual wireless access service selects, among all devices with connectivity to the wired network, a subset that will be used to receive traffic from the various vaps included in this service. In addition, the SWAM datapath enables mobility, whereby a client device can move between virtual APs belonging to the same wireless access service, while the BH paths are relocated.

As a consequence of the SWAM datapath introduced in the previous section, a given tenant enjoys a layer two abstraction whereby its access bridges are connected to each other forming a mesh, which may contain loops. Having loops is a problem because the access bridges are regular MAC learning bridges. SWAM contains control plane mechanisms that provide three main features: i) avoid loops in the resulting per-tenant overlays, ii) enable the possibility of using multiple gateways for a given tenant to balance load across the wireless BH, and iii) support client mobility, whereby when a client device hands over from one tenant vap to another the BH tunnels are re-configured appropriately.

Loop avoidance and support for multiple gateways: To avoid loops in the resulting tenant overlay the access module in the SWAM controller implements a traditional Spanning Tree algorithm to each tenant overlay. To apply Spanning Tree one SWAM node for each tenant needs to be appointed as the root node. In SWAM the

tenant root node is a SWAM gateway node having a tunnel interface to the tenant home network (c.f. Figure 3-35). Consequently, in the access bridges in the SWAM nodes for tenant t , we need to block all the ports $p<t:i>j$ linking to a BH tunnel other than the tunnel to the tenant root, i.e. $p<t:i>rt$, where rt in S is the SWAM node acting as root for tenant t , and $S = \{s_0, \dots, s_{|S|-1}\}$ is the set of all SWAM nodes. To block the ports in the SWAM datapath, the access module in the SWAM controller pushes a high priority drop rule into the integration bridge linked to the appropriate access port (c.f. Figure 3-35). Since all SWAM nodes have a direct tunnel to the root node, the resulting per-tenant overlay is a hub-and-spoke topology, which is loop-free. To enable load balancing, SWAM supports the allocation of multiple concurrent gateways for a given tenant. Multiple gateways are enabled simply by configuring the tenant access bridge in different non-root SWAM nodes to point to a different SWAM gateway using the appropriate drop rules in br_int . In this way, the resulting tenant overlay is partitioned in multiple sub-trees routed in the respective root nodes. In order to enable full network communication, the resulting overlay sub-trees are bridged in the tenant home network, using a regular bridge (not controlled by the SWAM controller).

Mobility support: client device attached to a vap of a given tenant should maintain connectivity while roaming through the network. There are two mechanisms involved in maintaining connectivity. First, the process of executing a handover between vaps, which is a standard feature supported by the Wi-Fi devices used in SWAM. Second, a client handover triggers an update of the path followed by the packets addressed to the client through the wireless BH. In order to maintain connectivity upon a client handover, the MAC lists in the access bridge in tenant root node and in the home network bridge, need to be updated in order to point to the tunnel connecting to the SWAM node where the client is currently attached. Handovers in SWAM are break before make, since the SWAM controller has no control over the client devices in order to instruct them when to execute the handover. Therefore, it is critical to properly update the MAC lists in the affected bridges as soon as a new point of attachment (vap) for a client device is detected. There are two mechanisms in SWAM to accomplish this. First, the control agent in the SWAM nodes is notified by the vaps when a new client attaches, and generates a spoofed ARP Requests. Second, we have observed that clients upon reassociation, spontaneously generate a broadcast layer-two packet for link discovery purposes (LLC-xID). This packet is forwarded to the root node through the newly attached tunnel, hence automatically updating the MAC tables appropriately.

3.7.3.2 SWAM evaluation

The SWAM architecture is implemented using Open vSwitch switches. Node s_2 is connected to a remote server were the SWAM controller, which is implemented using ODL, runs. In order to validate the SWAM architecture, we perform a series of experiments that showcase its functionality and performance during certain network events and under varying network conditions. For these evaluations, we set up a physical indoor testbed composed of five SWAM nodes built with the components shown in Figure 3-36. Each node is equipped with one IEEE 802.11ac wireless NIC to provide Wi-Fi connectivity to clients, and one or two IEEE 802.11ac NICs to establish wireless BH links. The layout of the five nodes within our office environment is depicted in Figure 3-36.

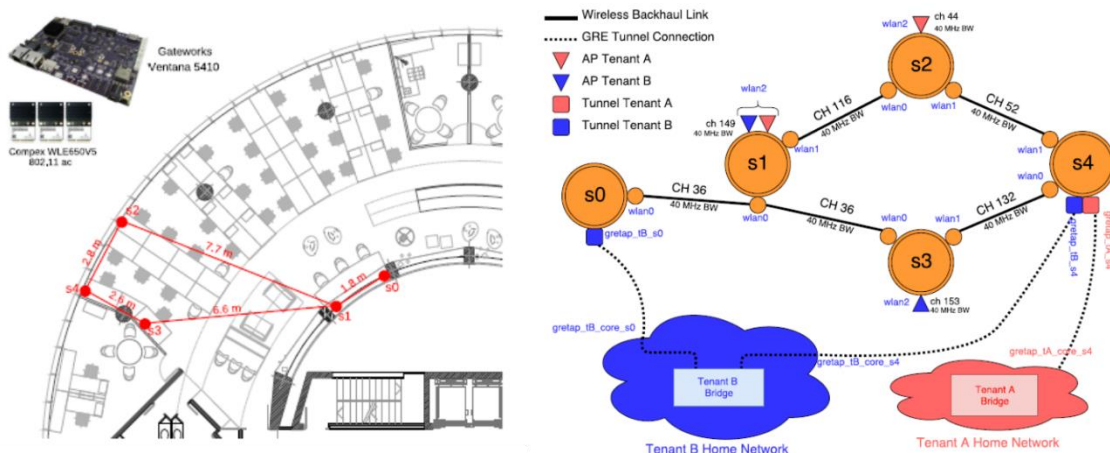


Figure 3-36. Testbed used in the evaluation.

To evaluate the performance of SWAM we carry out the following experiments: i) evaluate how SWAM isolates access and BH, ii) evaluate how traffic is affected when we relocate a SWAM root node within a slice, and ii) evaluate the impact of client mobility on an ongoing session.

Access-backhaul isolation: we show how SWAM reacts when a BH link carrying an end-to-end tunnel between two per-tenant access bridges breaks. To enable fast recovery, SWAM proactively installs backup paths and reallocates the end-to-end BH tunnel¹⁰. The left part of Figure 3-37 depicts the traffic flows transmitted in this experiment, before and after we break the BH link connecting nodes s1 and s3, which happens 60 seconds after initiating the experiment. Upon the link break, SWAM immediately reacts by reallocating STA B1's traffic to the only possible backup path towards s4 over the upper branch, as represented with a dashed line in the left part of Figure 3-37. The upper subplot in the right part of Figure 3-37 depicts the individual traffic flows of devices A.1, B.1 and B.2 before and after the link break. The lower subplot depicts the traffic of B.1 as measured in the s3 and s2 nodes, before and after the link break, demonstrating how SWAM is able to switch to a backup path in a seamless manner.

Gateway relocation: In the previous experiment, we can see that after the link breaks the overall network throughput reduces in about 10 Mb/s (from 83 Mb/s to 73 Mb/s). This can be addressed by the SWAM controller by relocating the ROOT node of Tenant B, given that Tenant B has two SWAM nodes that include tunnels to its home network, namely s4 and s0. In particular, by relocating the ROOT node for Tenant B in s1 to s0, the traffic of client B.1 will follow the path described in the left part of Figure 3-38, alleviating the link between s1 and s2, which carries traffic from client A.1.

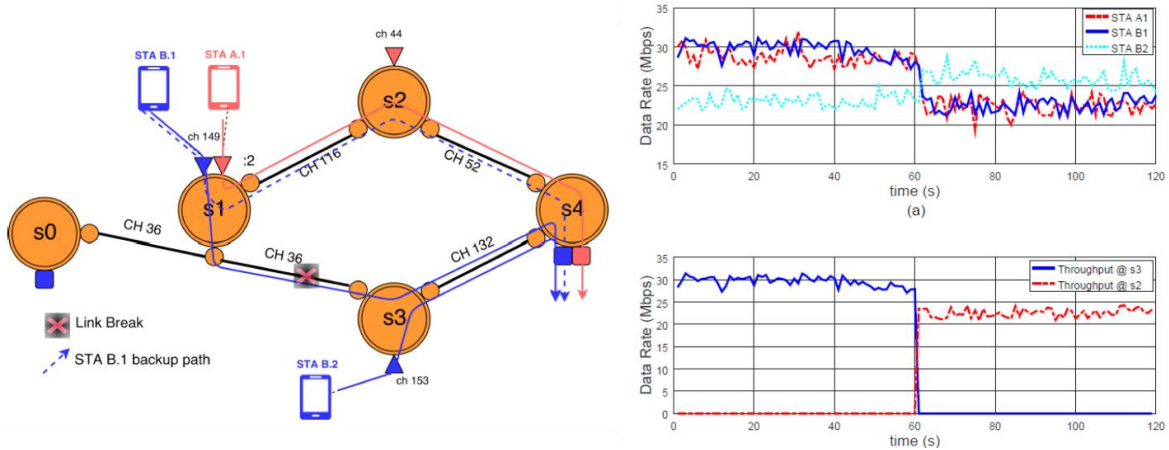


Figure 3-37. SWAM evaluation of access-backhaul isolation.

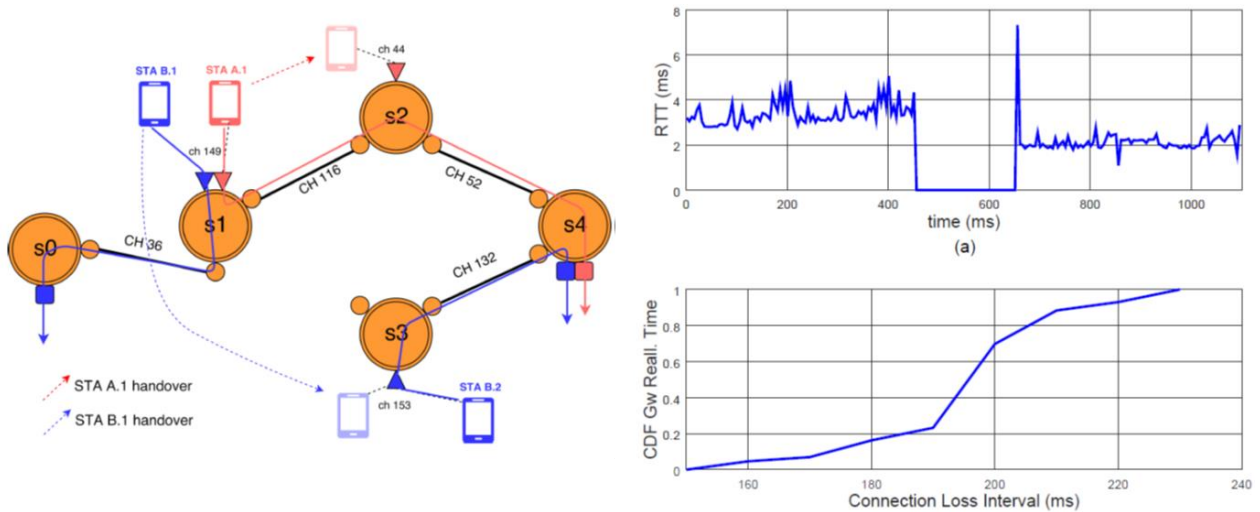


Figure 3-38. SWAM Gateway relocation evaluation.

¹⁰ <https://www.5g-picture-project.eu/download/swam.pdf>

To evaluate the impact of relocating a gateway on network performance, we perform two experiments. First, in order to measure the time the network is in outage, we send a continuous stream of ICMP messages every 5 ms using the ping tool and measure the RTT. Second, we study how a gateway reallocation affects an ongoing TCP stream. The upper plot in the right part of Figure 3-38 depicts the result from one of the ICMP experiments. At the beginning, when the gateway for STAB1 is s4, the average RTT fluctuates around 4 ms, as the end-to-end flow traverses two hops in the wireless BH (s1-s2-s4). After the gateway relocation, STA B1's flow crosses only one BH link (s1-s0) before reaching the tunnel interface to tenant B's home network, and the resulting RTT decreases to 2 ms. However, the gateway relocation process is not immediate, as the rule update and ARP spoofing performed by the SWAM controller require a certain amount of time for processing and packet transmissions. The whole relocation takes an average of 201.72 ms, the time measured between the moment in which the last valid ICMP packet reaches the tenant's home network via gateway node s4 and the time when the first valid ICMP packet reaches the tenant's home network through gateway node s0. The lower plot in the right part of Figure 3-38 shows the empirical CDF of the connection outage times resulting from the relocation process measured in all the experiments. Note that the relocation time depends on the performance and the delay of the control path between the nodes and controller. In our second experiment, an iperf with TCP traffic is launched towards tenant B home network, in order to determine the impact of the reallocation process on the end-to-end throughput of STA B1. The left part of Figure 3-39 depicts the evolution of STA B1's throughput during the course of two of these experiments. At the beginning of the experiment, a throughput of around 30 to 50 Mb/s can be observed. The fluctuations are a result of varying channel conditions and of TCP's congestion window that causes a non-steady data rate. At the 60 seconds mark, when the gateway reallocation occurs, the end-to-end flow is interrupted for a brief time, causing a drop in the throughput. The left part of Figure 3-39 captures two different cases that are representative of how the throughput evolves after the gateway reallocation process. In the first case (continuous line), TCP performs a fast recovery, quickly reaching similar throughput values as before the reallocation process. In the second case (dashed line), TCP performs a slow start procedure, effectively extending the time it takes for the client to reach the initial throughput levels. We conclude that, in both cases, ongoing flows using TCP recover from a gateway relocation in a reasonable time. Since relocation is not a common event, we prove that this is an effective mechanism to enable load-balancing while minimising impact on ongoing traffic.

Mobility: In order to determine how fast SWAM can react to a client that switches from one vap to another, two different events are analyzed: i) STA A1 moves from the vap on node s1 to the vap on node s2, and ii) STA B1 moves from the vap on node s1 to the vap on node s3. In order to analyze the two mobility experiments, a reachability test is performed. It consists in performing the handover process for each client separately, while they are pinging their home network every 1 ms. The overall time it takes to switch an vap is composed of two intervals: i) the client handover time, and ii) the time for SWAM to redirect traffic to the BH tunnel connecting to the target vap. The client handover time does not depend on SWAM and it is therefore not evaluated. Instead, we focus our evaluation on the time required by SWAM to update the BH tunnels.

The right part of Figure 3-39 shows the CDFs of the overall reallocation time measured for STAA1 and STAB1 across all experiment repetitions. The reallocation process proves to be fast, in average taking less than 30 ms and at most 50 ms, independent of the client. In average, STAA1 is reallocated slightly faster than STA B1, as it is only required to update the access bridge in the root node, and not in the tenant's home network. Overall, it can be stated that the reallocation times for both clients represent only a fraction of the handover time. This demonstrates that SWAM is a very agile architecture, capable of quickly adapting its network configuration when dealing with client mobility.

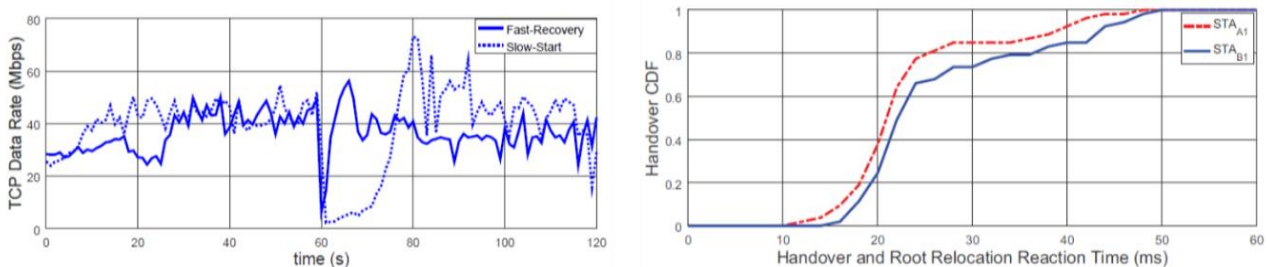


Figure 3-39. SWAM Mobility evaluation.

3.7.4 Packaging for the 5G OS

To understand how this function will interact with the 5G OS, we need to distinguish between the control function used to support service deployment, which was described in the previous section, and the service description itself. Thus, the Wi-Fi controller used to manage the Wi-Fi Small Cells is a control plane function that is part of the 5G OS itself, i.e. it is a Domain Controller using the 5G OS terminology. Instead, the access connectivity service, is a service described using a 5G OS service descriptor, which will be on-boarded into the 5G OS repository.

We sketch next how the service descriptor provided by this technical component would look like. A detailed definition of this service descriptor though, is within the scope of WP5.

SWAM-service descriptor:

- *Wireless Controller IP address // reachability information for the corresponding control function*
- *List of physical Wi-Fi APs where connectivity is requested*
- *Quota associated to the service // % of airtime*
- *Service template:*
 - o *SSID*
 - o *Security credentials*
- *List of physical Wi-Fi APs with wired network connection where a tunnel to the tenant's home network should be instantiated*
- *Home network tunnel template:*
 - o *Type: Transport VLAN, GRETAP*
 - o *For Transport VLAN: value*
 - o *For GRETAP: remote end point IP address*

The 5G OS would retrieve this service descriptor from the repository, complete it with the data corresponding to a particular service instantiation, and submit it to the control plane function, which would then communicate with the physical network devices.

4 PNFs and VNFs to support synchronisation services in converged FH/BH networks

4.1 Introduction

5G-PICTURE deliverable D4.1 [3] defines an architecture for synchronisation as a service in the context of heterogeneous networks. The vision is that multiple technology domains are going to feature on-path support for frequency and time distribution on a subset of their nodes. However, each domain will do so in accordance to its own physical and protocol limitations. The network, in turn, will track these individual capabilities and ensure that, in the end, timing distribution services can be successfully provided from time servers to end applications while meeting the tenant's service level requirements.

A dedicated control plane is envisioned for such a coordination of synchronisation functions. In essence, this control plane needs to acquire node information, configure the nodes in the network and gather real-time performance metrics. These are the functionalities that in deliverable D4.1 [3] were deemed as essential for a "synchronisation harmoniser" to be able to optimize and maintain the timing services requested by end applications. The acquisition of information, in particular, is expected to be provided by control functions embedded close-to or inside the timing-aware nodes pertaining to each technology domain. These functions will communicate to the harmoniser and, ultimately, allow it to acquire a global view of the timing network. Some of these control capabilities are further discussed in 5G-PICTURE deliverable D3.2 [5].

In addition to control plane aspects for synchronisation as a service, deliverable D4.1 [3] also covered synchronisation functions in key 5G-PICTURE technologies. More specifically, IEEE 802.11ad millimetre-wave (mmWave) mesh and IEEE 802.11 Sub-6 GHz networks, which are intended to support the IEEE 1588 Precision Time Protocol (PTP) [47]. Additionally, it considered over-the-air time synchronisation between timing division duplexing (TDD) remote radio units by relying on dedicated signalling between them and synchronisation over Flex-Ethernet (Flex-E). The timing support provided by these technologies is meant to support time-synchronised transmissions in the mobile radio access network (RAN), such as the ones for coordinated multipoint (CoMP) transmissions or TDD uplink-downlink interference coordination. For IEEE 802.11 in Sub-6 GHz, exceptionally, the work considers the usage of network-distributed timing references in the IEEE 802.11 transport domain itself, rather than the RAN. In this case, the goal is to enable time-oriented resource allocation for multi-tenancy in the transport network.

The present work describes the advances along the lines of work introduced in deliverable D4.1 section 5, that is, on synchronisation control and harmonisation capabilities, as well as timing support for 5G-PICTURE transport technologies. The organisation of the text is as follows: Section 4.2 comprehensively describes the adopted testbeds, with details regarding architecture, software, hardware resources and implementation. Section 4.3 then analyses concepts and results focused by each testbed, with independent subsections dedicated to each of the technical components from deliverable D4.1.

4.2 Technical Component 1: IEEE 1588 over IEEE 802.11ad

4.2.1 Summary Description

The main functionality offering is related to the detection and transportation of 1588 frames over mmWave devices with minimal delay, supporting timestamping.

4.2.2 Used programmable platforms and APIs

4.2.2.1 Overview

The work presented in section 5.3.1 of deliverable D4.1 [3] introduces MAC-level strategies for the transport of IEEE 1588 over IEEE 802.11ad. More specifically, it discusses an approach for identification of IEEE 1588 content in IEEE 802.11 frames such that the receiver knows when to timestamp the arrival of a PTP message without inspection of the 802.11 frame payload. Additionally, it analyses transmission settings that can be adopted to improve synchronisation performance, such as prioritisation in terms of quality-of-service (QoS), the strategy in terms of packet aggregation and acknowledgement policies. These are further analysed in this work based on experimental results.

As a reference for subsequent sections, Figure 4-1 illustrates a layer model of PTP operation in the network, particularly considering IEEE 802.11ad network interfaces. The architecture assumes that the IEEE 1588 PTP

implementation stack runs in the application layer, in the user space of an OS. This is the piece of software that effectively implements the PTP protocol, namely its data sets, state machines, algorithms and messages. The PTP messages generated by such an application traverse the OS network stack until they reach the network device driver that is responsible for pushing data for transmission by the appropriate network adapter. While crossing the network stack, these messages may either be encapsulated within UDP, IP and Ethernet, in case of UDP transport, or be framed directly in raw Ethernet frames. In any case, Ethernet frames are processed by the driver and then transferred to the network adapter. The MAC layer of the adapter, in turn, defines how these frames are sent over the link, that is, sets configurations that are supportive for timing transport. Finally, the PTP messages, now within the payload of MAC frames, are passed to the PHY for transmission in the wireless channel.

Once a transmission is deemed as complete, the departure timestamp taken at the PHY is sent up to the application through the same layers that it crossed originally. Completion, in this case, can either be right after the end of transmit frame or solely after an acknowledgement comes back from the recipient, in case acknowledgement is used. After that, the network card passes the timestamp to its driver, which passes it to the network stack such that the latter can deliver the timestamp to the application. Meanwhile, at the receiver, the incoming IEEE 802.11 frames are continuously parsed based on their headers and by doing so the receiver can identify when a frame has PTP content. Once found, the receiver takes an arrival timestamp and passes it to the driver along with the data. The driver, once again, delivers the timestamp to the network stack and the latter pushes the timestamp to the socket that is bound to the PTP application.

The testbed that is described in the sequel relies on an implementation following the architecture of Figure 4-1. The specific components implementing each layer are clarified in the sequel.

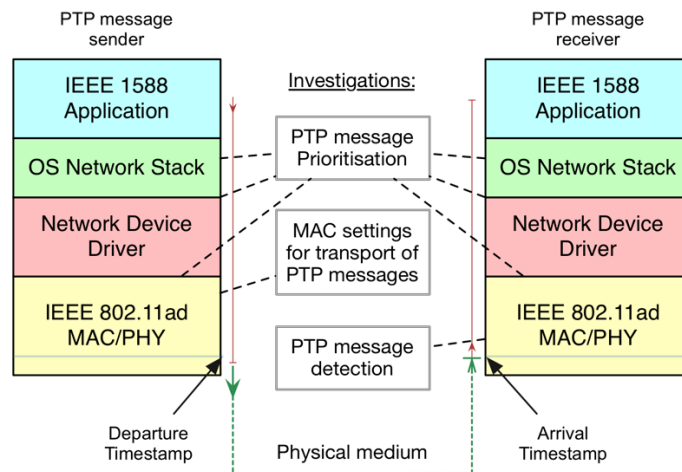


Figure 4-1. Layer model for IEEE 1588 operation over IEEE 802.11ad with summary of corresponding investigations.

4.2.2.2 Related technical components

The investigation on IEEE 1588 transport over IEEE 802.11ad consists in one of the components in deliverable D4.1 that addresses novel technologies for PTP transport. Given the context of synchronisation distribution over heterogeneous networks, this component represents one among multiple technology domains in the network and, hence, needs to be coordinated. Hence, this component relates to the Synchronisation Harmonizer component of deliverable D4.1 [3]. Besides, it also relates to synchronisation programmability interfaces described in deliverable D3.2 [5].

4.2.2.3 Resources: hardware and software components

The testbed covered in this section relies on the Blu Wireless Technology (BWT) Typhoon platform, which comprises two mmWave IEEE 802.11ad wireless modems connected via PCIe to a network processor unit (NPU), hereafter referred simply as the “host”. The platform is based on the RWM6050 chip from IDT Systems Inc., which includes patented silicon IP from BWT, referred as the “BH2”. The latter implements two independent IEEE 802.11ad modems, named “Hydra 0” and “Hydra 1”, with independent MAC and PHYs. These modems implement up to baseband processing, while relying on an external RF frontend, to (from) which the baseband signals are fed (received).

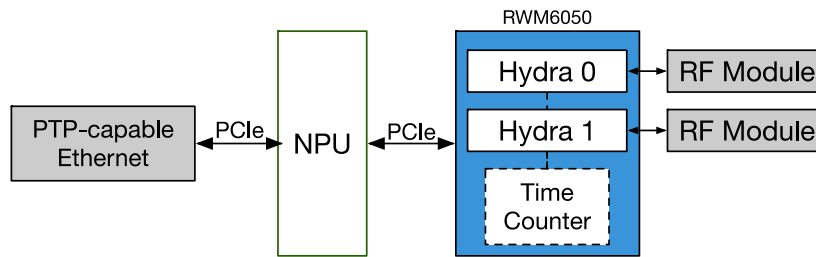


Figure 4-2: Blu Wireless Typhoon platform overview.

The diagram in Figure 4-2 summarises the high-level architecture of the Typhoon platform. Note that, in addition to the RWM6050 (with the BH2 IP), the host is also attached to a PTP-capable Ethernet interface. This allows interfacing to Ethernet-based PTP clocks in the network. The reader is referred to deliverable D3.1 [4] for further details on the Typhoon HW.

The BH2 PHY hardware is capable of taking nanosecond-accurate timestamps of incoming and egressing packets. In terms of the model in Figure 4-1, the Hydra modem of the BH2 (and hence the RWM6050) represents the bottom layer, i.e. the IEEE 802.11ad MAC and PHY. This layer communicates to the BH2 device driver, which runs in kernel space at a Linux host. Both the MAC and the BH2 driver provide special support for the transport of hardware-based packet timestamps. The driver delivers the timestamps into the Linux network stack when applicable and the latter, then, carries the timestamps up to the application.

Regarding software resources, two different PTP applications are adopted for the experiments. The first is the *ptp4l* open-source implementation of the *linuxptp* framework, which can operate as either PTP master or slave, that is, features PTP ordinary clock functionality [47]. This application can also operate in PTP boundary clock (BC) mode, where one network interface (PTP port) behaves as slave to an upstream node and the other interfaces behave as PTP slave to downstream nodes. Meanwhile, specifically for tests in PTP transparent clock (TC) mode, a self-developed TC application is exploited.

In the TC case, the layer model can be better represented by the diagram in Figure 4-3, which illustrates a transmission from PTP master to slave through a TC. In this scenario, the PTP master can be thought as a Typhoon, the TC as another and the slave as yet another Typhoon. In the master and in the slave, a single Hydra of the BH2 is used, whereas in the TC both Hydras are used as two independent PTP ports.

When the master application illustrated in Figure 4-3 transmits a PTP message, this message first arrives at the TC Typhoon via one of its IEEE 802.11ad modems (say Hydra 0) and is timestamped upon arrival. The TC application then forwards the message to the slave via the other IEEE 802.11ad modem (i.e. Hydra 1). The departure time towards the slave is made available to the TC application as the egress timestamp, and the application then works out the message's residence time (inside the TC) in order to perform its intended TC support. A similar layering model applies in the BC use case.

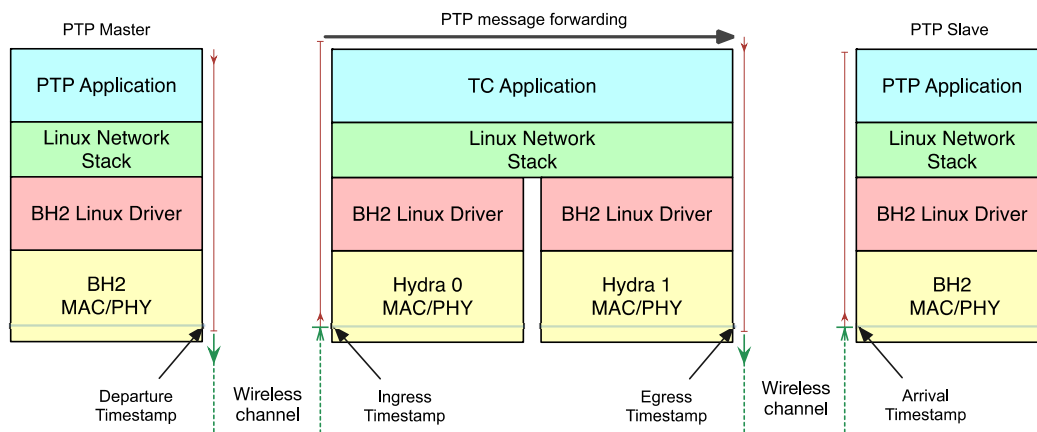


Figure 4-3: Transparent clock layer model.

In addition to *ptp4l* and the self-developed TC application, the *phc2sys* application from the *linuxptp* framework is used. The latter is capable of synchronising time counters corresponding to independent PTP-aware network interfaces attached to a Linux host, that is, the so-called PTP HW clocks (PHCs). In our case, it is used to synchronise kernel space time counters corresponding to the independent Hydra modems and the PTP-aware Ethernet card in the host. This usage is further detailed later. Moreover, the *iperf3* application is also adopted along the tests for generation of background (BG) UDP traffic, where BG in this case implies traffic other than PTP.

Lastly, standalone PTP grandmaster and slave clocks are used for some of the experiments. The specific grandmaster clock is the Meinberg M600, a device containing a high stability (oven controlled) crystal oscillator that interfaces to time sources such as GPS and IRIG. This device is used as the PTP master clock that is on top of the clock distribution hierarchy, namely the PTP grandmaster. Meanwhile, the standalone PTP slave that is adopted is the Meinberg SyncBox/PTPv2. Both the Meinberg M600 and the SyncBox provide pulse-per-second (PPS) outputs that can be used to evaluate the synchronisation accuracy. Table 4-1 summarises these and all aforementioned testbed resources.

4.2.2.4 Testbed Architecture

Several distinct topologies are explored and clarified along this section. The first is the one illustrated in Figure 4-4. The setup consists of two Typhoons, namely two hosts, each attached to its own BH2. The BH2s, in turn, are connected directly to each other by coaxial cables that transport baseband signals. That is, this setup does not involve up and down-conversion to (from) 60 GHz and neither does it include RF interfaces or wireless transmissions. It is used with the goal of running isolated tests that exclude effects from the RF frontend and the wireless channel.

The second topology is illustrated in Figure 4-5. Unlike the previous topology, this setup effectively employs RF interfaces and wireless transmissions in 60 GHz. These are particularly carried inside an anechoic chamber. Moreover, this setup consists in a point-to-multipoint topology. At host 1, a single Hydra of the Typhoon is used. Meanwhile, at the second host, both Hydras are used and connected to independent RF interfaces. Their transmissions to the station (STA) in host 1 are, therefore, time shared and the two stations in host 2 are spatially adjacent to each other.

Table 4-1: Summary of resources used for the IEEE 1588 over IEEE 802.11ad test setup.

Resource	Type	Role
BWT Typhoon	Hardware	Transport node with PTP-aware IEEE 802.11ad and Ethernet interfaces
BWT TC	Software	BWT-developed TC application
<i>ptp4l</i>	Software	Open-source PTP application
<i>phc2sys</i>	Software	Application that synchronises independent Linux PHCs
<i>iperf3</i>	Software	Application used to generate BG traffic
Meinberg M600	Hardware	Standalone time server used as PTP grandmaster and connected via Ethernet
Meinberg SyncBox/PTPv2	Hardware	Standalone PTP slave Ethernet-based device

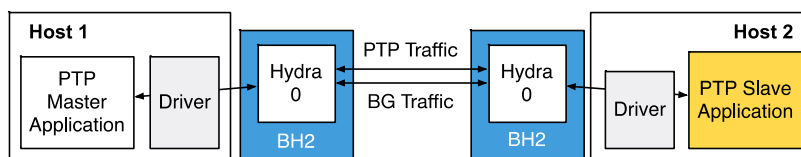


Figure 4-4. Cabled setup with direct baseband link between PTP master and slave hosts.

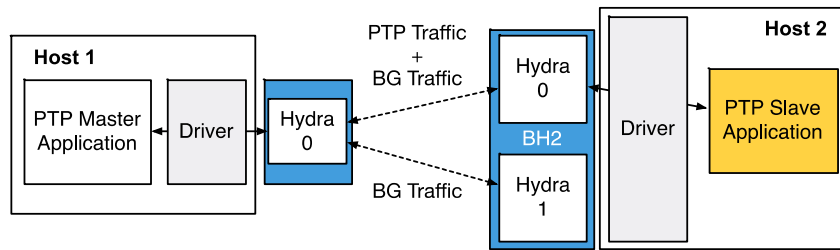


Figure 4-5. Point-to-multipoint wireless test setup.

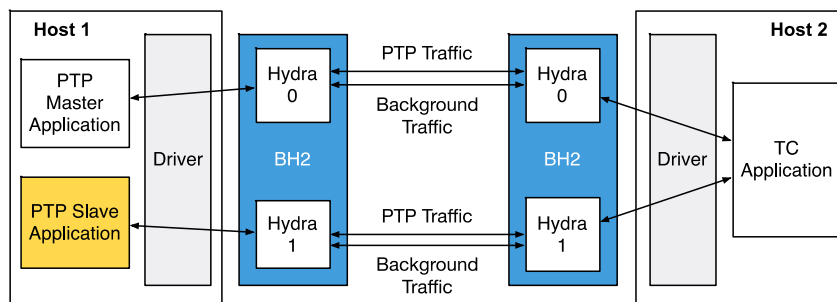


Figure 4-6. Transparent clock test setup using BH2s connected directly via coaxial cables.

Next, the setup used for TC tests is illustrated in Figure 4-6. It is again composed by two hosts, each with its own Typhoon. Host 1 launches two independent *ptp4l* instances, one as PTP master and the other as PTP slave, respectively bound to network interfaces corresponding to Hydra 0 and Hydra 1. In contrast, Host 2 launches the TC application and uses its two modems as the two independent ports of the TC. The communicating Hydras are connected directly to each other via coaxial cables.

Lastly, the setup that relies on standalone PTP clocks is illustrated in Figure 4-7. At first, a baseline performance is obtained by connecting the PTP grandmaster (Meinberg M600) directly to the Meinberg Syncbox/PTPv2 slave via ordinary 1000BASE-T Ethernet. The PPS output of the slave is plugged into an oscilloscope together with the PPS output of the M600. Then, the offset between the rising edges of these PPS signals is used to evaluate the time synchronisation performance.

With the baseline performance, the second step is to introduce two Typhoons operating as Transparent Clocks (TCs) in the path from master to slave, as illustrated in the bottom part of Figure 4-7. In this case, the messages traverse the Typhoons from Ethernet to mmWave and vice-versa. That is, a PTP message transmitted by the M600 grandmaster first arrives at Typhoon 1 through its PTP-capable Ethernet interface and, then, is forwarded to the next Typhoon via the mmWave interface. Similarly, at Typhoon 2, the incoming PTP message arrives via the mmWave interface and subsequently is forwarded by the TC application towards the Meinberg Syncbox/PTPv2 via the Ethernet interface. In the end, the time accuracy impact of these TCs is assessed by using the PPS signals from the standalone master and slave equipment, again by assessing their alignment and comparing to the baseline.

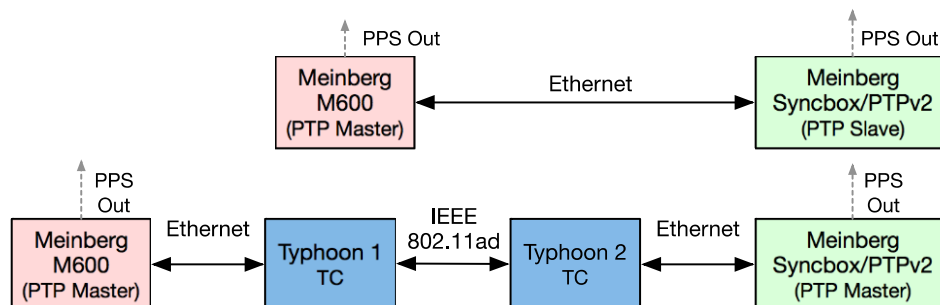


Figure 4-7. Setup for synchronisation performance measurements using standalone master and slave devices communicating through Typhoons in TC mode.

4.2.2.5 Configuration and Implementation details

In all tests with ptp4l, the PTP master clock is configured to send 128 Sync messages per second and 8 Announce messages per second, while the slave is configured to send 16 Delay_Req messages per second. Other than this, default *ptp4l* configurations are used, including the delay mechanism that is set to end-to-end (as opposed to peer-to-peer) [47]. Meanwhile, when using the M600 as master, i.e. in the setup of Figure 4-7, the master differs in the Sync rate only, which is configured to 16 packets per second.

PTP messages are encapsulated in both raw Ethernet frames and UDP datagrams along the experiments, depending on the test scenario. These will be clarified before the presentation of each experiment. Furthermore, background UDP traffic with gigabit bandwidth utilisation is streamed between the STAs during tests using the iperf3 application. In some tests, the maximum bandwidth allowed for this traffic is controlled and varied over time in order to assess the BG traffic impact on synchronisation.

With regard to IEEE 802.11ad specifics, it should be noted that contention-based channel access periods (CBAPs [48]) are adopted for all data transmissions. These are guarded by request-to-send (RTS) and clear-to-send (CTS) signalling exchanged between the STAs. That is, even though CBAP slots are used, an STA checks whether it can send or not to another STA before doing so by using the RTS/CTS exchange. Moreover, the link adopts auto-adapted modulation and coding scheme (MCS), which tunes to the appropriate choice at any given moment

A further important aspect along the tests is synchronisation of the time counters from independent network interfaces attached to the same host. This is necessary for TC (or BC) tests, specifically when relying on PTP ports on independent network devices. For example, the case of a TC with one port bound to an Ethernet card and the other port bound to the IEEE 802.11ad card, as in Figure 4-7. The motivation, in the BC case, is that synchronisation of the slave port (following an upstream master) must lead to synchronisation of the master ports, so that they can deliver the synchronised time downstream. This only holds if an external entity (or the PTP application itself) can synchronise the time scales of the independent ports. In the TC case, in turn, the motivation is that comparison of ingress and egress times taken at independent interfaces (refer to Figure 4-3) is reasonable only if the time scales of the involved interfaces are aligned. Any offset between them will lead to error parcels on the TC residence time computations.

Under the Linux architecture, this scenario arises when a BC or TC application relies on independent PTP Hardware Clocks (PHCs), where a PHC consists in an independent time count kept by the driver of a network interface. A PHC time count can either be following the time used by the timestamping unit (TSU) of the network device or it can be exactly equal to the TSU time. Correspondingly, interaction to the PHC of an interface can either lead to interaction to a data structure kept at the driver (one following HW time) or produce direct read/write transactions to the TSU of the device. The *phc2sys* application can be used in both cases. It synchronises multiple PHCs by taking one as the time master and other PHCs as slaves. The usage in our case is illustrated in Figure 4-8, where the PHC that assumes master role varies per setup.

In the test setup of Figure 4-6, the *phc2sys* application is used solely during initialisation of host 2 (the TC) and is not used at host 1. The rationale is that the time counts corresponding to the Hydras of the same BH2 are naturally synchronized, as they follow the same oscillator in hardware. Thus, the corresponding PHC time scales of the Hydras only differ by an initial offset that arises due to different times of driver loading, when the PHC time is reset. This initial error can be roughly corrected by *phc2sys* and, then, *phc2sys* can be turned off. Thereafter, the independent Hydra time counts are expected to remain aligned.

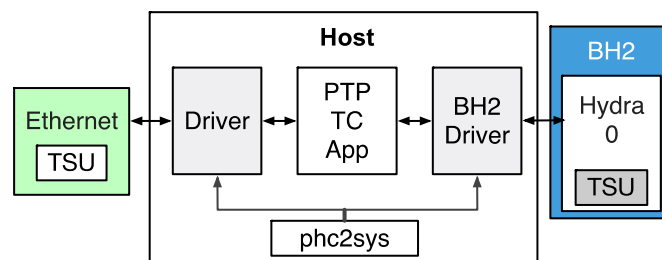


Figure 4-8: *phc2sys* usage for synchronisation between independent Linux time counters.

The same behaviour does not hold in the setup of Figure 4-7. Since the TC in this case operates from Ethernet to mmWave and vice-versa, it is now the Ethernet and BH2 time counts that need to be aligned. These are not synchronized, as they rely on different oscillators (on different boards). As a result, correction of time offset at a given moment does not imply that the time scales will remain aligned. Therefore, in this case, the phc2sys application needs to remain active during the entire operation of the TC, in which it continuously synchronises the Ethernet and BH2 timescales.

The problem with continuous operation of phc2sys is the performance loss, which arises due to the fact that phc2sys relies on software to synchronise hardware clocks. That is, it has to read the time at the master PHC and then apply it to the slave PHC in another (write) operation. Since these read and write operations are not atomic and there is jitter in both reading and writing, they introduce significant errors. This is acceptable when phc2sys is executed solely during initialisation, as the residual error left after phc2sys is stopped remains during the entire operation of the device, provided that the PHCs are synchronized. However, when operating continuously (when PHCs are not synchronized), the error is continuously changing and so is the error of a PTP slave downstream. The solution involves relying on hardware-based synchronisation of independent devices, which is typically implemented by connecting PPS signals between them. However, this infrastructure is out of scope and, instead, the degradation is accepted for the experiments that are presented in the following subsection.

4.2.3 Function Design/Implementation/Evaluation

4.2.3.1 Configuration/Simulation/Proof of concept details

This section presents experiments carried out with the testbed described in the previous section. The experiments aim at advancing discussions and propositions posed in Section 5.3.1 of deliverable D4.1. These are summarised in the layer model of Figure 4-1 and revisited in the sequel.

The first discussion is with regard to the IEEE 1588 identification strategy. That is, the approach of sending PTP messages within ordinary IEEE 802.11 data frames with a PTP-marking group receiver address (RA). One point of discussion concerns unicast PTP messages. More specifically, whether the approach is valid to ensure that a single destination receives a message whose RA is a group address, potentially matched by nearby stations. The motivation for believing it is safe to adopt a group address is that mmWave transmissions are highly directional and ultimately directed from point to point. Furthermore, specific stations are scheduled to a given access time slot (even in CBAP) and the transmissions are guarded by RTS/CTS exchange. In the end, this leads to being very unlikely that a PTP transmission to one station would leak to another. This is verified next by using the point-to-multipoint arrangement of Figure 4-5.

A second discussion refers to the acknowledgement policy. There is one potential motivation for using NoAck policy when transmitting 802.11 frames carrying PTP event (timestamped) messages, as discussed in deliverable D4.1. It comes from the choice of IEEE 1588 identification using a magic group address RA. Because this address is a group address, if acknowledgement is enabled, multiple stations could potentially acknowledge the same message. Nevertheless, it was argued just above that this is not expected, since transmissions are highly directional, so this motivation can be neglected.

Instead, the main motivation for transmissions using NoAck policy comes from current BH2 software constraints, regarding how group addresses messages can trigger the transmission of acknowledgement, which is out of scope. For the BH2, PTP event-bearing frames are indeed preferably transmitted using NoAck policy. The impact of using NoAck versus using acknowledgement is evaluated in the sequel to understand whether there is any drawback of not relying on acknowledgements for PTP frames.

An expected drawback of using NoAck policy is that it leads to higher probability of losing PTP messages in the link, due to absence of retransmissions. For that, we leverage on the robustness of PTP instead, since a PTP slave can recover from sporadic message losses. However, to reduce the probability of message loss, the PTP event-bearing frames are sent using MCS 1 [48], that is, with maximum reliability. This is suitable since PTP event messages are 44 bytes long and sent at a low rate (kbps), such that they do not impose a significant drawback in terms of air time and link utilisation.

One noteworthy perspective is that the approach of using NoAck policy inherently achieves a low-level PTP earliest arrival packet filter [49]. That is, it uses solely the PTP message exchanges corresponding to transmissions that are successful in the first transmission attempt, with no need for re-transmissions. This implies PTP delay-request response exchanges [47] whose overall duration are shorter. On the other hand, packet loss can lead to long intervals between slave servo updates and potentially loss of messages that

maintain the best master clock algorithm (BMCA). The trade-offs between the two approaches are not obvious and will be observed in the experimental section.

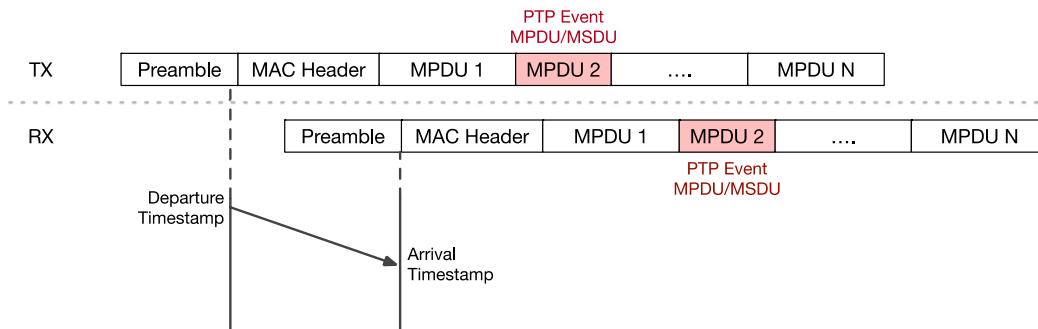


Figure 4-9: Timestamping of PTP event MPDU/MSDU on aggregated 802.11 frames.

A last aspect of using the group RA strategy for PTP event identification refers to aggregation. If the PTP message was aggregated to other packets in an aggregated MAC service data unit (A-MSDU) and (or) aggregated MAC protocol data unit (A-MPDU) [48], the offset of the PTP MSDU and the corresponding MPDU with respect to the frame start would not be a problem as far as PTP operation is concerned. This is because the goal in PTP is to have timestamps taken at two clocks in the network between which there is a constant and traceable delay, to allow clock comparisons. Packets are timestamped at the PHY layer just as an artifact for achieving timestamps at both sides of a link with a reliable delay in between. Hence, it makes no appreciable difference if the A-MPDU start is timestamped rather than the precise moment that the PTP MSDU goes on air.

The problem with aggregation is not the offset of the PTP message in the A-MPDU, since the offset is the same on departure and arrival, as illustrated in Figure 4-9. Instead, it is the fact that an A-MPDU has a single RA in its header, so that once the PTP-marking RA is set it cannot by itself indicate which MSDU in the frame consists in the PTP event message. Hence, the receiver cannot know which MSDU should be associated to the arrival timestamp when sending the timestamp (and the packet data) up to the driver. The adopted approach, then, is to disable aggregation on frames carrying PTP event messages, namely to send them isolated on non-aggregated MSDU/MPDUs.

Finally, the experiments presented in the sequel also evaluate the effects of applying QoS prioritisation to PTP messages. The IEEE 802.11ad link established by the BH2 is configured to send PTP packets with maximum QoS priority. This ensures lower latencies over the IEEE 802.11ad link alone. To complement this, when using UDP transport, prioritisation is also assigned at the OS network stack by using the type-of-service (ToS) field of the IP packet. The latter reduces the latency that the packet is subject to when crossing the network stack from the application to the device driver. The Linux commands for assigning ToS priority are presented below in Listing 1. Note both UDP ports used by PTP are prioritised.

Listing 1

```
sudo iptables -A PREROUTING -t mangle -p udp --dport 319 -j TOS --set-tos Minimize-Delay
sudo iptables -A PREROUTING -t mangle -p udp --dport 320 -j TOS --set-tos Minimize-Delay
sudo iptables -A OUTPUT -t mangle -p udp --dport 319 -j TOS --set-tos Minimize-Delay
sudo iptables -A OUTPUT -t mangle -p udp --dport 320 -j TOS --set-tos Minimize-Delay
```

4.2.3.2 Results Relation to KPIs

Deliverable D4.1 [3] discusses two categories of timing characterisation and KPIs, ones focused on radio access performance and ones focused at the timing transport alone. Here, we focus on timing transport metrics directly. The main adopted KPI is the slave “offset from master” [47], namely the time offset between the PTP slave and the PTP master. This time offset is observed with two main expectations: that its maximum value over long observation intervals is small enough and that it does not fluctuate rapidly. Hence, the two main parameters observed next are the maximum time offset and the root-mean-square (RMS) time offset. The maximum absolute time offset is interchangeably named $\max|TE|$ in the sequel.

It should be noted that two types of time offset measurements are used in the experiments. When relying on ptp4l, the time offset measurements are solely based on estimations taken at the slave application (at SW level). As discussed in the context of Figure 4-8, the offset seen by the application does not translate directly into the offset between the HW (TSU) time counts. This is because there is an abstracted time count in the driver (the PHC), which follows the TSU time, but is not equal to TSU time and also can be synchronised independently, while the TSU time remains free-running. This means this evaluation cannot be deemed as the HW synchronisation accuracy, but solely as a good indication of it. Meanwhile, the second measurement approach is the one based on PPS signals, specifically used in the setup of Figure 4-7. This, in contrast, is based on hardware time counts, so it reflects the HW accuracy directly.

An additional relevant KPI refers to the latency that PTP messages are subject to from transmission by the application to effective departure from the network adapter. This latency is particularly measured by the timestamp fetching latency, which consists in the interval from the time the timestamped messages is pushed by the application for transmission to the time the departure timestamp arrives back at the application. This interval is illustrated in Figure 4-10. Note that it involves crossing all layers from application to transmission and backwards, the latter in the departure timestamp’s way back to the application. On the timestamp’s way up, the MAC first delivers it to the driver while acknowledging the transmission was complete. The driver then sends an acknowledgement to the network stack indicating the transmission was complete and sends the timestamp along. Ultimately, the network stack delivers to the application. We evaluate this complete latency under the presence and absence of PTP prioritisation.

In practice, there are many implications imposed by latency on PTP. One problem is that slave clocks rely on timeouts for reception of Announce messages in order to keep their state machine in the so-called “slave” state [3]. An excessive latency realisation at some point could disturb this and other state machines. Another problem is that latency causes time offset estimations to be outdated by the time they are applied as corrections. This is especially accentuated in two-step PTP mode [47], since the departure timestamp needs to be fetched before a follow-up message can be sent. This means the complete Sync and delay request-response exchange can be prone to significant latencies, such that the time offset computation would be outdated. This is loosely observed by assessing whether the time offset performance improves when using prioritisation.

A last metric of concern is the PTP event loss rate. When PTP event messages are transported using NoAck policy, eventual losses are expected. These can impact the slave’s synchronisation performance, since a loss event leads to longer intervals without clock correction. Hence, it is also observed next.

4.2.3.3 Experimentation/Evaluation results

The first experiment contrasts the approach using normal acknowledgement (Ack) for the PTP event-bearing frames to the strategy in which these frames are sent with NoAck policy. The experiment is carried on the point-to-multipoint wireless setup of Figure 4-5, where Host 1 hosts the PBSS control point (PCP) and Host 2 hosts two non-PCP STAs. Bidirectional UDP BG traffic runs between PCP and STA 1, as well as PCP and STA 2. At the same time, a unicast PTP stream is set between the PCP (PTP master) and STA 1 (PTP slave), particularly using layer-2 (i.e. raw Ethernet) transport. The slave was captured during roughly one hour, over which the maximum throughput allowed for the background traffic was gradually increased from 100 Mbps to 1000 Mb/s each way.

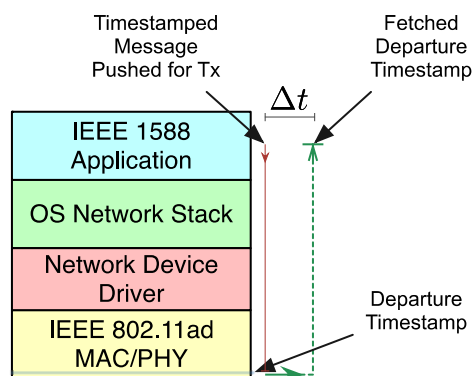


Figure 4-10: Timestamp fetching latency.

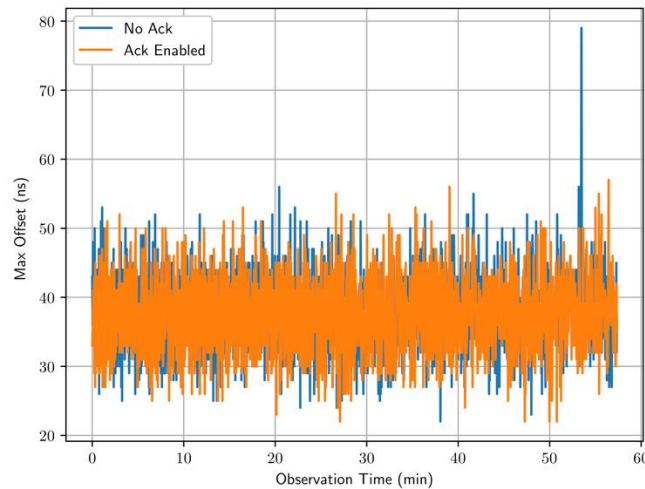


Figure 4-11: Maximum time offset under No Ack policy and with normal Ack.

The moving $\max|TE|$ estimation at the slave is shown in Figure 4-11. Note that both approaches present similar performance during the initial part of the experiment, while the BG traffic throughput is still relatively low. In particular, the $\max|TE|$ remains below roughly 55 ns with both approaches. However, as the BG throughput cap becomes higher, the PTP stream sent under NoAck policy starts to suffer from more frequent packet losses due to contention in the access. Furthermore, as the *iperf* throughput cap increases, this application fills the network stack buffer faster relative to the rate that the IEEE 802.11ad link can transmit and, as a result, PTP messages are not able to enter the buffer in a timely manner. This sporadically causes long delays to PTP messages and eventually leads to the loss of locked state at the slave, which impacts its synchronisation performance. Hence, performance degradation comes as a result of not throttling BG traffic fairly and also, under No Ack policy, due to PTP event losses.

In quantitative terms, both approaches on average presented a RMS time offset of approximately 12.6 ns. However, during the entire experiment, the $\max|TE|$ achieved with Ack enabled was 57 ns and under NoAck some error peaks were observed such that its $\max|TE|$ was 79 ns. In conclusion, the approach adopting normal acknowledgement showed superior and especially more reliable performance.

Next, we investigate the impact of prioritisation of PTP messages using the point-to-point cabled setup from Figure 4-4. Again, bidirectional BG traffic runs between the two hosts and its throughput cap is gradually increased, now from 1000 Mb/s up to 2000 Mb/s each way. This time, however, PTP is transported over UDP/IPv4 and, at the network stack, the PTP-bearing IP packets are assigned low-latency ToS, using the approach presented in Listing 1. At the IEEE 802.11ad link, in turn, the PTP-bearing frames are assigned the highest-priority QoS level. Meanwhile, NoAck policy is adopted along the experiment, since, as mentioned in Section 4.2.3.1, this is currently the preferable choice for the BH2.

The moving maximum time offset at the slave is presented in Figure 4-12. Due to the stability of the oscillators and the operation of the servo, the benefit of using prioritisation (QoS and ToS) was subtle. The histogram of the moving RMS slave time offset in Figure 4-13 reveals the slight difference, as the distribution with disabled prioritisation is less concentrated around the average. In particular, the distribution obtained with QoS and ToS enabled shows an RMS time offset varying over time with an SDEV of 1.017 ns, in contrast to 1.054 ns with prioritisation disabled. Also, the RMS offset was on average 14.87 ns with prioritisation and 15.05 ns without it. Importantly, note that during the hour-long experiment, the maximum time offset remained below 53 ns with prioritisation and 58 ns in the other case, which loosely suggests the ability to meet telecom-grade $\max|TE|$ targets that were summarised in deliverable D4.1 [3].

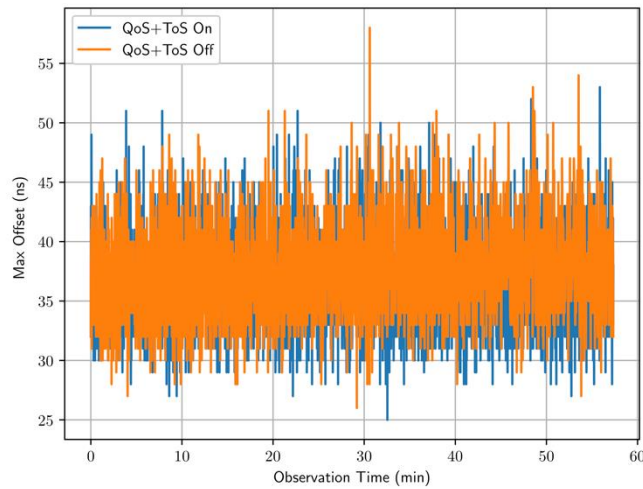


Figure 4-12: Maximum time offset with and without ToS and QoS prioritisation on cabled point-to-point setup.

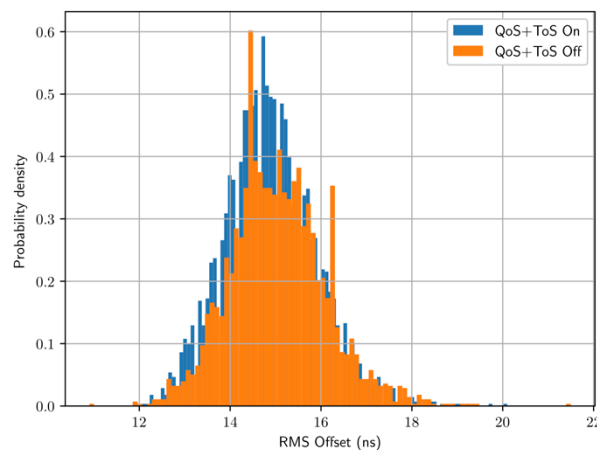


Figure 4-13: Histogram of the slave RMS time offset with and without prioritisation.

It is also worth to point out the reason why the average RMS offset is higher in this test (14.87 ns) when compared to the previous wireless setup (12.6 ns). It is not because of the physical medium (whether wireless or cabled). Instead, it is due to the adopted PTP transport layer. The present test adopts UDP transport, which takes longer to be processed. The extra latency leads to slightly inferior performance.

A subsequent experiment is set to exercise the performance of the platform as an IEEE 802.11ad TC. The experiment was carried on the cabled setup from Figure 4-6 and captured in the course of 24 hours. The resulting RMS and maximum time offset measurements are shown in Figure 4-14. Note that, throughout the experiment, the maximum time offset seen by the PTP slave was of 88 ns. While noting this includes both TC and slave error contributions, this, again, loosely suggests the ability to meet Class A and (or) B telecom max|TE| targets.

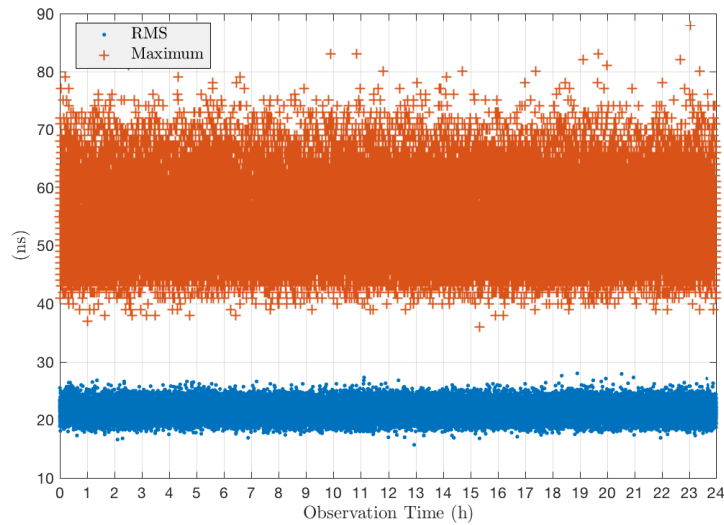


Figure 4-14: Moving RMS and maximum time offset estimations taken at the PTP slave with PTP transport through TC in the course of 24 hours.

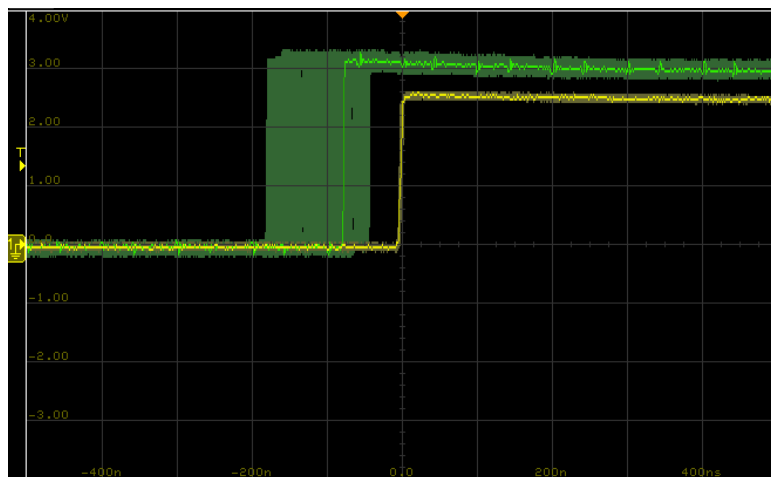


Figure 4-15: PPS rising edges generated by the PTP Slave (SyncBox/PTPv2) and Master (M600) when directly connected, captured over ~16min with infinite persistence.

Next, we exploit the setup that relies on standalone master and slave devices, in Figure 4-7. As explained in Section 4.2.2.4, first we seek a baseline performance achieved with a direct Ethernet link between the master and the slave. This is the result shown in Figure 4-15, which consists in 1000 consecutive trigger captures taken at the oscilloscope, while taking the PPS output from the PTP master as the trigger source (yellow waveform). Since PPS by definition ticks once per second, 1000 rising edge samples are taken in the course of approximately 16 minutes. The result in the figure is all of these captures overlapped with persistence. Note that, each horizontal axis division of the display represents 100 ns, so in this case the offset of the slave PPS rising edges (green waveform) relative to the master oscillates from around -200 ns to -50 ns. This offset changes over time, so it is only indicative of achievable performance, but does not strictly represent the actual slave accuracy specifications.

Next, we proceed to the setup in the bottom part of Figure 4-7. Now, instead of a direct link, the master and slave clocks communicate through two intermediate Typhoons that are running the BWT-developed TC application. The result is shown in Figure 4-16. Note that now the horizontal axis divisions represent 500 ns. Hence, the offset between the slave PPS (again in green) with respect to the master PPS (in yellow) in this particular capture oscillates from around -1.5 μ s to 1.5 μ s. As explained in Section 4.2.2.5, what is limiting the performance in this setup is mainly the approach adopted to synchronise the independent time scales of the Ethernet card and the BH2 in the Typhoon, which is being carried in software via the phc2sys. Hence, this

result does not reflect the actual performance capabilities of both the Ethernet card and the BH2. Nevertheless, it does validate the functionality of the TC application, as it is successfully operating in conjunction to COTS PTP equipment. It also loosely indicates that sub-microsecond performance is likely achievable with appropriate hardware-based synchronisation between the independent Ethernet/BH2 time scales.

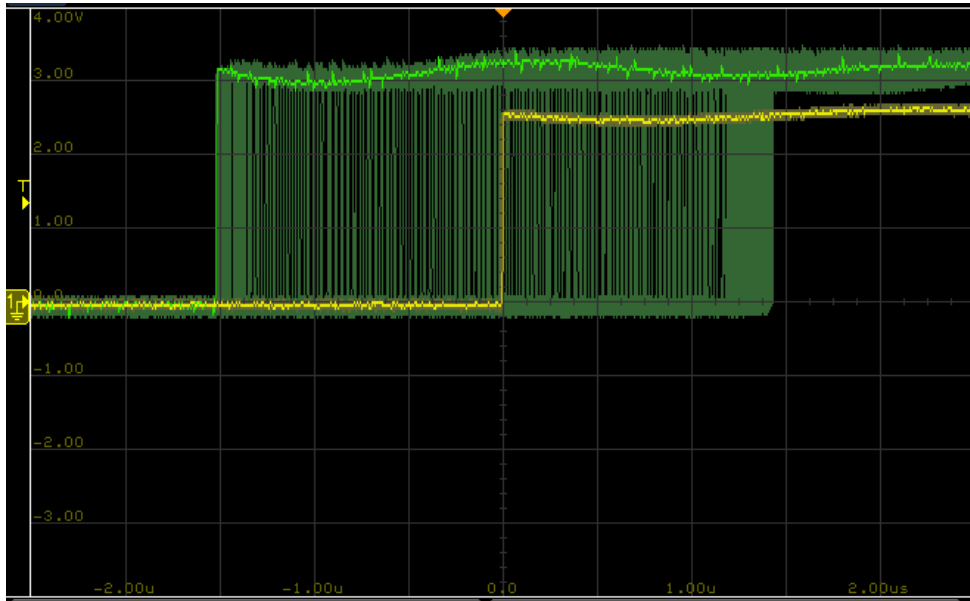


Figure 4-16: PPS rising edges generated by the PTP Slave (SyncBox/PTPv2) and Master (M600) when connected through two Typhoon TCs, captured over ~16min with infinite persistence.

4.2.4 Packaging for the 5G OS

The control plane, implemented by 5G OS, could be responsible for enabling PTP synchronisation support on a given node. To do so, 5G OS would have some metadata (the so-called descriptor), which would indicate that node X has HW timestamping capabilities and support PTP profile Y or PTP clock mode Z (say a transparent clock mode). 5G OS would then enable and configure the PTP support in the given node, by issuing e.g. an enable request. The OS would also be able to collect this knowledge (in descriptors) on-the-fly, for example by probing sync capabilities in the network via IEEE 1588 management messages, or via the synchronisation harmoniser architecture described below.

The synchronisation harmoniser is an intermediate piece of orchestration software to handle synchronisation configuration of nodes on behalf of 5G OS. A control function is needed on the node to be able to respond to harmoniser requests. A NETCONF/YANG interface permits the transfer of local synchronisation capabilities to the 5G OS, allowing it to form corresponding descriptors. In the case of the BWT Typhoon nodes the function chain is:

5G OS (running on a VM) -> Sync Harmoniser (running on a VM) -> network -> Sync Controller (running in the Typhoon) -> PTP TC app (running in the Typhoon).

A typical action sequence is:

- The 5G OS asks for synchronisation for a tenant X.
- The request is handled by the Synchronisation Harmoniser, which has direct communication to local Synchronisation Controllers (co-located with devices). The harmoniser then sends a request to a BWT Typhoon.
- A synchronisation controller (say a daemon) is running in the Typhoon and receives the request. This is translated into a management mechanism to launch or configure a PTP application. For example, to launch the TC application bound to given Ethernet/802.11ad ports and with that enable PTP TC support.

The descriptor would indicate capabilities that relate to the user space PTP applications that we can run in the BH2. For example, the descriptor could indicate (assuming something like a JSON format):

```
{
  "sync": {
    "ieee1588": {
      "modes": [TC, BC, OC],
      "profiles": [default],
      "ports": [enp4s0, enp4s1, wlp8s0f0, wlp8s0f1]
    }
  }
}
```

which indicates the supported PTP modes, the PTP profiles and the hypothetical network ports through which we could send PTP traffic.

And then we could indicate many other properties in the descriptor as we see fit to aid with the sync harmoniser. For example, the node's contribution to time error, whether it supports IEEE 1588 management messages, maximum message rates, syntonisation functionalities and so on.

4.3 Technical Component 4: Synchronisation harmonizer

4.3.1 Summary Description

New services envisioned in 5G networks will require a slightly different synchronisation architecture for synchronizing different entities in the network. The standard "tree" topology for synchronisation is no longer optimal for some of the expected new services enabled in 5G networks. Algorithms like best master clock algorithm (BMCA) would probably satisfy the requirements of most of the services, but there are also services this algorithm would not be able to support optimally. One such service is localisation. In order to perform a time of arrival (ToA) or time difference of arrival (TDoA) localisation, the anchor nodes, i.e. access nodes, must be synchronised with a precision in the area of a few nanoseconds. In this scenario, a precise synchronisation of the access nodes with the grand master clock (GMC) is not of interest. More important is the precision of the mutual time synchronisation between the neighbouring access nodes, which provide the localisation service.

Having new services with significantly diverse synchronisation requirements would bring additional complexity to the transport nodes, whether they are TCs or Boundary Clocks (BCs). These nodes would require more intelligence and the ability to support different algorithms for different requirements. Additionally, more knowledge about the topology of the network should be pre-stored in each of the nodes, or being communicated among them. Otherwise, the configuration of the network would be slower and not optimal.

To cope with the new synchronisation requirements, the main intelligence must be shifted towards the control plane, instead of residing in the nodes. The synchronisation harmonizer, such as the one presented in [50], takes the responsibility for offering and providing different synchronisation services, topologies architectures and capabilities.

One of the main objectives of the synchronisation harmonizer is to collect/obtain the synchronisation capabilities of the nodes in the network. It is supposed to have an overview on every synchronisation domain within the network. In addition, the synchronisation harmonizer should be aware of the established timing paths used for synchronisation, and whether the underlying technologies enable these paths. Since a mesh topology of the network is expected, the available synchronisation paths can be manifold. The synchronisation harmonizer would need to obtain information about the physical location of some of the access/transport nodes, in order to obtain synchronisation for location-based services [50].

The synchronisation harmonizer would have the role of deciding the optimal paths/configuration for fulfilling the synchronisation requirements for different services/applications. In this sense, different algorithms are being developed for fulfilling the envisioned requirements. The main role of these algorithms would be to select the optimal primary reference clock (PRC) - or multiple PRCs - to be used, as well as the optimal paths for distribution of the synchronisation, depending on the desired requirements.

IEEE 1588 [47] would be primarily used for synchronisation, leaving the role for the synchronisation harmonizer to run additional algorithms for aligning the synchronisation requirements with the synchronisation capabilities of the network.

4.3.2 Used programmable platforms and APIs (WP3)

The synchronisation harmoniser should be able to communicate with most of the platforms that support IEEE 1588 management interface. They were presented in deliverable D3.1 [4] and some of the developments have been recently incorporated to deliverable D3.2. The IEEE 1588 management interface [51] should be leveraged by the synchronisation harmonizer to configure the nodes belonging to different technology domains.

A NETCONF/YANG management interface [36] can be also used to configure the individual nodes, when the required functions are not supported by the IEEE 1588 management protocol. A local configuration API can be used for exchanging the needed data with the synchronisation harmonizer.

The synchronisation harmonizer provides an interface toward the network, responsible for obtaining node attributes needed for synchronisation. Using this interface, all of the necessary information for synchronisation capabilities of the nodes should be communicated/collected by the synchronisation harmonizer.

The synchronisation harmonizer offers an API, responsible for processing different request for synchronisation. This is a high-level interface that introduces an abstraction of the synchronisation service.

4.3.3 Function Design/Implementation/Evaluation

The synchronisation harmonizer is not a hard real-time control function. This fact allows implementation of this function in SW on a general-purpose computer or a virtual machine. A Linux-like OS should be utilised to exploit the already available network/synchronisation tools. Under Linux, a PTP management client (pmc) is available and can be used for collecting information from PTP-capable nodes. The machine running pmc does not need to support PTP itself.

The synchronisation harmoniser is able to accept requests from different applications. For this purpose, a separate API is being developed. In case the required requests cannot be fulfilled, the synchronisation harmonizer should inform the application about the offered synchronisation services. If multiple options are available, a negotiation should be conducted.

The detailed block diagram of the synchronisation harmonizer function is shown in Figure 4-17. It follows the description as in deliverable D4.1 [3] but, in this case, all of the necessary blocks are separately shown. The database, as introduced in D4.1, consists of minimum 3 datasets. The first one contains the available TCs and BCs, as well as the topology of the synchronisation network. The second dataset contains the synchronisation paths, whose topology is not exactly known, but it can be utilised. Finally, the third dataset contains the synchronisation paths that were already established by the synchronisation harmonizer.

Few different subroutines are available as well. A separate subroutine is used to collect synchronisation network topology data as well as data regarding available TCs and BCs and their capabilities. The main control subroutine is responsible for accepting requests from users/tenants regarding synchronisation services. Depending on the requests for a specific synchronisation service, a set of functions shown in Figure 4-17 (in the Set block) can be started in order to find a synchronisation topology as well as to optimise different parameters of the requested synchronisation service. Depending on the available use cases, the functions contained in this set can be updated, or new functions can be added. The communication towards the user/tenant is performed using the northbound interface and the configuration of the TCs and BCs is performed using the southbound interface. Depending on the discovery of topology change by adding/removing TCs and BCs, the main control subroutine can rerun the optimisation and topology function to obtain better synchronisation performance.

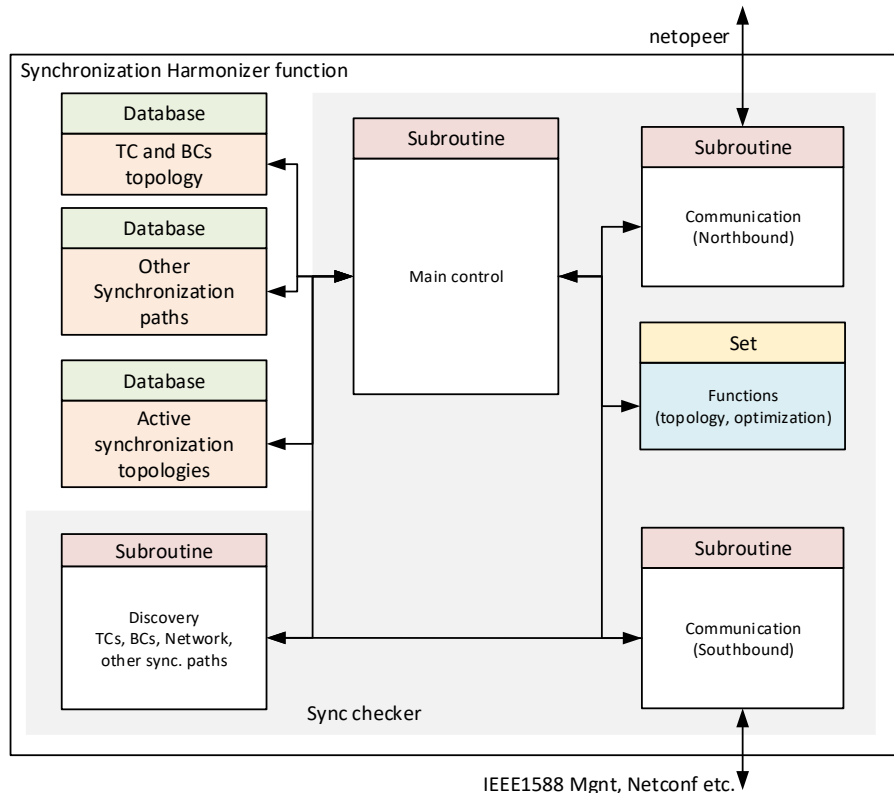


Figure 4-17: Detailed block diagram of the synchronisation harmonizer function.

The implementation of the synchronisation harmonizer is performed using standard available programming languages (e.g. C/C++) on a Linux-based platform. Nevertheless, since this function is not hard real-time, Python represents the best option. For the needed database for storing the node synchronisation capabilities, many open source databases are available under Linux-like operating system. At this stage, a database stored in sqlite3 will be used, being the support for additional databases straightforward.

To evaluate the developed algorithms, different scenarios for different synchronisation architectures would be considered. The algorithms for optimising the synchronisation topology will be implemented and their evaluation will be first performed in simulation. The main challenge is to confirm that the algorithms implemented in the synchronisation harmonizer lead to better synchronisation parameters at the slave nodes than that of the BMCA. To perform this evaluation, the models of the TCs/BCs should be developed, which are given in G.8271.1. Different scenarios and topologies will be provided for simulation. It should be verified that the developed algorithms perform better than BMCA or, at least, achieve similar performance. This is very dependent on the network topology and it should be taken into account. The simulations can be performed in different simulation environments. One simulator that can be used is NS-3. MATLAB can also be used, but this would involve more effort in order to develop a network model.

Results on the evaluation of the synchronisation harmonizer will be captured in deliverable D4.3, when all of the platforms' synchronisation capabilities and interfaces developed under WP3 are available. This would provide a quantitative insight of the performances that can be achieved. At this moment only functional testing can be performed but no real quantitative data can be presented.

4.3.4 Packaging for the 5G-OS

Not applicable.

5 5G-PICTURE integration plans

This document has described so far each of the functions developed in WP4 in isolation. However, within the broader context of 5G-PICTURE, the functions that are developed in WP4 bear a relation with the programmable platforms developed in WP3, and with the 5G OS developed in WP5. In addition, the functions that have been developed and reported in this deliverable, are the building blocks that will be used in the last year of the project to demonstrate larger integrated functions within the context of WP4 and WP6.

Hence, in this section we put the WP4 functions in perspective, and discuss: i) the relation between each WP4 function, the WP3 platform, and the WP5 5G OS, ii) the approach that 5G-PICTURE will take to on-board functions into the WP5 5G OS, and iii) an initial sketch of the integration plans between WP4 functions for D4.3 and WP6.

5.1 WP4 functions in the context of WP3 and WP5

Table 5-1 describes the relation between each WP4 function reported in this deliverable, the WP3 platform where this function needs to be executed, and the way this function can be integrated with the 5G OS developed in WP5.

Looking at the WP3 platforms (left column), we can distinguish two main categories. First, we have WP4 functions that are developed in software and executed over a generic x86 compute platform (not specific to WP3), marked in purple. Second, we observe a set of WP4 functions that target a specific programmable/configurable platform developed in WP3, marked in green.

Regarding at the integration between WP4 functions and the 5GOS in WP5 (right column), we see three main categories. First, we have WP4 functions that do not need to be integrated with the 5G OS, marked in red, since these simply provide some new non-programmable functionality. Second, we have a group of WP4 functions that are control plane functions, marked in purple. These functions have to be integrated as components of the 5G OS, for example they could be a specific Domain Controller [52]. Finally, we have another type of WP4 functions, marked in orange, which can be packaged and on-boarded onto the 5GOS catalogue. The 5G OS will then be able to deploy the on-boarded functions dynamically over the appropriate WP3 platforms, in order to compose an end-to-end service. In the next subsection we provide more details on the different on-boarding strategies considered in 5G-PICTURE.

Table 5-1: Relation between each WP4 functions, the WP3 platform where this function needs to be executed, and the way this function can be integrated with the 5G OS developed in WP5.

WP3 Platform	WP4 function	Integration in WP5 5G OS
RAN FUNCTIONS		
AIR platform – D3.2 section	Tech.Comp.#1: Optimal Massive MIMO functional split	Descriptor describing Massive MIMO antenna configuration
Generic x86 compute platform	Tech.Comp.#2: Software based RAN functions in OAI	VNF in VM/container plus associated descriptor
Generic x86 compute platform	Tech.Comp.#3: Control function to select optimal functional split in RAN	Control plane function belonging to 5G OS
Generic x86 compute platform or FPGA Xilinx Zynq - D3.2	Tech.Comp.#4: RAN as a VNF for custom PDCP split and heterogeneous wireless DUs	VNF in VM/container/FPGA plus associated descriptor API for bootstrapping the functions
I2CAT Small Cell platform – D3.2	Tech.Comp.#5: Sub6 wireless for fronthaul-like RAN splits	No interaction with 5G OS (it is a physical technology)
TRANSPORT FUNCTIONS		
Generic x86 compute platform	Tech.Comp.#1: TSON controller	Control plane function belonging to 5G OS (domain controller)

Huawei routing function – D3.2	Tech.Comp.#2: Ethernet channel isolation through Flex-E	No interaction with 5GOS (it is a physical technology)
Huawei routing function – D3.2	Tech.Comp.#3: Low latency cross-connect through X-Ethernet	No interaction with 5G OS (it is a physical technology)
Generic x86 compute platform	Tech.Comp.#4: Segment routing	Control plane function belonging to 5G OS (domain controller)
Open Packet Processor – D3.2	Tech.Comp.#5: Mobility function for railways scenarios	P4 based function on-boarded into 5G OS
I2CAT Small Cell platform – D3.2	Tech.Comp.#6: On demand wireless slice on joint access/backhaul small cells	Controller is a control function in 5G OS (domain controller) On demand wireless slice, onboarded as descriptor wrapping API call
SYNCHRONISATION FUNCTIONS		
BWT Typhoon platform – D3.2	Tech.Comp.#1: 1588 over IEEE 802.11ad	No interaction with 5G OS (it is a physical technology)
Generic x86 compute platform	Tech.Comp.#4: Synchronisation Harmonizer	Control plane function belonging to 5G OS (domain controller)

5.2 The 5G-PICTURE approach to P/VNF on-boarding to 5G OS

In this section we discuss the approach that 5G-PICTURE will take to onboard WP4 functions onto the WP5 5G OS; namely the functions marked in orange in the right column of Table 5-1.

Broadly, we can classify the taxonomy of “onboardable” functions considered in 5G-PICTURE in three groups:

- *Group 1:* Functions packaged in a Virtual Machine (VM) or container, executed over a traditional x86 compute platform. See as example the RAN Technical Component#2 in Table 5-1.
- *Group 2:* Functions instantiated through a specific configuration of a WP3 platform. See for example the on demand wireless service provided by the Transport Technical Component#6 in Table 5-1.
- *Group 3:* Functions deployed by instantiating a program on a WP3 programmable platform. See for example the mobility function implemented in P4 over the WP3 Open Packet Processor, in the Transport Technical Component #5 in Table 5-1.

Group 1 functions can be considered traditional NFV functions, where descriptors for 5G service platforms and orchestrators are more mature, and are usually implemented through a JSON descriptor that points to the corresponding VM/container image in a repository, and provides the required meta-data to configure the function.

Group 2 functions, will be packaged using a descriptor that points to the control element in the 5G OS that controls the WP3 platform where this function can be instantiated. In addition, the descriptor will contain the necessary information for the DO in the 5G OS to build the API call required to instantiate the function.

Group 3 functions are developed as a P4 program defining a datapath that can be instantiated over a programmable WP3 platform. Notice that different programmable platforms can instantiate the same P4 defined datapath, however the P4 program has to be compiled independently for each specific target (WP3 platform). The approach considered in 5G-PICTURE is to have in a central repository an image of each P4 function pre-compiled for all its potential targets. Thus, a P4 function descriptor will point to the specific WP3 platform that can run this function, and to the corresponding pre-compiled image for that target. In order to deploy the function, the Domain Controller in the 5G OS needs to deliver the pre-compiled image to the corresponding target, where a local agent is in charge of instantiating the new image. Finally, when a Group 3 function representing a datapath is deployed, it will return a configuration endpoint, namely a control plane element in the 5G OS that can be used to configure that datapath (e.g. using P4Runtime). Hence, one can define additional Group 2 functions wrapping the API calls to that control plane function.

The interested reader can find a more in depth discussion about the 5G-PICTURE approach to onboard functions executed over programmable platform in deliverable D5.1, section 3.3.1.2 and 3.3.1.3 [52].

5.3 Initial WP4 integration plans toward D4.3 and WP6

We describe in this section our tentative integration plans for the various WP4 functions, which will be executed during the last year of the project. We want to emphasize that these are tentative plans, hence subject to change. However, we believe it is worth to discuss them here in order for the reviewers to have a more clear understanding of the roadmap intended for the various functions.

We foresee two main integration paths for the various WP4 functions. The first one is an integration among various WP4 functions, in order to demonstrate the WP4 vision of a multi-tenant transport network for 5G RANs introduced in deliverable D4.1 [3]. The second integration path is for a WP4 function to be included as part of the demonstration activities in WP6, to support the railways, stadium, or smart city demonstrators.

So far, three tentative integration activities among WP4 functions have been identified, which will be reported in deliverable D4.3:

- RAN integration activity: The goal will be to demonstrate the concept of flexible RAN functional splits implemented through software based RAN functions.
 - o Involved WP4 functions: RAN Tech.Comp.#2 (RAN functional splits over OAI), and RAN Tech.Comp.#3 (FlexRAN controller).
- Transport integration activity: The goal will be to demonstrate end-to-end connectivity between RAN functions over a multi-domain transport network.
 - o Involved WP4 functions: Transport Tech.Comp.#1 (TSON), Transport Tech.Comp.#4 (Segment Routing), and Transport Tech.Comp.#6 (Small Cells with joint access/backhaul).
- Synchronisation integration activity: The goal will be to demonstrate the concept of Synchronisation as a Service introduced in D4.1 [3].
 - o Involved WP4 functions: Synchronisation Tech.Comp.#4 (Sync Harmonizer), and Synchronisation Tech.Comp.#1 (1588 over 802.11ad).

Finally, the following WP4 functions will be used to support WP6 demonstrations:

- Railway demonstration: Transport Tech.Comp.#5 (Mobility over OPP), will be used to support fast handovers in the railway demonstration.
- Stadium demonstration: Transport Tech.Comp.#6 (Small Cells with joint access/backhaul), will be used to provide on demand connectivity services in the fan zone in the stadium use case.
- Smart City demonstration: RAN Tech.Comp.#2 (RAN functional splits over OAI), and Transport Tech.Comp.#1 (TSON) will be used to demonstrate smart city services in Bristol as part of the smart city use case.

Notice that the described plans are tentative, and new WP4 functions may be added to the WP6 demonstrations during the next year.

6 Summary and Conclusions

This deliverable reports on activities associated with all tasks in WP4, namely VNFs and PNFs for dynamic 5G RAN deployments (Task 4.1), Transport slicing for converged wired-wireless FH/BH networks and integration with 5G-PICTURE Orchestrator (Task 4.2), and PNFs and VNFs to support synchronisation services in converged FH/BH networks (Task 4.3). The goal of this deliverable is twofold. On one hand, it analyses and evaluates a number of technical components defined in deliverable D4.1 related to each task. On the other hand, it provides the insights of how each technical component can be integrated with the 5G-PICTURE 5G OS delivered in the context of WP5 activities.

In more detail, a number of 12 technical activities were presented and analysed regarding a) disaggregated RAN VNFs and PNFs (namely Optimal functional split, Implementation of functional split using the OAI platform, Flexible Functional Splits, Disaggregated Heterogeneous Base Station functionality and Wireless Transport Technologies with Functional Split Support), b) transport network technologies (TSN technology, X-Ethernet, Flex-Ethernet, Segment routing, OPP and a solution based on IEEE 802.11 technologies, both for access and BH (802.11ac modems) and c) synchronisation functions (IEEE 1588 over IEEE 802.11ad) and synchronisation harmonizer.

Summary of contributions:

Disaggregated RAN VNFs and PNFs

- Optimal functional split: we optimize the access network parameters such as transmit powers of UEs and downlink beamformers for split 7-2 to improve the SE and EE.
- Implementation of functional split using the OAI platform: our implementation work on the 3GPP split option 8, 7-1 and 6 in the deliverable D4.1. In deliverable D4.2 we present the compression scheme for the low physical functional split, and the latest implementation on the F1 interface aligned with the recent standardisation activities by 3GPP.
- Flexible Functional Splits: we investigate ways to flexibly compose the logical BS from different applied RAN functional splits between disaggregated RAN entities.
- Disaggregated Heterogeneous Base Station functionality: the processes for the CU (RRC, PDCP) and DU (RLC, MAC, PHY) operation already exist in OAI. In deliverable D4.2 we focus on the new elements of the network, being the F1oIP exchange protocol, and the Wi-Fi DUs.
- Wireless Transport Technologies with Functional Split Support: we study eCPRI split [9] that is based on the transport of frequency domain samples between the Distributed Unit (CU) and the Remote Unit (RU).

Transport network technologies

- TSN technology: we enable TSN network programmability based on the SDN design paradigm while an external module to ODL, written in python, is developed to compute the path in the TSN network. Flex-E we provide technology primitives and functionalities while, the performance of a Flex-E demonstrator is evaluated in terms of achieved traffic protection and isolation, average throughput and latency.
- X-Ethernet: X-Ethernet is a Huawei proprietary technology, where X stands for extended distance, expanded granularity and extremely low latency. Technology primitives together with demonstrators to investigate various performance metrics like throughput, delay, jitter etc. are presented.
- Segment routing: a segment routing testbed was built based on Linux container technology and virtual router infrastructure. We provided background information for the basic segment routing functionality and terminology used, we analysed the testbed built, together with the configuration primitives. Relevant information for the functionality of the testbed, messaging and the protocol signalling were presented.
- OPP: two different functionalities are described exploiting the OPP functionality. The first is a stateful load balancer, while the second is a network function for handover management.
- Solution based on IEEE 802.11 technologies, both for access and BH (802.11ac modems): This technical component is used to provide a wireless connectivity service over a distributed area, and it is instantiated over prototype Wi-Fi Small Cells being developed in WP3. The services investigated are a) Service 1: Instantiation of e an access connectivity service composed of virtual APs over a set of physical Aps and b) Service 2: allocation of a connection through the wireless backhaul, which transport the traffic from such access service until a fibre attachment point.

Synchronisation functions

- IEEE 1588 over IEEE 802.11ad: we study and analyze detection and transportation of 1588 frames over mmWave devices with minimal delay, supporting timestamping.
- Synchronisation harmonizer: We devise and analyse the concept of synchronisation harmonizer. One of the main objectives of the synchronisation harmonizer is to collect/obtain the synchronisation capabilities of the nodes in the network. It has an overview on every synchronisation domain within the network. In addition, the synchronisation harmonizer is aware of the established timing paths used for synchronisation, and whether the underlying technologies enable these paths.

Three technical components (13, 14 and 16) regarding synchronisation services defined in deliverable D4.1 were reprioritised and will be presented in deliverable D4.3. A note is also provided regarding integration plans for the next deliverable D4.3 which is coupling with the integration of developed functions with the 5G OS 5G-PICTURE orchestrator and also potential exploitation of the technical components by the project use cases defined in WP6.

7 References

- [1] 3GPP TR 38.816 V15.0.0, Study on CU-DU lower layer split for NR, 2018.
- [2] 3GPP TR38.801 V14.0.0, "Study on new radio access technology: Radio access architecture and interfaces (Release 14).".
- [3] 5G-PICTURE D4.1, "State of the art and initial function design," 28 February 2018. [Online]. Available: https://www.5g-picture-project.eu/publication_deliverables.html. [Accessed 9 September 2018].
- [4] 5G-PICTURE D3.1, "Initial report on data plane programmability and infrastructure components," 4 2018 April. [Online]. Available: https://www.5g-picture-project.eu/publication_deliverables.html. [Accessed 11 April 2018].
- [5] 5G-PICTURE - D3.2, "Intermediate report on Data Plane Programmability and infrastructure components," November 2018.
- [6] 3GPP TS 38.401 V15.2.0, NG-RAN; Architecture description, 2018.
- [7] 3GPP TR 36.756 V1.0.0, Study on architecture evolution for Evolved Universal Terrestrial Radio Access Network (E-UTRAN), 2017.
- [8] NGMN Alliance, NGMN Overview on 5G RAN Functional Decomposition (v1.0), 2018.
- [9] CPRI forum, eCPRI interface specification (v1.0), 2017.
- [10] China Mobile Research Institute, Next generation fronthaul interface (version 1.0), 2015.
- [11] IEEE 1914 working group, "P1914.3: Standard for radio over Ethernet encapsulations and," [Online]. Available: <http://sites.ieee.org/sagroups-1914/p1914-3/>.
- [12] xRAN Fronthaul Working Group, xran fronthaul control, user and synchronization plane (v2.0), 2018.
- [13] Telecom Infra Project, Creating an ecosystem for vRANs supporting non-ideal fronthaul, 2018.
- [14] 3GPP TR 38.801 V14.0.0, Study on new radio access technology: Radio access architecture and interfaces, 2017.
- [15] O. E. F. T. a. T. L. M. E. G. Larsson, "Massive MIMO for next generation wireless systems," IEEE Communications Magazine, vol. 52, no. 2, pp. 186-195, 2014.
- [16] J. F. a. G. Fettweis, "Power allocation for massive MIMO-based, fronthaul-constrained Cloud RAN systems," Online: <https://arxiv.org/abs/1810.04529>, 2018.
- [17] J. P. A. Y. a. R. W. H. S. Park, "Exploiting Spatial Channel Covariance for Hybrid Precoding in Massive MIMO Systems," IEEE Transactions on Signal Processing, vol. 65, no. 14, pp. 3818-3832, 2017.
- [18] C. S. S. L. S. M. J. J. a. I. L. Y. Jeon, "New Beamforming Designs for Joint Spatial Division and Multiplexing in Large-Scale MISO Multi-User Systems,," IEEE Transactions on Wireless Communications, vol. 16, no. 5, pp. 3029-3041, 2017.
- [19] "nFAPI and FAPI Specification".
- [20] 3GPP TS 38.470 V15.0.0, NG- RAN; F1 general aspects and principles, 2018.
- [21] 3GPP TS 38.401 V15.2.0, NG-RAN; Architecture description, 2018.
- [22] 3GPP TS 36.876 V15.0.0, Study on eNB(s) Architecture Evolution for E-UTRAN and NG-RAN, 2018.
- [23] X. Foukas, N. Nikaiein, M. M. Kassem, M. K. Marina and K. P. Kontovasilis, "FlexRAN: A Flexible and Programmable Platform for Software-Defined Radio Access Networks," in ACM CoNEXT, 2016.
- [24] 3. T. 38.401, 3GPP TS 38.401 V0.2.0 (2017-07), 3rd Generation Partnership Project; Technical Specification Group Radio Access Network; NGRAN; Architecture description (Release 15), 2017.
- [25] C. Z. P. B. T. K. N. N. a. L. T. N. Makris, "Cloud-Based Convergence of Heterogeneous RANs in 5G Disaggregated Architectures," IEEE International Conference on Communications (ICC), pp. 1-6, 2018.

- [26] C. Z. S. K. T. K. I. S. a. L. T. . Makris, “Enabling open access to LTE network components; the NITOS testbed paradigm,” Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft), pp. 1-6, 2015.
- [27] Protobuf-c, Protobuf-c–C bindings for Google’s Protocol Buffers–Google Project Hosting, Available: <https://code.google.com/archive/p/protobuf-c/> [Accessed 10 Oct. 2018].
- [28] P. Biondi, “Scapy: explore the net with new eyes,” Technical report, EADS Corporate Research Center, 2005.
- [29] “Cloud-Init: the standard for customizing Cloud Instances,” [Online]. Available: <https://cloud-init.io/>.
- [30] OAI Consortium, “IF4p5,” [Online]. Available: <https://gitlab.eurecom.fr/oai/openairinterface5g/wikis/how-to-connect-cots-ue-to-oai-enb-via-ngfi-rru>.
- [31] P. H. P. a. P. M. Y. Zeng, “A first look at 802.11ac in action: Energy efficiency and interference characterization,” in IFIP Networking Conference, 2014.
- [32] R. Khalili, “IETF 87: Congestion Control of Multipath TCP: Problems and Solutions,” [Online]. Available: <https://datatracker.ietf.org/meeting/87/materials/slides-87-icrg-7>.
- [33] A. P. Q. H. J. & L. S. Walid, “Balanced linked adaptation congestion control algorithm for MPTCP,” Internet Draft draft-walid-mptcp-congestion-control-04, 2016.
- [34] I. OIF-FLEXE-02.0, “Flex Ethernet 2.0 Implementation Agreement,” Optical Internetworking Forum (OIF), 2018.
- [35] Common Public Radio Interface specification version 7.0, <http://www.cpri.info/spec.html>.
- [36] M. B. J. S. A. B. R. Enns, “Network Configuration Protocol (NETCONF),” IETF 6241, 2011.
- [37] 3GPP TR 38.913 V15.0.0, “Study on Scenarios and Requirements for Next Generation Access Technologies,” 2018.
- [38] IETF RFC 8402, “Segment Routing Architecture,” 2018.
- [39] K. M. K. T. Clarence Filis, Segment Routing, Part I, Cisco Systems, 2016.
- [40] Huawei Technologies, “New IP Technologies,” 2018. [Online]. Available: <http://support.huawei.com/enterprise/en/doc/EDOC1000173015?section=j004>.
- [41] IETF, “Deterministic Networking working group,” [Online]. Available: <https://datatracker.ietf.org/wg/detnet/about/>.
- [42] IETF Draft, “Enhanced Virtual Private Networks (VPN+),” 2018.
- [43] IETF draft - OSPF WG, “draft-psenak-ospf-segment-routing-extensions,” 2018.
- [44] O. D. Anselme Sawadogo, “FRR OSPF-SR,” 2018. [Online]. Available: <https://github.com/FRRouting/frr/blob/master/doc/developer/ospf-sr.rst>.
- [45] 5G-XHaul deliverable D3.3, “D3.3 5G-XHaul algorithms and services Design and Evaluation,” 2018.
- [46] M. C.-M. D. B. A. A. J. J. & C.-C. M. Grandi, “SWAM: SDN-based Wi-Fi Small Cells with Joint Access-Backhaul and Multi-Tenant Capabilities,” arXiv preprint arXiv:1804.07106., 2018.
- [47] IEEE Std 1588-2008, IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems, 2008.
- [48] IEEE Computer Society LAN/MAN Standards Committee, Wireless LAN medium access control (MAC) and physical layer (PHY) specifications, 2016.
- [49] I. Hadžić and D. R. Morgan, “On packet selection criteria for clock recovery,” in 2009 International Symposium on Precision Clock Synchronization for Measurement, Control and Communication, Brescia, 2009.
- [50] P. I. M. F. a. T. T. S. Ruffini, “A Novel SDN-Based Architecture to Provide Synchronization as a Service in 5G Scenarios,” IEEE Communications Magazine, vol. 55, no. 3, pp. 210-216, 2017.

- [51] l. 1. m. interface, <http://manpages.ubuntu.com/manpages/bionic/man8/pmc.8.html>.
- [52] 5G-PICTURE - D5.1, "Relationships between Orchestrators, Controllers, slicing systems," 30 November 2018. [Online]. Available: https://www.5g-picture-project.eu/publication_deliverables.html. [Accessed 2018].
- [53] ITU-T Recommendation G.8275.1, Precision time protocol telecom profile for phase/time synchronization with full timing support from the network, 2016.
- [54] ITU-T Recommendation G.8265.1, Precision time protocol telecom profile for frequency synchronization, 2014.
- [55] 3GPP TS 38.473 V15.0.0, "NG- RAN; F1 application protocol (F1AP)," 2018.
- [56] OpenAirInterface, [Online]. Available: <http://www.openairinterface.org/>. [Accessed 27 3 2018].

8 Acronyms

Acronym	Description
3GPP	Third Generation Partnership Project
5G	Fifth Generation
A-CPI	Application-Controller Plane Interface
A-MPDU	Aggregated MAC Protocol Data Unit
A-MSDU	Aggregated MAC Service Data Unit
AP	Access Point
API	Application Programming Interface
BBU	Base Band Unit
BC	Boundary Clock
BH	Backhaul
BMCA	Best Master Clock Algorithm
BS	Base Station
BSS	Business Support System
BVT	Bandwidth Variable Transponders
CBR	Constant Bitrate
CCAMP	Common Control and Measurement Plane
CDMA	Carrier Sense Multiple Access
CDR	Clock and Data Recovery
CLI	Command Line Interface
CN	Core Network
CO	Central Office
CoMP	Coordinated Multi-Point
COTS	Commercial Off-The-Shelf
CP	Control Plane
CPRI	Common Public Radio Interface
C-RAN	Cloud-RAN
CSMA	Carrier-Sense Multiple Access
cTE	constant Time Error
CU	Centralised Unit
CWDM	Coarse WDM
DA	Destination Address
DA-RAN	Disaggregated Radio Access Network
DAS	Distributed Antenna System
DC	Data Center

DCI	Data Center Interconnect
DetNet	Deterministic Ethernet
DHCP	Dynamic Host Configuration Protocol
DL	Downlink
DMG	Directional Multi-Gigabit
DMS	Directed Multicast Service
D-RAN	Distributed-RAN
DA-RAN	Disaggregated-RAN
DS	Distributed System
DSCP	Differentiated Services Code Point
dTE	dynamic Time Error
DU	Distributed Unit
DWDM	Dense WDM
eCPRI	enhanced CPRI
EDFA	Erbium Doped Fibre Amplifier
eMBB	enhanced Mobile Broadband
eNB	evolved Node B
EoMPLS	Ethernet over Multi-Protocol Label Switching
EoS	Ethernet over SONET/SDH
EPC	Evolved Packet Core
EPL	Ethernet Private Line
ETSI	European Telecommunications Standards Institute
EVM	Error Vector Magnitude
EXP	Experimental bits
F1AP	F1 Application Protocol
F1oIP	F1 over IP
FDD	Frequency-Division Duplexing
FFT	Fast Fourier Transform
FH	Fronthaul
Flex-E	Flexible Ethernet
FlexO	Flexible OTN
FPGA	Field-Programmable Gate Array
FTM	Fine Timing Measurement
FTS	Full Timing Support
FTS	Full Timing Support
FWA	Fixed Wireless Access
GEO	Geosynchronous

GFP	Generic Framing Procedure
GNSS	Global Navigation Satellite Service
GPP	General Purpose Processor
GPS	Global Positioning System
GTP	GPRS Tunneling Protocol
HARQ	Hybrid Automated Repeated reQuest
HDLC	High-Level Data Link Control
HL-SYNC	Higher Layer timer Synchronisation
HSS	Home Subscriber Server
HW	Hardware
I/Q	In-phase/Quadrature
IAB	Integrated Access and Backhaul
ICP	Internet Content Providers
IFFT	Inverse Fast Fourier Transform
IS-IS	Intermediate System to Intermediate System
ISP	Internet Service Provider
KPI	Key Performance Indicator
LTE	Long Term Evolution
LTE-A	LTE Advanced
MAC	Medium Access Control
MAN	Metropolitan Area Network
max TE	maximum absolute Time Error
MCS	Modulation and Coding Scheme
MEF	Metro Ethernet Forum
MH	Midhaul
MME	Mobility Management Entity
mMTC	massive Machine-Type Communications
mmWave	millimetre Wave
MNO	Mobile Network Operator
MTIE	Maximum Time Interval Error
MTU	Maximum Transmission Unit
NB-IOT	NarrowBand-Internet of Things
nFAPI	network Functional Application Platform Interface
NFV	Network Functions Virtualisation
NGFI	Next Generation Fronthaul Interface
NGMN	Next Generation Mobile Networks
NG-PON2	Next-Generation PON 2

NG-RAN	Next Generation RAN
NIC	Network Interface Card
NLoS	Non-Line of Sight
NR	New Radio
NS	Network Service
NSI	Network Slice Instance
NSSI	Network Slice Sub-network Instance
OAI	OpenAirInterface
OBSAI	Open Base Station Architecture Initiative
OC	Ordinary Clock
OCXO	Oven Controlled Crystal Oscillator
ODU	Optical channel Data Unit
OFDM	Orthogonal Frequency-Division Multiplexing
OLT	Optical Line Terminal
ONF	Open Networking Foundation
ONU	Optical Network Unit
OPP	Open Packet Processor
OS	Operating System
OSPF	Open Shortest Path First
OSS	Operation Support System
OTN	Optical Transport Network
PCS	Physical Coding Sublayer
PDCP	Packet Data Convergence Protocol
PDU	Protocol Data Unit
PDV	Packet Delay Variation
P-GW	Packet Data Network Gateway
PHY	Physical
PLME SAP	PHY-layer Management Entity SAP
PMD	Physical Media Dependent
PMP	Packet Manipulator Processor
PNF	Physical Network Function
PON	Passive Optical Network
PoP	Point of Presence
PoS	Packet over SONET
PPDU	PHY Protocol Data Units
PPP	Point-to-Point Protocol
PPS	Pulse-Per-Second

PRACH	Physical Random Access CHannel
PRTC	Primary Reference Time Clock
PSDU	Physical Service Data Unit
PTP	Precision Time Protocol
PTS	Partial Timing Support
QoE	Quality of Experience
QoS	Quality of Service
RA	Receiver Address
RAN	Radio Access Network
RAT	Radio Access Technology
RAU	Radio Aggregation Unit
RCC	Radio Cloud Center
RF	Radio Frequency
RLC	Radio Link Control
RN	Remote Node
RoE	Radio-over-Ethernet
RRC	Radio Resource Control
RRH	Remote Radio Head
RRU	Remote Radio Unit
RTT	Round Trip Time
RU	Radio Unit
SAP	Service Access Point
S-BVT	Sliceable-Bandwidth Variable Transponder
SCF	Small Cell Forum
SDH	Synchronous Digital Hierarchy
SDN	Software-Defined Networking
SD-RAN	Software-Defined RAN
SF	Service Function
SFC	Service Function Chaining
SFP+	Enhanced Small Form-factor Pluggable
S-GW	Serving Network Gateway
SID	Segment Routing Identities
SLA	Service Level Agreement
SML	Station Management Layer
SNR	Signal to Noise Ratio
SONET	Synchronous Optical Networking
SPE	Synchronous Payload Envelope

SR	Segment Routing
STA	Station
STF	Short Training Field
SW	Software
SyncE	Synchronous Ethernet
TA	Timing Advertisement
TAE	Time Alignment Error
T-BC	Telecom Boundary Clock
TC	Transparent Clock
TCXO	Temperature Compensated Crystal Oscillator
TDD	Time-Division Duplexing
TDEV	Time Deviation
TDMA	Time Division Multiple Access
TDM-PON	Time Division Multiplexed PON
TE	Traffic Engineering
TEID	Tunnel Endpoint Identifier
T-GM	Telecom GrandMaster
TIE	Time Interval Error
TM	Timing Measurement
TN	Transport Node
TOSCA	Topology and Orchestration Specification for Cloud Applications
TSF	Timing Synchronisation Function
TSON	Time Shared Optical Network
TSU	Timestamping Unit
TTI	Transmission Time Interval
TWSTT	Two-Way Satellite Time Transfer
UDN	Ultra-Dense Networking
UE	User Equipment
UL	Uplink
UNI	User Network Interface
UP	User Plane
uRLLC	ultra-Reliable and Low-Latency Communications
UTC	Coordinated Universal Time
vAP	virtual Access Point
vBBU	virtual BBU
VBR	Variable Bitrate
VC	Virtual Concatenation

VLIW	Very Long Instruction Word
VMiet	Virtual Machine
VNF	Virtual Network Function
VPN	Virtual Private Network
WAN	Wide Area Network
WDM	Wavelength Division Multiplexing
WRPTP	White Rabbit extension to PTP