5G Programmable Infrastructure Converging disaggregated network and compUte REsources

D4.3 Integration of developed functions with 5G-PICTURE orchestrator

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**: 34 months

Call: H2020-ICT-2016-2 **Date of delivery**: 1st January 2020

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:	**D4.3**
Document Title:	Integration of developed functions with 5G-PICTURE orchestrator
Version:	1.0
Delivery Date:	31st November 2019 (**1st January 2020**)
Responsible:	Blu Wireless Technology (**BWT**)
Editor(s):	Peter Legg (**BWT**)
Authors:	Robert Schmidt (**EUR**), Navid Nikaein (**EUR**), Daniel Camps-Mur (**I2CAT/UPC**), Ferran Cañellas (**I2CAT**), Ricardo González (**I2CAT**), Eduard García-Villegas (**I2CAT/UPC**), , James Cheung (**BWT**), Peter Legg (**BWT**), Kostas Katsalis (**HWDU**), Sushmit Bhattacharjee (**HWDU**), Atul Kumar (**TUD**), Jay-Kant Chaudhary (**TUD**), Jesús Gutiérrez (**IHP**), Darko Cvetkovski (**IHP**), Meysam Goodarzi (**IHP**), Vladica Sark (**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
0.0	Table of Contents	Peter Legg (BWT)	19 March 2019
0.1	First Draft	Various	28 May 2019
0.2	Second draft after merging of contributions	Various	22 Oct. 2019
0.3	Section “4.2 Integrated RAN Transport [HWDU, EUR]" flagged as missing Section 4.4 completed by UTH	Various	6 Nov 2019
0.4	Integrated RAN Transport section added (renamed, see 4.2. Time-Sensitive Networking for 5G Networks). Header numbering fixed. Inserted revised 5.3 text (i2CAT)	Kostas Katsalis (HWDU), Peter Legg (BWT), Daniel Camps (I2CAT)	6 Nov 2019
0.5	Added 3.2.3 content (COP module implemented within ONOS)	Thierno Diallo (UNIVBRIS-HPN)	7 Nov 2019
0.6	Added section 4.6 References done	Robert Schmidt (EUR) Peter Legg (BWT)	26 Nov 2019
0.7	Update to section 5.3 Introduction added Acronyms added Reordered sections 2 to 4 to align with deliverable D4.2	Daniel Camps (I2CAT) Peter Legg (BWT)	29 Nov 2019
0.8	Removed unnecessary acronyms	Peter Legg (BWT)	30 Nov 2019
0.9	Revised contributions on Synchronization Harmonizer and Synchronization demonstrator	Darko Cvetkovski (IHP) Meysam Goodarzi (IHP) Vladica Sark (IHP) Jesús Gutiérrez (IHP)	12 Dec 2019
0.91	Revisions to section 5 Removed the section related to JOX integration with TSN since we will report to deliverable D5.4. Editing	Kostas Katsalis (HWDU) Peter Legg (BWT)	17 Dec 2019
0.92	General review	Daniel Camps (I2CAT)	19 Dec 2019
0.93	Additions to Section 5.4	Darko Cvetkovski (IHP) Jesús Gutiérrez (IHP)	26 Dec 2019
1.0	Final review and submission to the EC	Jesús Gutiérrez (IHP)	1 Jan 2020

Table of Contents

List of Figures

List of Tables

Executive Summary

This document corresponds to deliverable D4.3, “Integration of developed functions with 5G-PICTURE orchestrator” of the Horizon 2020 5G-PICTURE project. This deliverable concludes the work in WP4 by reporting results of the Technical Components in the areas of Synchronisation, Dynamic RAN Deployments and data Transport that were not included in deliverable D4.2. Furthermore, this deliverable demonstrates for the first time how the various physical and virtual network functions developed in WP4 can work together in three integrated demonstrators in the RAN, transport and synchronisation domains.

In the synchronisation domain prototypes have been constructed using three different air interface technologies, IEEE 802.11ad, IEEE 802.11ac and LTE, and their assessed performance is presented here, extending results from earlier WP4 deliverables. For IEEE 802.11ad, BWT demonstrate that their hardware (HW) timestamping has an accuracy of ~1 ns. The use of PTP over IEEE 802.11ac achieves an accuracy of about 0.5 ms. Preliminary results are presented for time synchronisation for an LTE-based fronthaul system. Additionally, two pieces of work that consider synchronisation over heterogeneous transport networks have been developed. The NITOS test bed is used to demonstrate a live migration of a cloud component of a disaggregated LTE base station using a PTP synchronisation framework to ensure seamless operation. The Synchronisation Harmoniser concept is also presented as a means to coordinate and manage synchronisation with multiple clock sources and potential clock paths over heterogeneous technologies. The synchronisation algorithms are presented and simulated.

In the area of massive MIMO, a new 2D phased array antenna is presented that improves the SINR in a system level simulation.

In the Transport function domain, improvements to the Time Shared Optical Network (TSON) are presented. The control plane has been redesigned to be more flexible and scalable, using an ONOS controller and Netconf configuration method.

Finally, three demonstrators in the RAN, transport and synchronization domains showcase how the WP4 Technical Components can work together to deliver more complex services. First, in the RAN domain we show how Ethernet TSN networks based on IEEE 802.1Qbv and IEEE 802.1Qbu can be used to carry real fronthaul traffic generated by software Base Stations components, while protecting high-priority traffic flows even in overloaded conditions. Second, in the transport domain we demonstrate a hierarchical control plane connecting heterogeneous transport domains that features some of the technologies developed in WP4 (e.g. wireless backhauling). In the synchronisation domain we present the design and initial results of an integrated demonstrator demonstrating the capabilities of the Synchronization Harmonizer function.

1 Introduction

The aim of this deliverable is to capture research work on network functions in the areas of Synchronisation, Dynamic RAN Deployments and data Transport. Furthermore, three demonstrators that integrate these functions together are described and results are presented.

WP4 develops the physical and virtual functions that execute over the WP3 platforms and are orchestrated by the WP5 5G OS, Figure 1-1. In earlier deliverables D4.1 [1] and D4.2 [2] the state of the art was assessed, network functions were designed and initial implementations were presented. This deliverable presents the finalised implementations and shows the integration of functions under the management of the 5G OS.

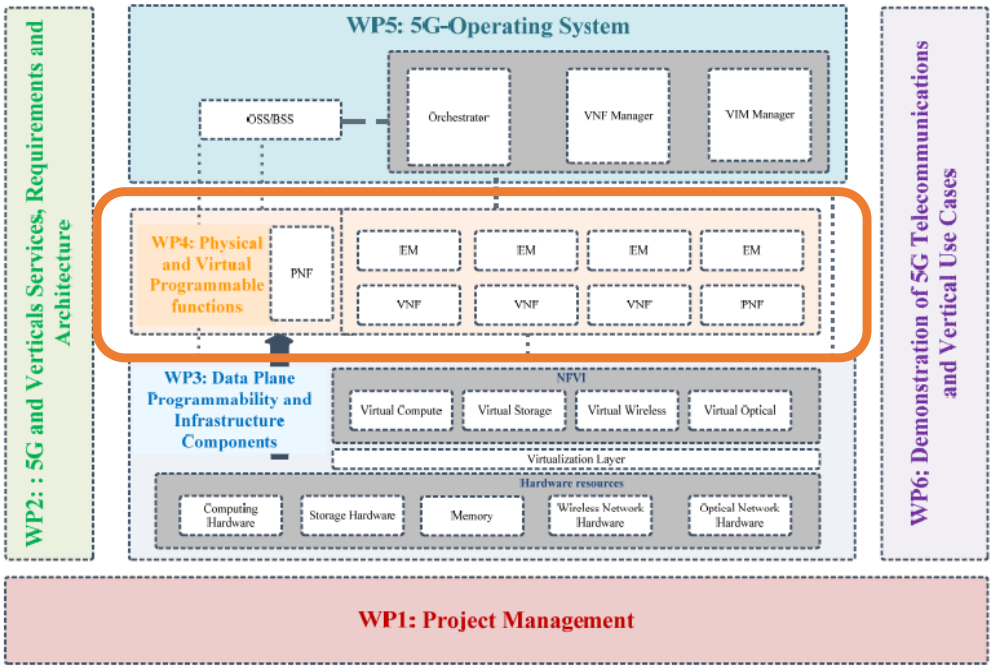

Figure 1-1. Placement of WP4 within 5G-PICTURE.

Table 1-1 summarises the technical components developed in WP4 and where they have been delivered.

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, update in D4.3
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, update in D4.3
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, update in D4.3
13	IEEE 1588 over off-the-shelf IEEE 802.11ac	D4.3
14	Heterogeneous synchronisation transport testbed	D4.3
15	Synchronisation harmonizer	D4.3
16	Over-the-air synchronisation for FH networks	D4.3

In addition, the integrated demonstrators draw together a selection of the previous Technology Components, as well a pre-existing network functions, to demonstrate more complex network function. These demonstrators are summarised in Table 1-2.

Table 1-2: Summary of the integrated demonstrators

Integrated Demonstrator	*Technical Components*
TSN for 5G Fronthaul	TC2 (OAI base station functions), TC8 (Ethernet TSN[1])
Multi-domain Transport	TC11 (joint wireless access and backhaul), TC6 (TSON[2]), pre-existing Openflow switching technology and a hierarchical SDN control plane from WP5
Synchronization integration	TC15 (synchronisation harmonizer), TC12 (1588 over IEEE 802.11ad), and a pre-existing synchronization technology developed in WP3

A complementary goal of the developed integrated demonstrators is to support the demonstrations of 5G OS in WP5. In particular, the TSN for fronthaul and the multi-domain transport demonstrators are used to support 5G OS demonstrations in deliverable D5.4 [35].

Organisation of the document

This deliverable is structured into 3 sections that describe the additional functional components in the areas of, Dynamic RAN Deployments, data Transport and Synchronisation (section 2 to 4). Section 5 then describes the integrated demonstrators, and Section 0 summarises and concludes the deliverable.

[1] We have used TSN instead of X-Ethernet, as originally used for TC8, since TSN is the technology that is currently enjoying more market support. Both X-Ethernet and TSN though pursue the same goal of being able to support latency sensitive services over Ethernet

[2] Due to lack of time TSON could not be integrated in the multi-domain testbed. However the protocol adapters (COP) required for such integration are developed in this deliverable as reported in section 3.

2 Dynamic 5G-RAN deployments

2.1 Introduction

In this chapter we update work performed on Technical Component 1 by addressing refinements to the beam pattern of a massive MIMO 2D phase array antenna to improve the system performance.

2.2 Technical Component 1: Optimal Functional Split and mMIMO Optimization

To achieve high data-rates and capacity massive multiple-input multiple-output (mMIMO) system based on two dimensional (2D) active antenna arrays (AAAs) is a promising solution. Here, we would like to summarize what has been done and what comes now:

- Goal was to find optimal split for mMIMO, and optimize beamforming for that.
- Analysis of optimal split in terms of data rate and complexity of the HW was done in deliverables D3.1, D3.2 and architecture for Airrays mMIMO designed based on this, we expanded our activity of optimal splits with 5G New Radio (NR) as shown in Figure 2-1.
- In deliverables D4.1 and D4.2 joint beamforming was analysed for that split and it was shown by considering optimal split spectral efficiency (SE) and energy efficiency (EE) can be improved.
- Now we finally want to optimise the beam patterns, to get optimal performance for the system designed based on this optimal split.

Fronthaul bandwidth in cloud radio access network (C-RAN) can be significantly reduced with an appropriate functional split by offloading more signal processing functionalities to the remote radio unit (RRU). Figure 2-1 shows that RRU complexity increases for split 7.2 compared to split 8. However, at the same time, the required FH bandwidth is significantly reduced, which is one of the main motivations to consider the 7.2 is an optimal functional split. Therefore, based on the compute the computational versus fronthaul bandwidth requirement of the RRU with 5G New Radio (NR) considering functional split 7.2 as recently standardized split by the xRAN Forum.

Based on the optimal split definition for mMIMO, here, we extend our contribution to beam pattern optimization. We consider mMIMO based on the 2D-AAA with beamforming both in azimuth and elevation directions in order to improve the spectral efficiency and energy efficiency.

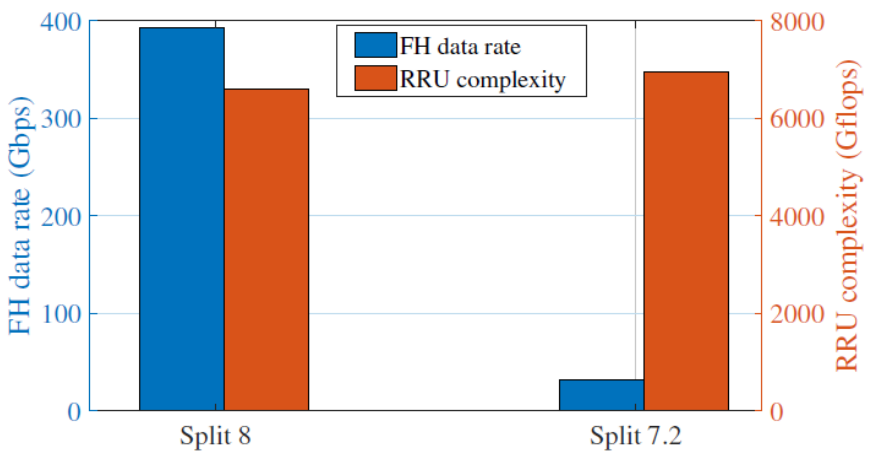

Figure 2-1: Required FH data rate versus RRU complexity for split 8 and split 7.2.

Moreover, optimized 2D-AAA design based on subarrays is consider in order to reduce cost and improved spectral efficiency as shown in Figure 2-2. We demonstrated from the Table 2-1 that by proper designing of the 2D-AAA with optimal sub-array amplitude and phase tapering, it is possible to reduce the side lobe level (SLL) as shown in Figure 2-3. The SLL is the level of the highest side lobe in the pattern. Moreover, due to the reduction of the SLL, an increase in spectral efficiency and SINR performance can be obtained. These generated beam patterns can be tested on the Airrays antenna.

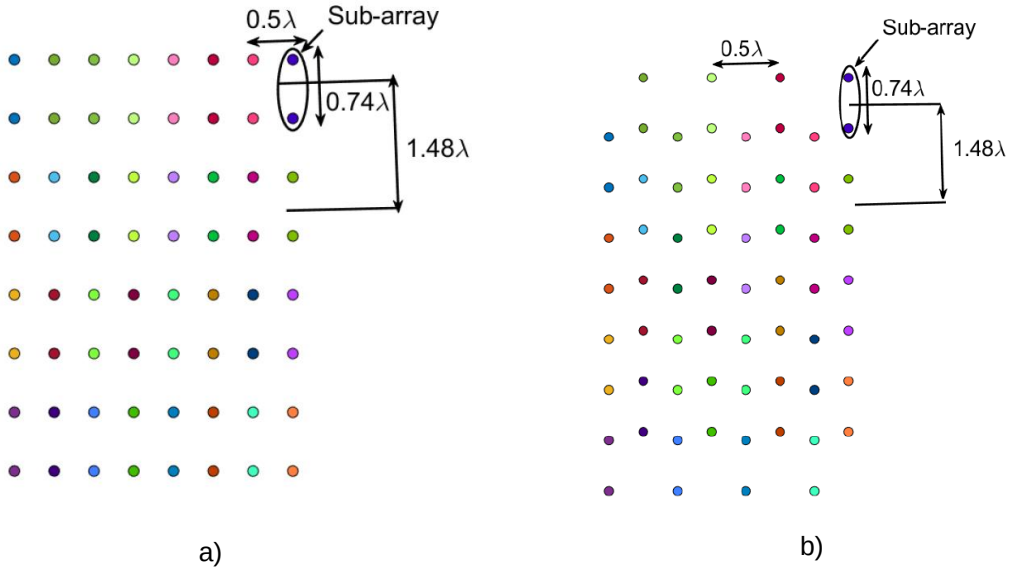

Figure 2-2: a) Front view of Type-1 2D-AAA design, b) Front view of Type-3 2D-AAA design.

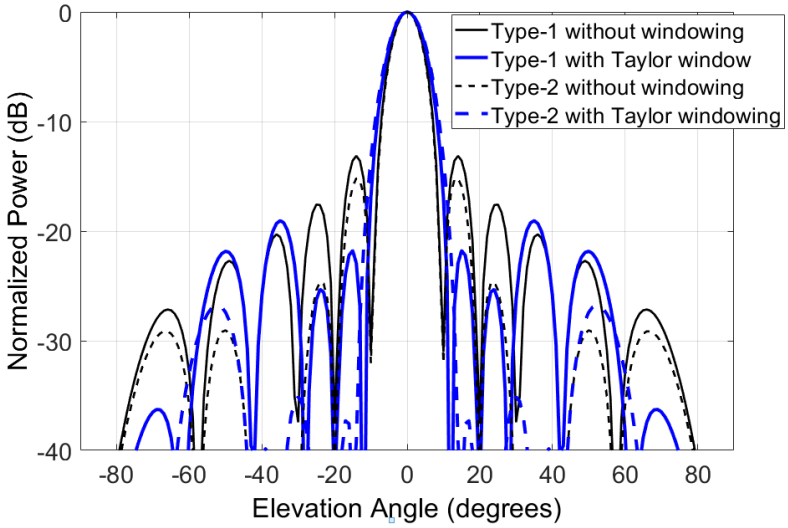

Figure 2-3: 2D-AAA patters synthesis results for Type-1 and Type-2 with optimal Taylor window.

Table 2-1: Comparison of the different array configurations with Taylor window for fixed steering beam width and side lobe level

2D-AAA Design	SLL without Steering	SLL at 10° Steering	Steering beam width below 15dB SLL	SINR at 50% CDF
Type-1	19.27dB	7.52dB	$5° \rightarrow [-2.5°$ to $2.5°]$	4.8dB
Type-2	27.31dB	14.41dB	$19° \rightarrow [-9.5°$ to $9.5°]$	7.5dB

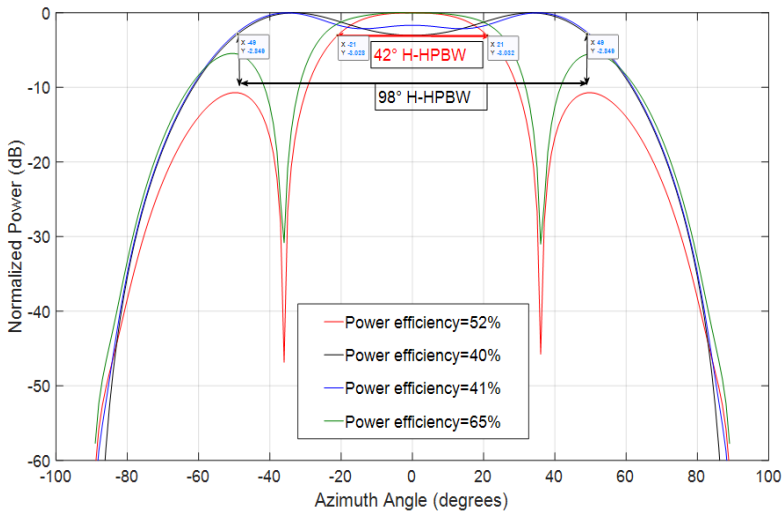

Figure 2-4: Beam pattern for broadband beam generated using 2D-iterative Fourier transform algorithms and corresponding power efficiency of the generated beam for 2D-AAA design for the mMIMO system.

Moreover, in the practical deployment of the mMIMO system, the control signaling must be delivered to all users in the cell reliably. Therefore, it is desired to have an energy efficient broadbeam with no variation in radiated power in all directions. In the current cellular networks, this can be achieved via sending control signaling with one antenna, for instance Cell-specific Reference Signaling.

However, in mMIMO systems, due to the low-power units, sending signal using only one antenna is extremely power inefficient. Therefore, main motivation to develop the broadbeam generation with higher energy efficiency is to form reliable control channels. Therefore, beam-pattern optimization for broadbeam generation is also consider in this deliverable. With this aim, we propose a novel techniques to generate the energy efficient broadbeam, which radiate equal power in all directions, for broadcast the control message to all the users which guaranteeing the quality of service for randomly deployed users in any place in a cell.

Moreover, propose approach of the broadbeam generations is based on the use of the iterative Fourier transform (FT). Therefore, the proposed method shows the superiority inters of the energy efficiency and complexity as compared to those give in the literature.

Finally, after designing the different types sub-array based 2D AAA, in this section, we employ system level simulations to analyse the SINR performance using the 3D channel model given by 3GPP for UMi scenarios. For this we are using the open-source Quadriga implementation of the 3D channel model. we generate the 3D channel coefficients that can be used further to evaluate the system level performance in terms of the SINR for different design of 2D-AAA system.

Figure 2-5 shows the positions of the different UE and BS in the simulated scenario. The link between each UE and the BS. Here, we consider 8 different UEs and each UE has one omni antenna at fixed location connected to a single BS with 3 sectors. Each sector contains a sub-array based 2D-AAA

In Figure 2-6 we show the cumulative distribution function (CDF) of the SINR based on the simulation parameters given in Table I. As it can be seen, the staggered array shows a higher SINR when compared with the regular one, due to its reduced SLL, which yields lower overall interference. The results clearly showed that staggered performed better than regular ones in terms of SINR.

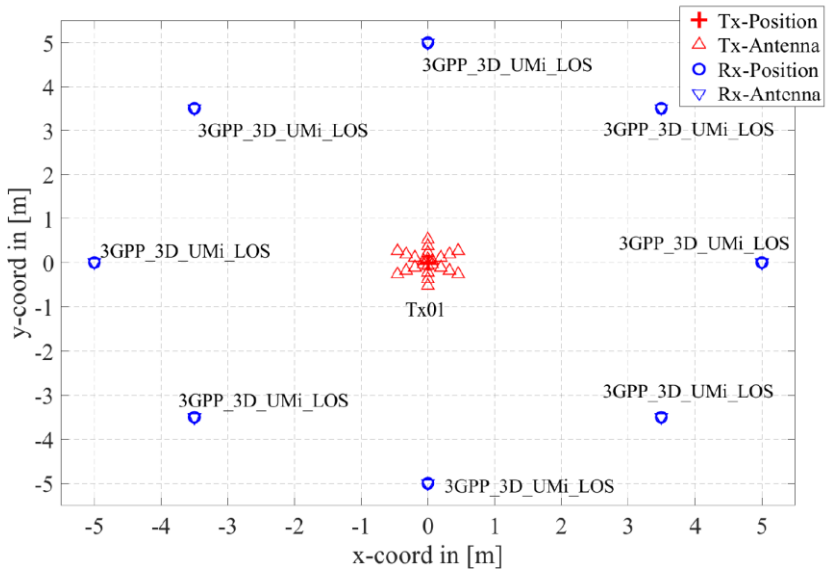

Figure 2-5: Simulation setup to analyses the SINR performance using 3D channel model for different type of 2D-AAA design for mMIMO system.

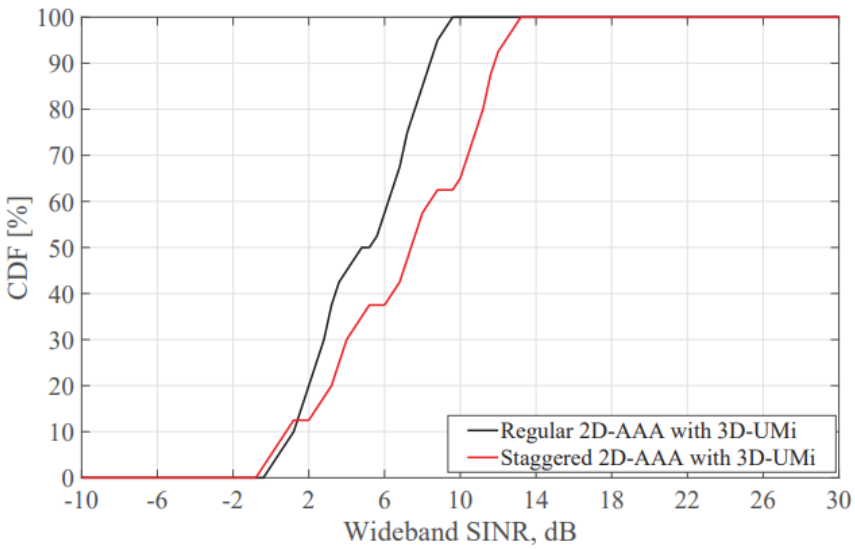

Figure 2-6: CDF vs SINR performance analyses using the 3D channel model under the scenarios of UMi specified by 3GPP in TR 38.913.

3 Transport Functions

3.1 Introduction

In this section, the Time Shared Optical Network (TSON) technical component described in deliverables D4.1 [2] and D4.2 [1] has been updated. Indeed, TSON node has evaluated and provide enhanced features and functionalities to perform better performance in the data plane. Since OpenFlow is obsoleted and not much flexible to support efficiently the evaluation of the enhanced TSON node, the control plane has also been redesigned. To bring more flexibility and scalability of the TSON node in term of abstraction in the control plane, the OpenConfig yang model has been investigated. The following sections present the new TSON design and the new control plane based on ONOS controller and NETCONF/OPENCONFIG implemented to increase the performance of the TSON domain.

In addition, this section also describes the development of a Control Orchestration Protocol (COP) adapter for the TSON controller. This adapter is the necessary interface required to integrate a TSON domain with the hierarchical control plane used in the multi-domain transport demonstrator reported in Section 5.3.

3.2 Technical component 6: TSON

The TSON technology is a dynamic optical transport network solution. This solution proposed by High Performance Networks (HPN) Group, University of Bristol, to provide high bandwidth and low-latency connectivity in support of the 5G technology requirements. TSON is an active Wavelength Division Multiplexing (WDM) solution. It offers variable sub-wavelength switching granularity and the capability to dynamically allocate optical bandwidth elastically with a unique time-sharing mechanism. The time-sharing mechanism offers dynamic connectivity with different granularity of bandwidth to achieve the required Quality of Service (QoS). This technology is the first multiple protocol programmable interface that meets 5G KPIs.

Figure 3-1 illustrates the TSON edge node data plane architecture. The ingress TSON edge nodes are responsible for parsing, aggregation, and mapping of any input traffic combination with different bandwidth into TSON output with different granularity, while the egress edge nodes have the reverse functionality. The TSON implementation parameters are programmable by the SDN controller. These parameters include, for example, quality of Transmission (QoT) overhead programmable, time-slice numbers in a frame programmable and time-slice allocation.

The new TSON design can support up to 4 VLANs per implemented line. Unlikely the previous one where the transmission mode (Ethernet or TSON) is applied to all the system, the new TSON design allows specifying the mode of each single output line. In other words, in the same FPGA platform we can transmit both mode Ethernet and TDM on different outputs lines.

The TSON node is a fully programmable FPGA platform. The previous version of the TSON node uses an Ethernet port for programming the Look-Up Table (LUT). In this case, an optical channel is lost because it cannot be used to transmit data. The enhanced TSON is pluggable inside the computer box and is programmable via PCIe.

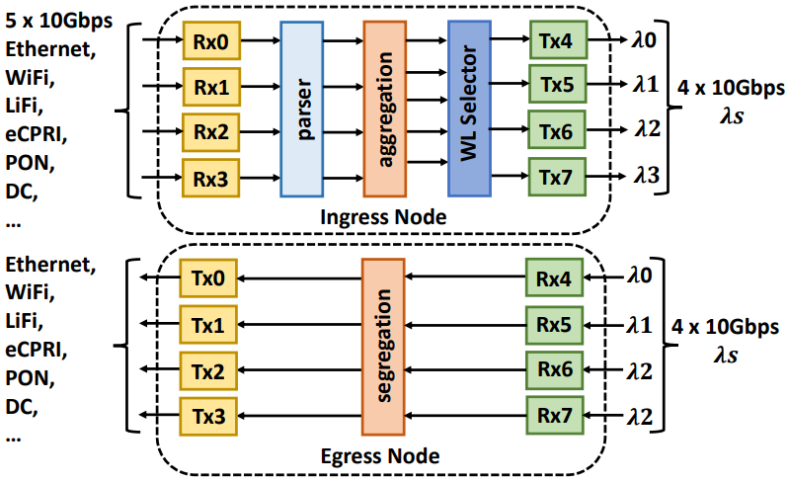

Figure 3-1: TSON Architecture.

The control plane to manage the TSON network has also evaluated. To bring mode flexibility and since the TSON node can be scalable, OpenConfig over NETCONF is used. To handle all the new properties of TSON, it is necessary to extend the current OpenConfig standard.

3.2.1 OpenConfig extended for TSON

OpenConfig Yang model is a neutral-vendor model data developed for managing and configuring the hardware platform. Also, it proposes a generic flexible and scalable abstraction of hardware based on several modules, in order to simplify the device management in the control plane side. Hence, it is very easy to extend the current model in order to cover all the features proposed by the TSON device.

Figure 3-2 summarises the TSON extension performed within the Terminal device yang model. Specially 4 parameters have been defined:

- The frame size (frame-size): represents the maximum size of the ingress packets from the client side.
- The time slot size (time-slot-size): it is relative to the time that is allocated to each client port that is going to share the same output line.
- The pair of VLANs (Vlan-in, Vlan-out): The new TSON device can host 4 VLANs per line, and sometime the bidirectional communication need a pair of VLANs as presented in the integrated demonstration (cf section 5). To handle this kind of bidirectional communication, this pair of VLANs has been defined.

The Extended OpenConfig is used to build the NETCONF agent. Indeed, a NETCONF agent has been developed to get the instructions coming from the SDN controller and to translate them a file understandable by the FPGA.

```
module: openconfig-terminal-device
    +--rw terminal-device
        +--rw config
        +--rw state
        +--rw logical-channels
          |
              ...
          |
          |   +--rw tson
              |   | +--rw config
              |   | | +--rw frame-size?          uint64
              |   | | +--rw time-slot-size?      uint64
              |   | | +--rw vlan-in?             uint16
              |   | | +--rw vlan-out?            uint16
              |   | +--rw state
              |   |   +--rw frame-size?          uint64
              |   |   +--rw time-slot-size?      uint64
              |   | | +--rw vlan-in?             uint16
              |   | | +--rw vlan-out?            uint16
```

Figure 3-2: TSON OpenConfig Extension tree view.

3.2.2 NETCONF agent based on the TSON OpenConfig Extension "and NETCONF driver based on OpenConfig extended to TSON"

The NETCONF protocol has been chosen to bring more flexibility and manage easily the future different enhancements that will bring in the TSON network.

Figure 3-3 presents the general architecture of the NETCONF agent. Our NETCONF agent is based on ConfD software which is a solution for managing southbound device provided by Tail-f. ConfD implements several protocols like NETCONF or RESTCONF and data bases to register the different yang models supported or used to represent the southbound devices. It also provides the different APIs to operate with the database and perform the management tasks.

The NETCONF module in Figure 3-3 represents the implementation of ConfD where the extended OpenConfig is registered in the database. First of all, each device administrated by the NETCONF/OpenConfig agent must have an abstraction inside the database. This representation based on the extended OpenConfig permits the list the TSON device capabilities (number of ports implemented, transmission mode supported, features of each port, etc.). Figure 3-4 shows an example of the TSON representation based on the extended OpenConfig.

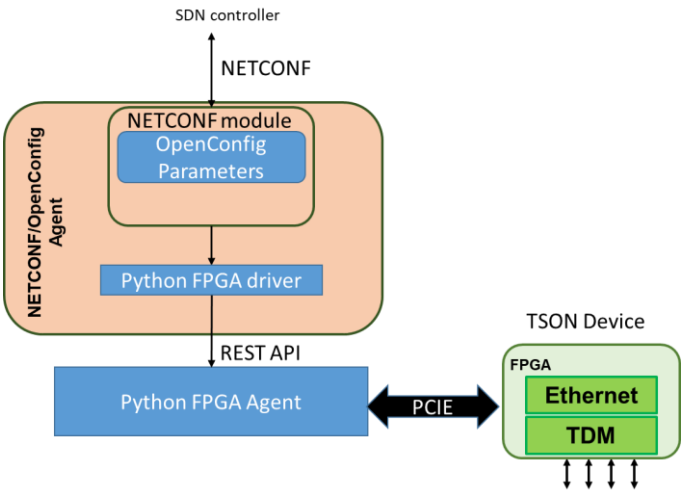

Figure 3-3. TSON Southbound Architecture.

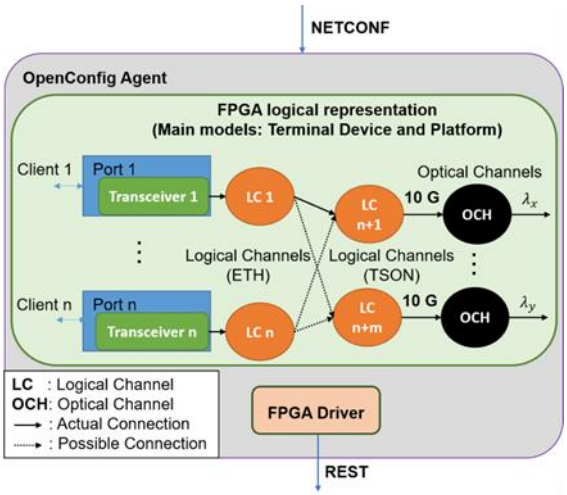

Figure 3-4. Logical abstraction of TSON node based on OpenConfig.

The Client port of the TSON device is represented by the combination of "*port"* and *"transceiver"* which are described as component in OpenConfig. In *"port"* component, we can specify all parameters relative to an optical port (date rate supported, status of the port, etc.). In *"transceiver"* component, the transceiver type is described where the wavelength, the type of the transceiver or further details relative to the optical transmitter are noticed. This combination is associated to a logical channel. These logical channels can be multiplexed/demultiplexed to the other logical channel where is implemented the transmission mode (Ethernet or TSON). If the TSON mode is activated, the different extended parameters depicted above can be configured.

Once the TSON device is defined, the NETCONF module can expose the capabilities of the node and registers it to the SDN controller. The SDN controller is able to the send the configuration parameters to the NETCONF module. The corresponding received parameters are updated in the database. A python FPGA driver has been developed to listen to the different changes corresponding to the TSON device in the database. The updated parameters are caught by the python FPGA driver, the latter formats them to a JSON file and sends it to the Python FPGA agent via REST API. Then, the FPGA agent converts these new parameters to a binary stream and sends it to the FPGA to set it up via PCIe.

3.2.3 TSON developed modules in ONOS SDN Controller

To program the TSON device through the ONOS SDN controller, a TSON application has been developed and the OpenConfig model using the NETCONF driver has also been extended. Figure 3-5 depicts the ONOS architecture to manage the TSON device.

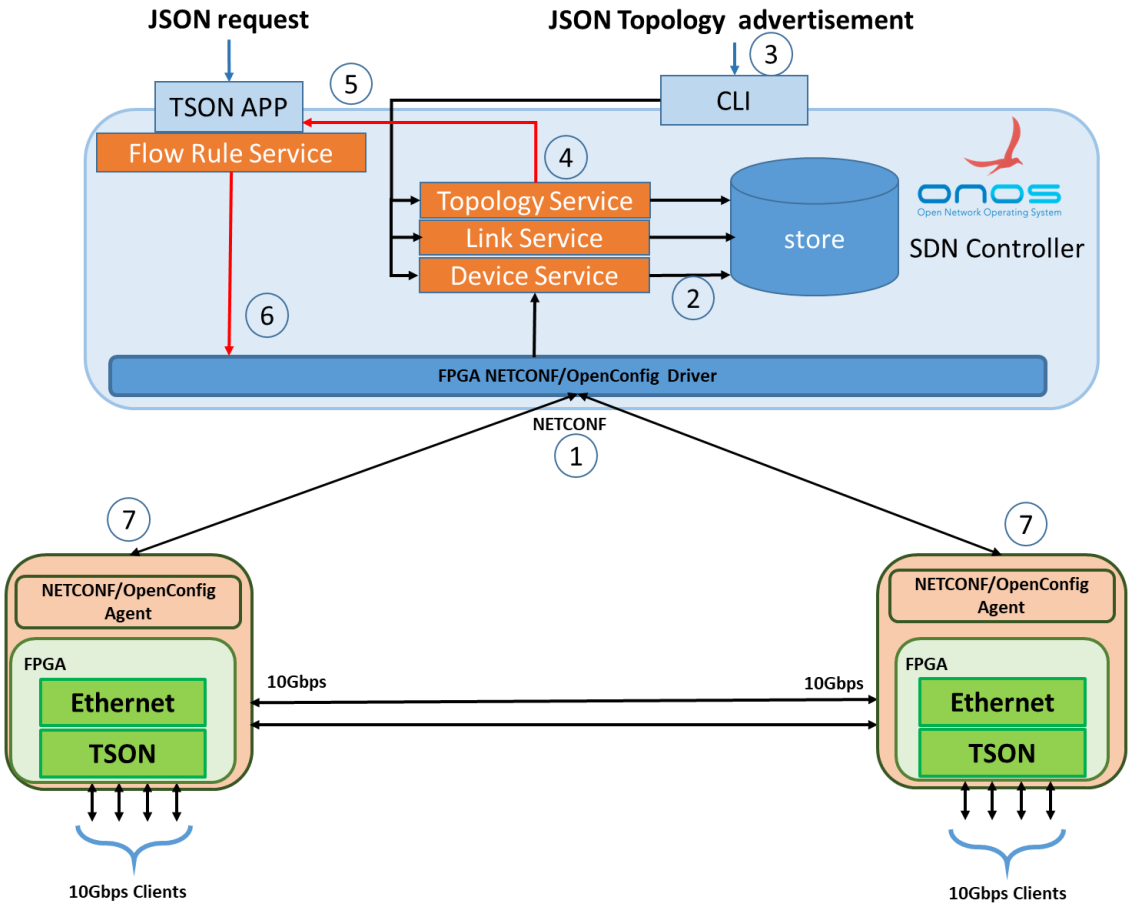

Figure 3-5: General architecture of TSON control plane.

Figure 3-5 describes the general TSON control plane architecture including the main building blocks involved to establish the end to end communication through the TSON node. The following points shows the different steps to configure the TSON based on a JSON request:

1. The different NETCONF devices are connected to the SDN controller and send it their capabilities and features.
2. The NETCONF driver through the Device Service Abstracts and registers the different TSON nodes in the Store.
3. A JSON Topology is edited to declare the different links which connect the different line ports.
4. Then, the TSON Application service is activated and can get the topology.
5. A JSON request containing for each device, the client ports to activate, their output line (Multiplex/demultiplex), the transmission mode (Ethernet/TSON) are declared.
6. The TSON APP according to the topology and the JSON request sets the appropriate parameters up based on the extended OpenConfig in order to configure the Flow rule. Once the flow rule is configurated the parameters are sent to the NETCONF driver.
7. The NETCONF driver transfers the OpenConfig data configuration to the NETCONF/OpenConfig agent which is going to process the received parameters and program the TSON device.

3.2.4 COP module implemented within ONOS

The Control Orchestrator protocol (COP) has as goal to unify the northbound transport orchestration, and is the interface used by the 5G OS to orchestrate the previsioning of end-to-end connectivity services as described in the multi-domain transport demonstrator reported in Section 5.3.

The COP API is divided into 4 services defined by yang models:

- Service topology.
- Service call.
- Service path computation.
- Service Virtual network.

In term of development, only the "*Service Topology"* (ST) and the *"service call"* (SC) have been implemented.

Two applications have been developed to handle the COP protocol, Figure 3-6 :

- COP module.
- COP – REST API.

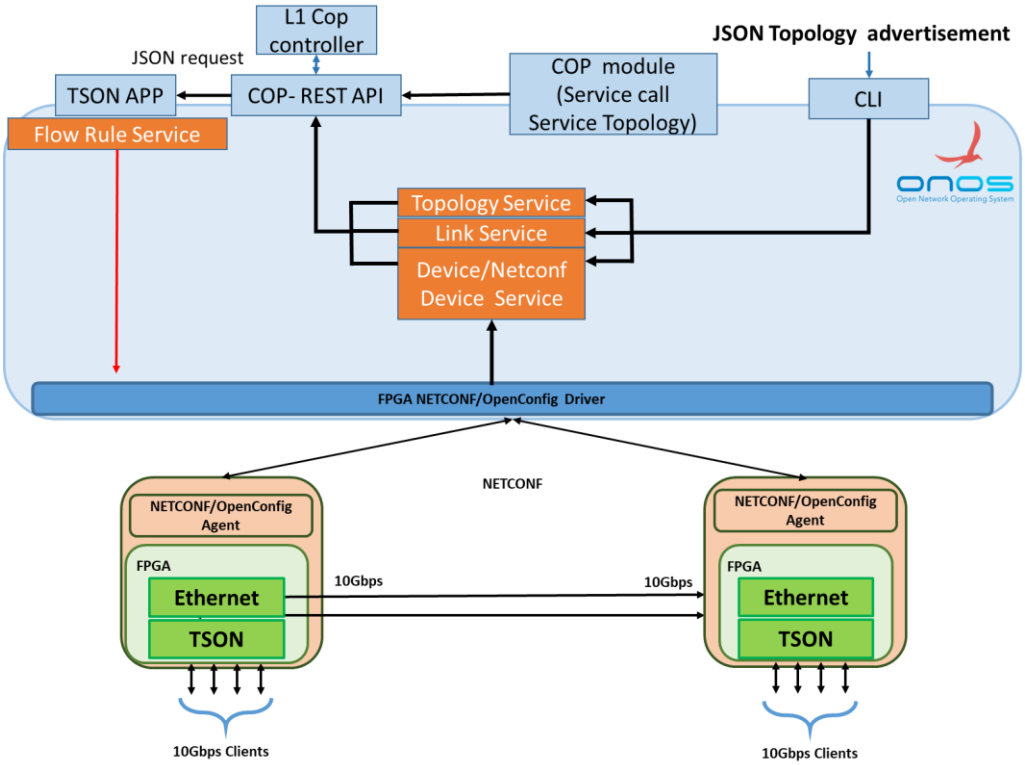

Figure 3-6: COP Implementation and integration.

The COP module generates the difference interfaces based on the yang module. These generated interfaces have been implemented in order to provide the ST and the SC. ST defines the topology by specifying nodes, edges, links which abstract the southbound domain. The SC permits to establish and end to end connection abstraction by specifying the connection requirements.

The COP-REST API is also developed to expose the COP topology formatted in JSON to the COP controller via REST. Indeed, the COP – REST API application consumes the native services in ONOS as Topology service, Link service and Device/NETCONF Device service (depending on the southbound protocols) in order to map their different resources to the COP representation based on YANG model.

In term of ST, first, the implemented cop module exposes the ST to the COP REST-API module. When the GET topology request is sent from the L1 COP server to the COP REST-API application, to get the COP topology, the application gets the information about the current devices connected to ONOS, the description of the different links used to bind the devices and the information relative to the topology. Once all the necessary information is collected, it creates a mapping of the ONOS topology information onto the COP based on the services exposed by the COP Module. Indeed, the links are mapped onto the COP edges and the devices are mapped onto the COP nodes. Then a COP topology instance is instantiated mainly composed by the lists of the nodes and edges. This topology instance is converted to JSON instance and the latter is sent as response to the COP controller Figure 3-7.

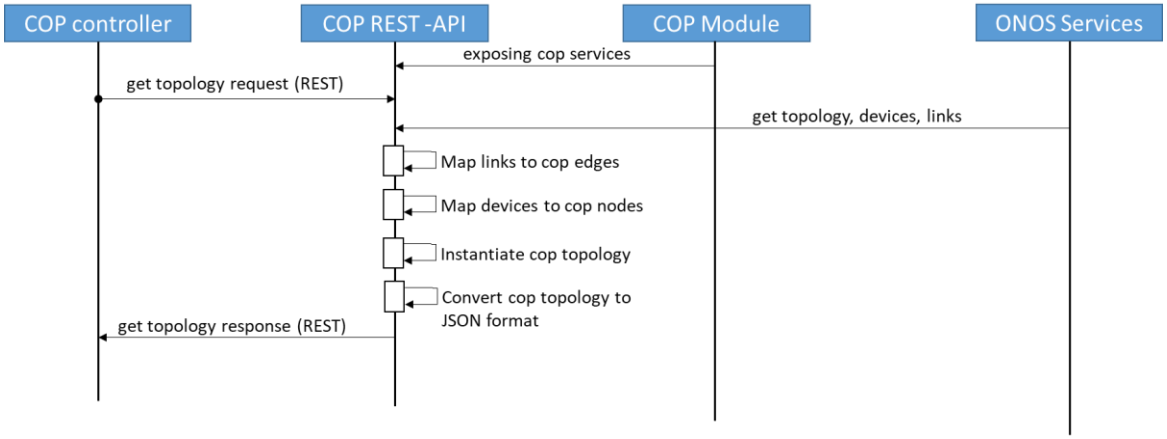

Figure 3-7. COP topology service workflow.

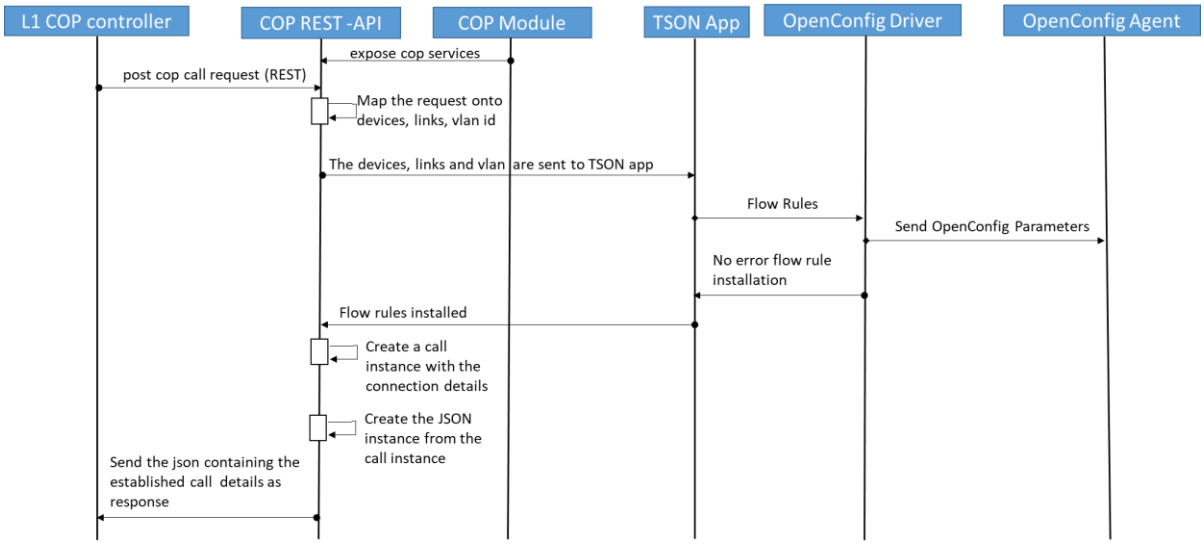

Figure 3-8. COP call service workflow.

Concerning the SC, a POST request is sent from the L1 COP controller to the REST API application to set up an end to connection according to the COP topology. The call request contains the end nodes, the connections that identify the port for each node and the VLAN tag. In our case, only two nodes are run in the TSON network. This use case reduces the complexity of network management and we can easily handle the end to end connection. In the future, when several nodes will constitute the network domain the path computation module must be implemented.

Once the call request is received, the COP REST-API application processes the mapping of the call onto the ONOS resources (devices, links). The results are sent to the TSON application which uses the

received information to send a flow rule to the extended OpenConfig driver. Then the OpenConfig driver sends the right parameters to the appropriate OpenConfig agent via NETCONF. If no error is raised when the flow rules are installed, the TSON application sends the notification ***"Flow rules installed"*** to the COP REST-API application which builds the response and formats it in JSON and then sends it to L1 COP controller (Figure 3-8).

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

4.1 Introduction

Synchronisation is a key requirement for many 5G RANs. This chapter explores how clocks can be propagated and synchronisation achieved over different wireless and wireless links, which is the goal of Technical Components 12 (Section 4.2) and 13 (Section 4.3). In addition, since transport networks are inherently heterogeneous the provision of integrated synchronisation is essential – this is a particular focus for Technical Components 14 (Section 4.4) and 15 (Section 4.5). Finally, we also report a novel technical to synchronize Remote Radio Heads over the air in Technical Component 16 (Section 4.6).

4.2 Technical Component 12: IEEE 1588 over IEEE 802.11ad

Previous evaluation of IEEE 1588 has been performed using the Blu Wireless Technology (BWT) Typhoon platform with an Intel host processor. In this section we detail additional tests to verify the accuracy of the hardware timestamping functionality of the Typhoon.

With the existing BWT's Typhoon platform, there is no way to measure the hardware timestamping accuracy directly, i.e. to compare the hardware timestamp of a PTP event packet with the 'true' time. In addition, the Typhoon platform does not have support for PPS output that could traditionally be used to compare clock synchronisation accuracy between a PTP master and slave.

As described in deliverable D4.2 [1], the BWT Typhoon platform comprises two independent 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 to as the "BH2". The latter implements the two independent IEEE 802.11ad modems, named "Hydra 0" and "Hydra 1", with independent MAC and PHYs.

The diagram in Figure 4-1 shows a high-level architecture of the Typhoon platform. Note that, in addition to the RWM6050 (with the BH2 intellectual property), the host is also attached to a PTP-capable Ethernet interface. This allows interfacing to Ethernet-based PTP clocks in the network.

The BH2 PHY hardware is capable of taking nanosecond-accurate timestamps of incoming and egressing packets. The BH2 has one system timestamping (STS) module per Hydra, and the two STS modules are independently configurable and appear as two PTP Hardware Clocks to a PTP application. However, both STS are driven from the same internal or external clock source and are therefore always syntonised. In the latest version of the Typhoon platform, the time counter for STS module uses a 440 MHz clock, giving a theoretical timestamping accuracy of ±1.13 ns.

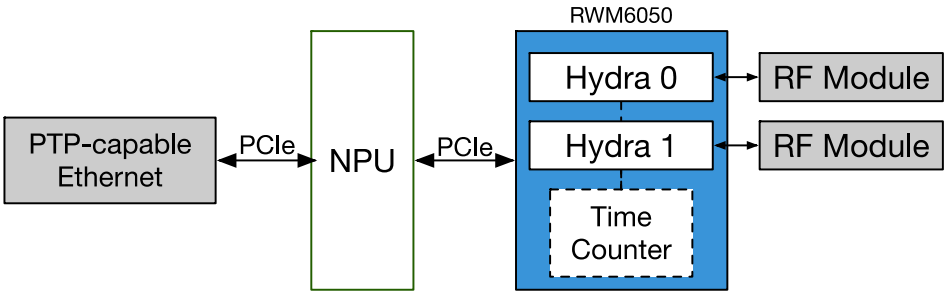

Figure 4-1: Blu Wireless Typhoon platform overview.

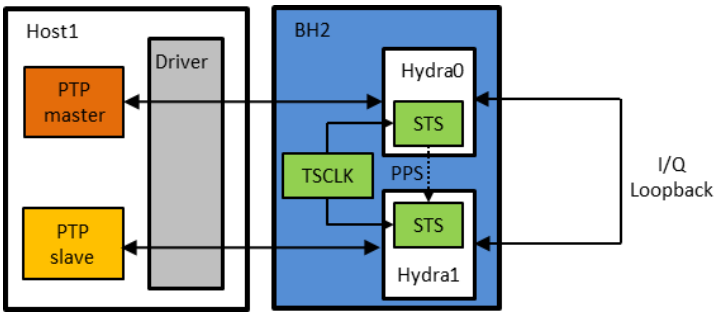

Figure 4-2: Cabled set-up with Hydra0 connected to Hydra1.

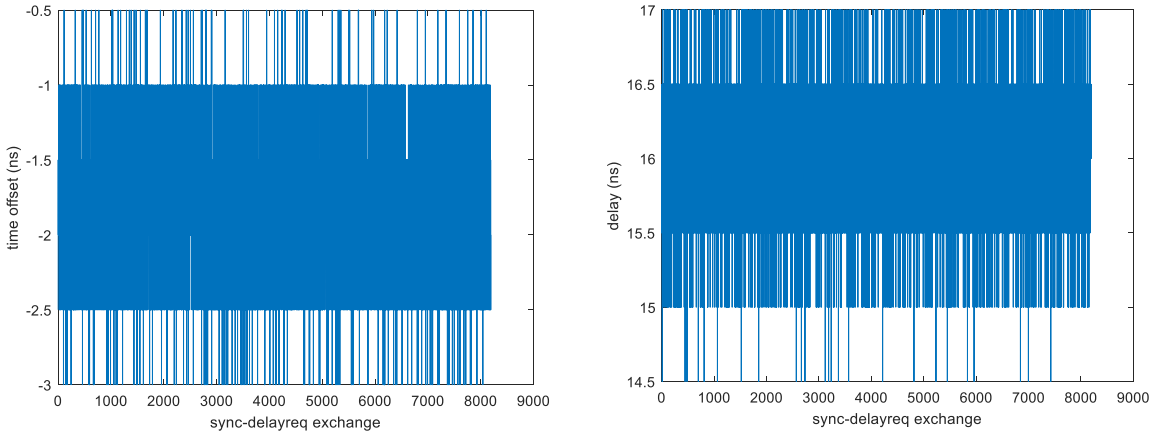

Figure 4-3: Calculated time offset (left) and delay (right) between Hydra0 and Hydra1 STS using custom BWT PTP software.

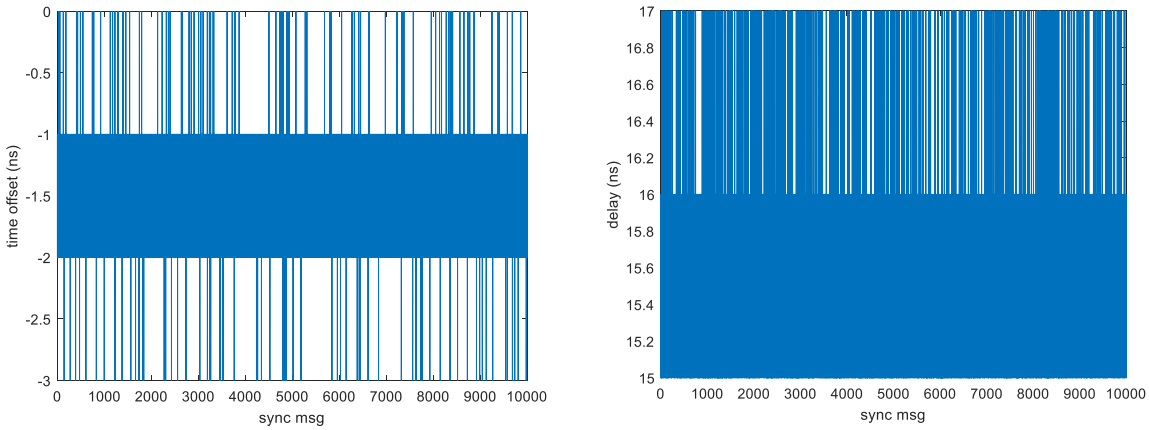

Figure 4-4: Calculated time offset (left) and delay (right) between Hydra0 and Hydra1 STS using ptp4l.

The fact that the two Hydra STS clocks are syntonised by default can be used together with PTP delay and time offset measurements obtained with PTP sync/delay request exchanges to get an indication of the variation in the BH2 timestamping accuracy. The BH2 hardware platform set up shown in Figure 4-2 is used. This consists of a single Typhoon, i.e. a single host, with Hydra 0 connected to Hydra 1 over I/Q cables. The STS clocks in Hydra0 and Hydra1 are syntonised as they use the same reference clock, and the actual time offset between Hydra0 STS and Hydra1 STS may be non-zero but should be constant. In fact, the Hydra STSs can be synchronised to within ±3 ns of each other using an internal PPS synchronisation mechanism.

Next, the BH2 timestamping can be exercised by running PTP sync/delay request exchanges between Hydra 0 and Hydra 1. The time offset and delay between the Hydra0 and Hydra1 STS can then be calculated for each PTP sync/delay request exchange. This calculation was done using a BWT written PTP master/slave software implementation in Python, and also using the open-source ptp4l application from the Linux PTP project.

When using ptp4l, two instances of the ptp4l application were used on Host1, one as a PTP master bound to Hydra0 and the other as a PTP slave bound to Hydra1. The time offset and path delay between Hydra0 and Hydra1 can then be calculated at the PTP slave. Any variation in the calculated time offset must be due solely to variations in the STS timestamping in BH2. When using the custom BWT PTP software, a similar approach was used with the master and slave applications running on Host1 and bound to Hydra0 and Hydra1, respectively.

The path delay and time offset values are calculated using the departure and arrival timestamps of the PTP Sync and Delay Request messages. Using error propagation analysis, it can be shown that the uncertainty in each individual path delay and time offset calculated from the set of 4 timestamps is equal to the uncertainty in each individual timestamp. This is only valid if a raw path delay is calculated from each timestamp tuple, and a raw time offset is then calculated from the raw delay, i.e. averaging is not used on the path delay beforehand. This can be achieved using the 'raw' mode in ptp4l. When running in slave mode, the ptp4l application will try to minimise the time offset by adjusting the frequency of the PTP Hardware Clock it is controlling using the software time counter approach of the linuxptp framework. This is not needed in this experiment, and indeed would be detrimental to the results. This frequency adjustment was disabled by modifying the BH2 driver.

The calculated time offset and path delay between Hydra0 and Hydra1 STS are shown in Figure 4-3 and Figure 4-4 for the custom BWT PTP software, and ptp4l, respectively. With the BWT PTP software, the time offset has a mean of -1.764 ns and a standard deviation of 0.487 ns, and the delay has a mean of 16.044 ns and a standard deviation of 0.486 ns. With ptp4l, the time offset has a mean of -1.505 ns and a standard deviation of 0.533 ns. The path delay has a mean of 15.802 ns and a standard deviation of 0.534 ns. As predicted from theory, the deviation of the time offset and path delay is identical. This is also consistent with the expected standard deviation in the timestamps, which can be assumed to be uniformly distributed between -1.13 ns and 1.13 ns, giving a deviation of 0.655 ns. The slight difference can be explained by the fact that the timestamps are always rounded to the nearest whole ns, and ptp4l reports time offset and delay with ns accuracy.

This experiment has confirmed that the timestamping modules in the BWT Typhoon platform perform as expected and have a timestamping accuracy within the expected range of ±1.13 ns.

4.3 Technical Component 13: IEEE 1588 over off-the-shelf IEEE 802.11ac

Most of today's Linux (operating system used in 5G-PICTURE's Sub-6 GHz platform) Wi-Fi drivers follow the *softMAC* approach. In *softMAC*, the device's Media Access Control (MAC) Sublayer Management Entity (MLME) is expected to be managed in software. In modern Linux, the *mac80211* is the wireless subsystem working as an API for those *softMAC* WLAN drivers. That is, *mac80211* is the MAC interface for upper layers, for example, for the Linux PTP Project's implementation of IEEE 1588 running in user space. However, *mac80211* does not support software timestamping nor do Wi-Fi *softMAC* drivers support hardware timestamping. Hardware timestamping support would require changes at hardware level and, therefore, this option is not contemplated due to the use of commercial off-the-shelf (COTS) devices. On the other hand, software timestamping could be enabled with software modifications both at *mac80211* and the Wi-Fi driver. Figure 4-5 shows the different hooks in the software stack that allow the implementation of (software) timestamping required by Linux PTP, which are described in the following lines.

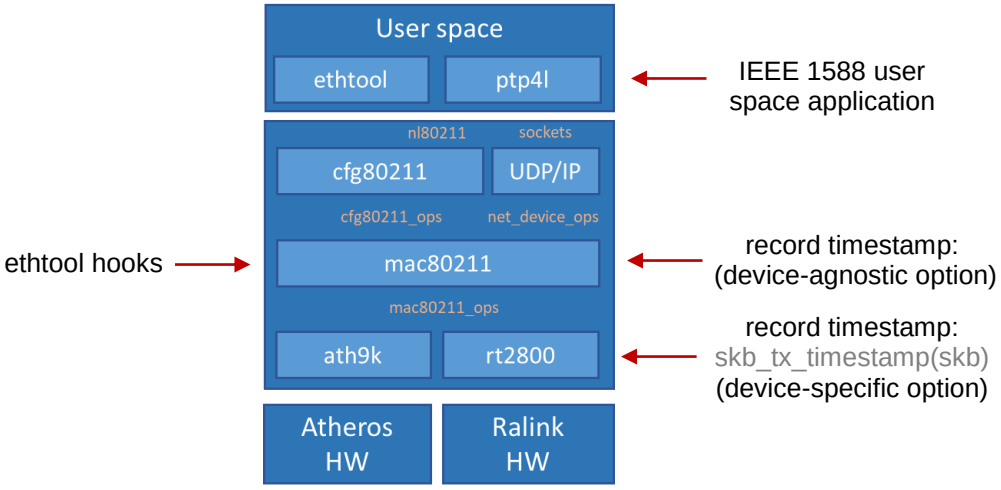

Figure 4-5: Hooks for enabling IEEE 1588 transport in Linux-based Sub-6 GHz nodes.

To enable IEEE1588 over Wi-Fi links, the following modifications are needed in Linux's Wi-Fi data path from user space to the hardware:

- *mac80211* should report its new timestamping capabilities to upper layers
  - Modify *ethtool* hooks for *cfg80211* subsystem (the configuration API for 802.11 devices in Linux, bridging user space and drivers through *mac80211*).
- *Apply the timestamp at the closest point to the hardware in the driver's transmission path. Note that this solution becomes device-dependent. Recording the timestamp at mac80211 will allow compatibility with any softMAC driver at the cost of increasing latency and jitter (i.e. losing synchronisation accuracy).*
  - **rt2x00mac_tx()** function in rt2800usb-based Wi-Fi NICs.
  - **ath9k_hw_txstart()** in ath9k-based drivers.
  - Here, we evaluate different implementation options to enable support of IEEE 1588 transport (based on Linux PTP Project's tools) over *softMAC* off-the-shelf Wi-Fi devices.

The operation of PTP over COTS IEEE 802.11 devices is compared against two benchmarks: NTP (Network Time Protocol) over an Ethernet link and PTP over an Ethernet link. The interest in NTP is due to the fact that it is the de-facto standard for the synchronization of operating systems in personal computing, while PTP over Ethernet constitutes an upper bound, unachievable in this case due to the random and unpredictable contention-based access used in IEEE 802.11.

4.3.1 Benchmark 1: NTP over Ethernet

The testbed is composed of two nodes, one of 5G-PICTURE's Sub-6 GHz node (cf. deliverable D3.1 [3]) and one desktop computer, which will play the role of NTP server and PTP master. For the benchmark measurements, both devices are connected by means of an Ethernet switch.

In the case of NTP, a third station is used to send broadcast UDP packets on the same LAN at a pace of one packet per second. Upon reception of each of those UDP packets, the two synchronized nodes record a timestamp. After one hour, the records of the two devices are compared to measure the time difference between the two timestamps recorded for each packet. The result is shown in Figure 4-6.

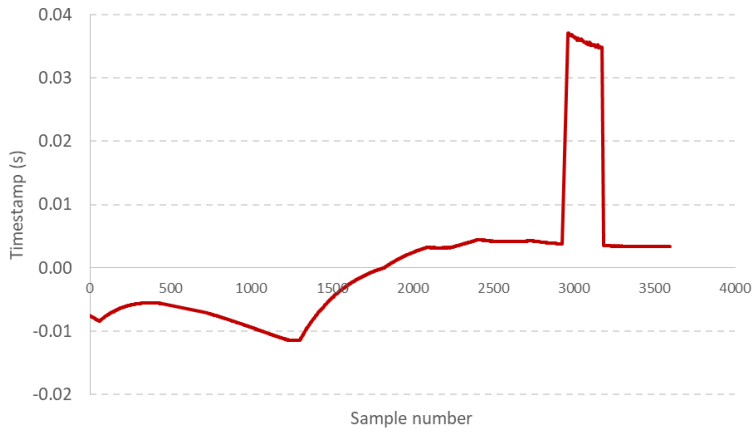

Figure 4-6: Time offset between two nodes synchronised using NTP over Ethernet.

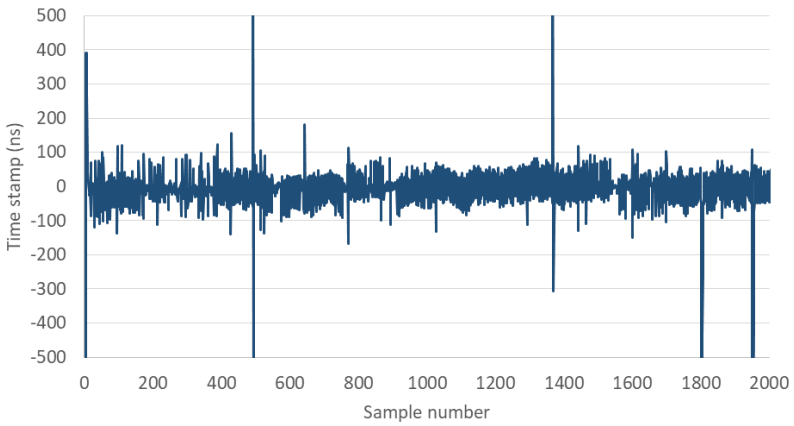

Figure 4-7: Time offset between two nodes synchronised using PTP over Ethernet (HW timestamp).

Although the difference, on average, is very small (~1 ns) due to the effect of having positive and negative offsets, if we take absolute values, the average difference grows to ~25 ms. Only 5% of the samples is below 1 ms. We also observed periods of re-synchronisation with peaks of ~1 s.

4.3.2 Benchmark 2: PTP over Ethernet

In the second benchmark, we measure the time offset from the output of the linuxptp tool (*ptp4l*) and, therefore, no third station is needed for the measurement. In a first set of experiments, we use hardware timestamping since both the Ethernet NICs and the driver support it. Figure 4-7 shows the results using hardware timestamps.

Although peaks of near 4 µs were observed, the average offset was measured below 50 ns. On the other hand, if we configure the PTP nodes to use software timestamps, the offset raises to the tens of microseconds, as shown in Figure 4-8 (note that the vertical axis here is in µs units, not ns).

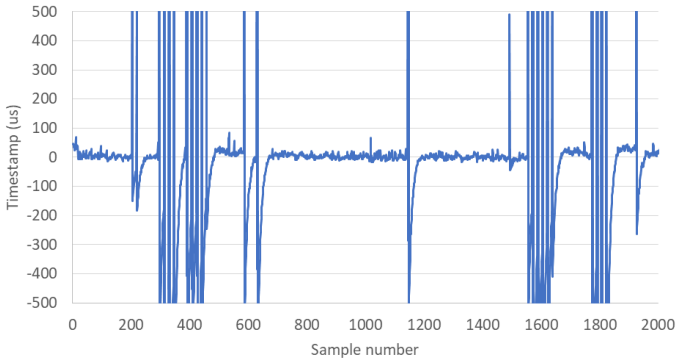

Figure 4-8: Time offset between two nodes synchronized using PTP over Ethernet (SW timestamp).

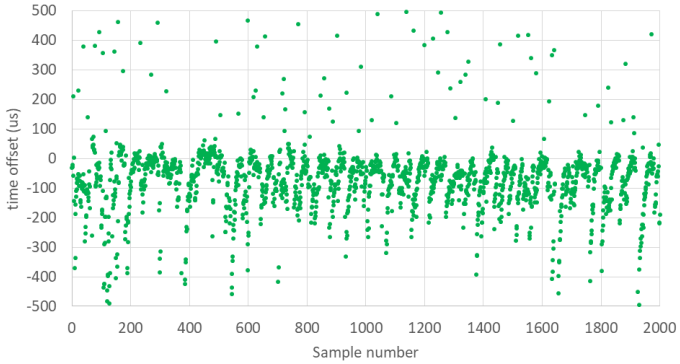

Figure 4-9: Time offset between two nodes synchronized using PTP over Wi-Fi (nightly hours).

In the case of PTP over Ethernet with SW timestamps, the average measured offset is close to 180 µs, although most of the samples (80%) were below 20 µs and several peaks were observed above 0.5 ms.

4.3.3 PTP over IEEE 802.11: measurement results

As mentioned earlier, using COTS IEEE 802.11 NICs, there is no option to perform hardware timestamps and, therefore, only software timestamps are available. Over a wireless channel, prone to transmission errors, and using a license-free spectrum (2.4 GHz band) in a public space, affected by interference, the conditions of the experiment will have a great impact on the results. Note that the software timestamp is recorded before the PTP packet is sent to the hardware transmission queue. The hardware then keeps the packet for an unpredictably variable time, depending on the contention of the channel (other transmission on the channel will make the PTP transmitter wait until the channel is idle, including a random backoff time) and the possible retransmissions needed in case of a bad reception.

The first set of measurements was carried out during night hours (2 to 9h am) to minimize the impact of interference due to the reduced activity in the surroundings of the lab. As shown in Figure 4-9, even without notable activity, the variability of the measured time offset has notably increased with respect to the benchmarks using a wired connection.

Despite the variability, the measured average offset is similar to the case of PTP over Ethernet with software timestamping (i.e. ~180 µs). However, during office hours, activity in any Wi-Fi channel is increased, affecting the medium access time of all nodes and, thus, degrading synchronization accuracy. In that scenario, the average offset raises above 600 µs. A possible remedy for this medium access issue could come with the use of QoS. IEEE 802.11 defines four different access categories, each having different access parameters (congestion window size, inter-frame spacing and transmission opportunity duration). The highest priority is given to network control/management frames

through IEEE 802.1p priority code point of 7, which is identified by a Differentiated Services Code Point (DSCP) of 48. That is, by forcing a DSCP 48 in the IP header of PTP packets (e.g. using Linux's *iptables* tool), synchronization control frames will have the highest priority. By default, PTP packets are classified as best-effort traffic (DSCP 0). Enabling QoS, we made new measurements during office hours. Results are shown in Figure 4-10, showing an improvement over the no-QoS scenario, but still worse than the behaviour observed during night hours.

In the new configuration, PTP achieves an average time offset of 400µs, which is more than enough for some applications (e.g. TDMA-based Sub-6 GHz backhaul [4]). However, it is still far from the synchronization required in order to implement 4G/5G advanced techniques such as Coordinated Multi-Point (CoMP) or carrier aggregation, which have synchronisation needs in the scale of hundreds of nanoseconds [2]. Finally, Figure 4-11 shows a comparison of the most relevant results in form of a CDF.

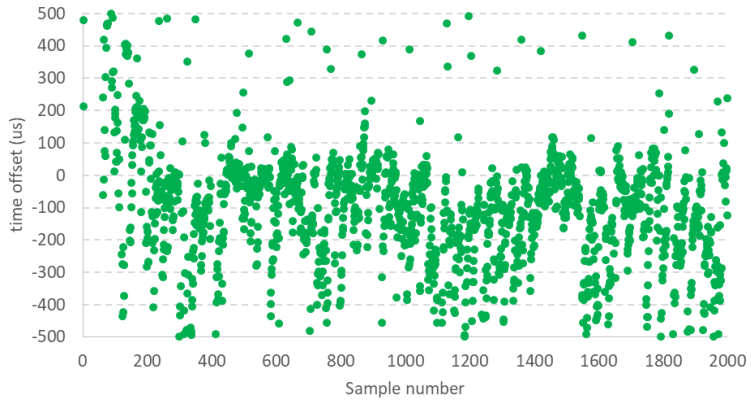

Figure 4-10: Time offset between two nodes synchronised using PTP over Wi-Fi (office hours).

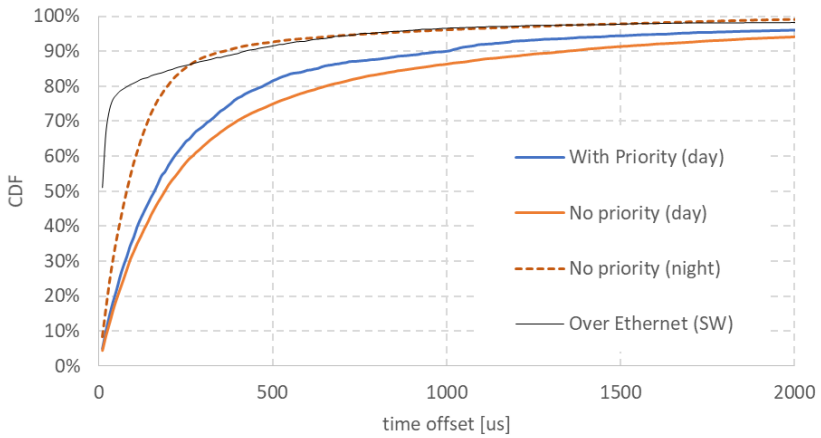

Figure 4-11: CDF of the time offset measured in PTP over Wi-Fi links.

4.4 Technical Component 14: Heterogeneous synchronisation test bed

4.4.1 NITOS testbed to support synchronisation activities

The NITOS facility is providing the IEEE 1588 PTP as a service installed on all the NITOS nodes, used for minimizing any clock skew between the interconnected nodes, regardless of the (wireless) network technology used. It relies on the ptp4l Unix service [5], which implements the Boundary Clock (BC) and Ordinary Clock (OC) of the protocol. Based on the fact that the service can run either with software or hardware timestamping, users of the testbed can synchronise the nodes of the testbed with a sub-ms accuracy, regardless of the network technology that they might be using. In the case that hardware

timestamping is supported (depending on the network interface card used), the time stamping policy is applied at the network driver level, using the *SIOCSHWTSTAMP* flag [6] in the *ioctl* calls.

The PTP execution at the application layer for the synchronisation of a network segment that is hosting VNFs in NITOS is independent of the underlying transport technologies used by this segment. This happens because the packets are encapsulated as Ethernet packets after leaving or before being pushed to each technology-specific driver. The use of PTP is critical depending on the applications hosted over the testbed. Prior solutions (e.g. NTP) are not able to provide sub-millisecond accuracy, a fact that is a prerequisite for several services that need to be tightly synced. In the following paragraphs, we provide an indicative proof of concept experiment that illustrates the adoption of PTP in NITOS.

4.4.2 Demonstration of the PTP installation in NITOS using OAI

To evaluate the efficiency of the PTP installation in NITOS, we conduct a live migration experiment when using the disaggregated implementation of the LTE stack provided by the OAI platform. As the OAI implements the LTE stack, it is expected that within 1 ms it is able to change modes from transmitting to receiving. This is the minimum sub-frame time defined for LTE. These timing requirements need to be respected by all the protocols running in the stack (and therefore change their mode from transmit to receive and vice-versa per each ms) [7]. When disaggregating the base station stack, these rules need to be respected at the Remote Unit level, apart from the higher layers that can operate in a looser manner. To this aim, we experiment with two types of splits in the stack, based on the technical components presented in deliverable D4.1 [2]: the 3GPP option 2 split (between the PDCP and RLC layers of the stack, and 3GPP option 7-2 split (inside the lower physical layer). We expect that for the former split, which has very slack timing requirements between the CU and DUs of the network we do not need any synchronisation across the different entities, whereas for the latter sub-ms timing synchronization is needed. We use virtual machines in order to orchestrate the operation of the LTE network as VNFs, deployed through OSM and OpenStack installation in NITOS. The NITOS nodes are used as the compute infrastructure in the experiment, and we use different technologies for the joint fronhaul/backhaul transport network between the access network and the core network VNFs (Ethernet, Wi-Fi, mmWave).

Figure 4-12 illustrates the deployed setup in the NITOS testbed. The OAI disaggregated stack is running as VNFs, interconnected through a heterogeneous transport network (Wi-Fi/mmWave or Ethernet). The compute nodes hosting the VNFs are synchronised with the *ptp4l* service, in with a sub-ms accuracy. We attach a wireless client to the LTE network and perform a live migration of the cloud-located part of the base station.

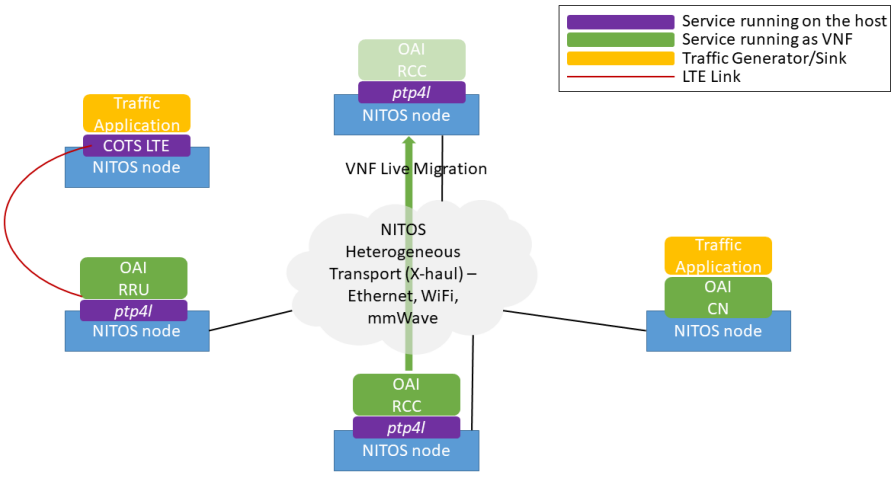

Figure 4-12: Experimental topology for PTP validation in NITOS.

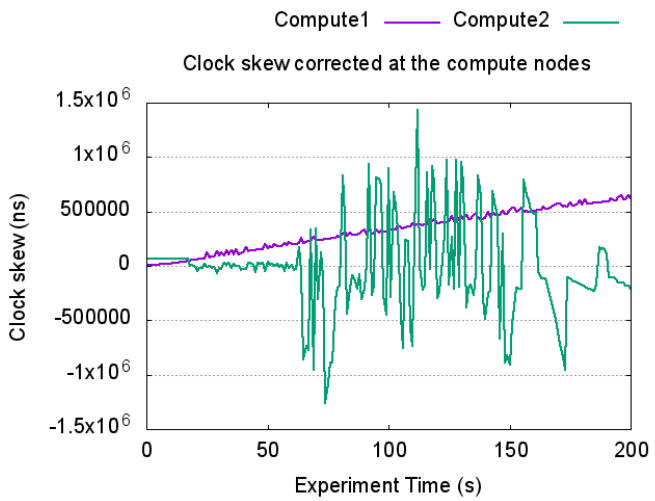

Figure 4-13: Clock corrections at the compute nodes.

Depending on the implemented split (Option 2 or 7-2), we observe different behaviours: in the case of completely unsynchronised hosts, the live migration of the CU (3GPP option 2 split) is successful. This is due to the fact that the higher layers implemented in the CU (PDCP and above) have lower demands on the latency over the fronthaul interface and the communication with the DU is asynchronous. On the other side, when experimenting with Option 7-2 split and we migrate the RCC part of the base station, for non-synchronized compute nodes, the network operation is broken. When introducing the *ptp4l* synchronisation framework, the live migration is performed in a seamless manner, and without affecting the UE connection to the network. In Figure 4-13 we depict the clock skew (and subsequent corrections) that took place during our experiment between the compute nodes that host our VNFs. The measured times for the completion of the live migration of the VNFs ranges between 195 - 237 seconds.

4.5 Technical Component 15: Synchronisation harmonizer

Synchronisation has been the basic requirement to provide a wide variety of services, e.g., localisation. To achieve that, several approaches have been adopted in the literature. One of the common approaches is the Precise Timing Protocol (PTP) specified in IEEE 1588v2 [8], which can potentially provide sub-µs synchronisation accuracies. However, this method is appropriate only for tree structure network topologies. Furthermore, the Grandmaster reference clock selection is based on a static algorithm, the Best Master Clock Algorithm (BMCA), which does not take the statistical performance of the individual clocks into consideration. Given that, it appears necessary to adopt novel approaches capable of overcoming the above mentioned downsides while reducing the end-to-end synchronization delay to approximately 130 ns, as elaborated in [9].

In this deliverable we focus on the Bayesian methods for synchronisation. In particular, synchronisation with the aid of Bayesian filtering [10], Kalman filtering [11], and Factor Graphs (FG) [12] [13] are presented and discussed. The aim is to elaborate on the concept of the synchronization harmonizer raised in [9] and cover server aspects thereof, e.g. data base unit and function unit.

4.5.1 System Model

The idea of synchronisation harmonizer has been presented and described in [9]. That is, as shown in Figure 4-14, to provide a centralised unit being capable of storing the data regarding network characteristics, e.g. hardware properties of each node. Moreover, the harmonizer is able to decide on the method of synchronisation. That is, choosing the algorithm utilized to synchronise the nodes with each other given the topology of the network and the hardware capabilities of each node. Since the properties of the harmonizer have been extensively discussed in [9], we, in this deliverable, focus on

the developing of the synchronisation algorithms and will investigate the necessity of the presence of the data storage units in the harmonizer while describing the algorithms.

In particular, we are considering a two-level synchronisation harmonizer approach. On the higher level, the role of the harmonizer is to decide on the synchronisation method based on network capabilities and service requests from the northbound interface (e.g. a NETCONF server). On the lower level, the method itself is implemented locally on each node of the network. For the control and parameter exchange between the two levels, the southbound interface can be implemented through, e.g. the IEEE 1588v2 management client. Furthermore, we introduce the mesh network in Figure 4-15 as an exemplary case where the synchronization of the network elements is to be carried out.

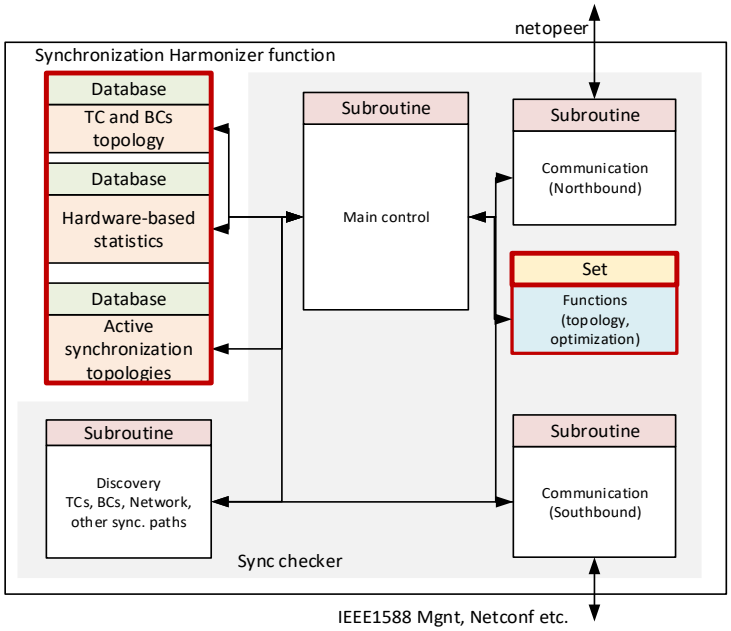

Figure 4-14: Architecture of the synchronisation harmonizer.

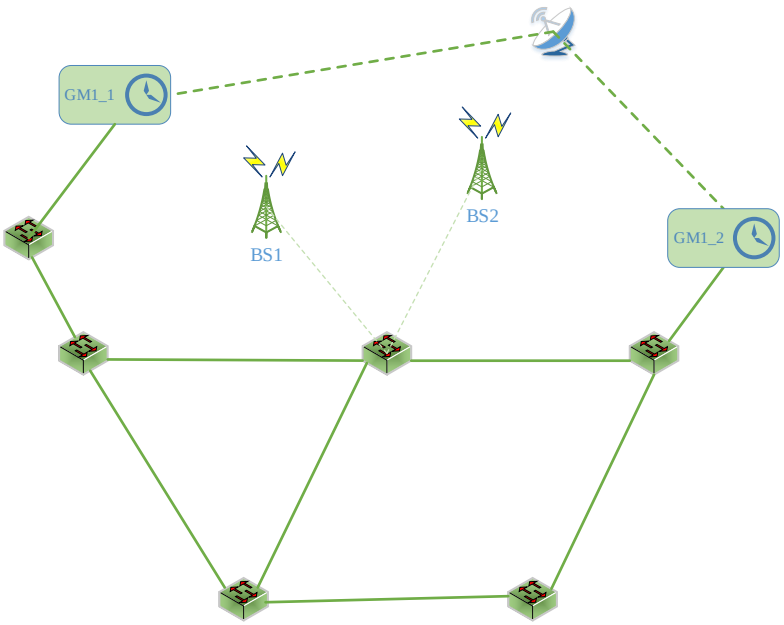

Figure 4-15: Exemplary mesh network.

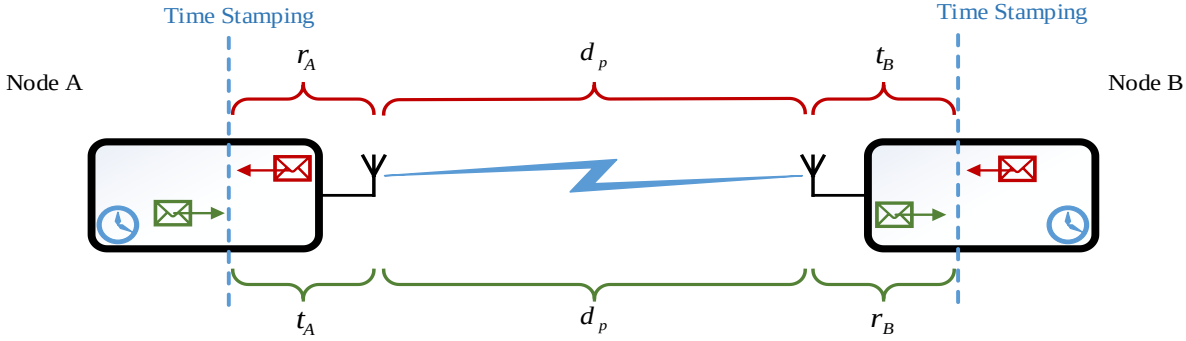

Figure 4-16: Decomposition of the offset between two adjacent nodes.

Each node i is considered to have the clock model:

$$c_i(t) = \gamma_i t + \theta_i,$$

where γ_i and θ_i are the clock skew and offset respectively. γ_i is generally random but is assumed to be constant in each synchronization interval. Furthermore, t represents the reference time of the clock. In fact, the function $c_i(t)$ determines how reference time and clock of node i are mapped onto each other. With that being explained, the goal of synchronisation is to find γ_i and θ_i for each node and apply correction such that, ideally, all the clocks show the same time as the reference time.

The offset θ_i is comprised of several components, as shown in Figure 4-17. T is the time that a packet needs to leave the transmitter after being time stamped, also known as the *egress time*, d is the delay due to the propagation, and R is the time that a packet needs to reach the time stamping point after arrival at the receiver, also known as the *ingress time*.

In general $T_K = t_A + r_B$ and $R_k = t_B + r_A$ are random variables due to several random independent processes and therefore can be assumed i.i.d. Gaussian random variables, whereas d_p is due to propagation and usually assumed to be deterministic and symmetric [12]. As in [12] and [13], we use the time stamping shown in Figure 4-17 to estimate the offset between two adjacent nodes. Thus:

$$c_j(t_{ij,2}^k) = \alpha_{ij}\left(c_i(t_{ij,1}^k) + \gamma_j(d_{ij} + T_k)\right) + \theta_j - \theta_i,$$

$$c_j(t_{ij,3}^k) = \alpha_{ij}\left(c_i(t_{ij,4}^k) - \gamma_j(d_{ij} + R_k)\right) + \theta_j - \theta_i,$$

where $\alpha_{ij} = \frac{\gamma_i}{\gamma_j}$ represents the relative clock skew. Considering that higher end clock nodes typically have a tolerance in the order of 1 to 2 ppb, the skew can generally be considered close to 1. Furthermore, the term $d_p + T_k$ and $d_p + R_k$ are (based on measurement results in [13]) expected to be of low value. Therefore, we can adopt the approximation [15]:

$$\gamma_j(d_{ij} + T_k) \approx d_{ij} + T_k,$$

$$\gamma_j(d_{ij} + R_k) \approx d_{ij} + R_k,$$

and, consequently, the above equations turn into

$$c_j(t_{ij,2}^k) = \alpha_{ij}\left(c_i(t_{ij,1}^k) + (d_{ij} + T_k)\right) + \theta_j - \theta_i,$$

$$c_j(t_{ij,3}^k) = \alpha_{ij}\left(c_i(t_{ij,4}^k) - (d_{ij} + R_k)\right) + \theta_j - \theta_i.$$

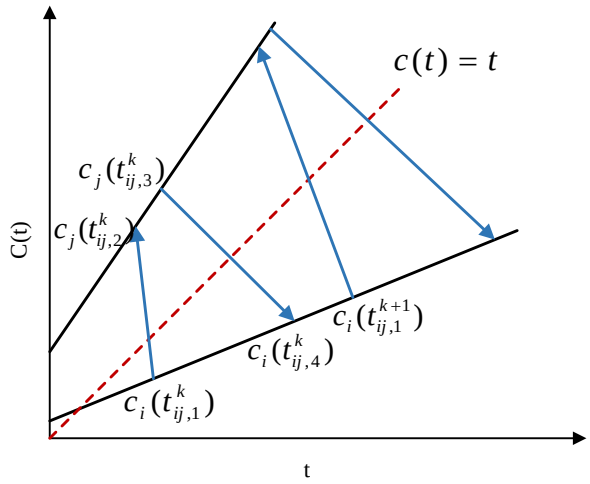

Figure 4-17: Hardware-timestamped message exchange between two adjacent nodes.

Nevertheless $\alpha_{ij} = \frac{\gamma_i}{\gamma_j} \approx 1$ does not hold since the value of recorded time stamps $c_i(t^k_{ij,1})$ and $c_i(t^k_{ij,4})$ can be large and therefore even the small amount of change in α_{ij} could lead to a huge difference in their multiplication and consequently in the estimation of the clock offsets. Summing up the above equations leads to:

$$c_j(t^k_{ij,2}) + c_j(t^k_{ij,3}) = \alpha_{ij}\left(c_i(t^k_{ij,1}) + c_i(t^k_{ij,4})\right) + 2(\theta_j - \theta_i) + \alpha_{ij}(T_k - R_k)$$

For the sake of simplicity, we do not calculate T and R mathematically but rather rely on measurement results and assume that their parameters are static. Furthermore, we define the combined metric $Z_n = T_k - R_k$ since distinguishing between T and R is not of interest in this work.

The last equation states that for a given d_{ij} the relation between the set of time stamps, $\boldsymbol{c}_{i\to j}$ and $\boldsymbol{c}_{j\to i}$, and the offset parameters, θ_i and θ_j is as follows:

$$p(c_{ij}|\theta_i, \theta_j;\ d_{ij}) = \left(\frac{1}{\sqrt{2\pi}\alpha_{ij}\sigma}\right)^N \exp\left(-\frac{\|\boldsymbol{c}_{i\to j} - \alpha_{ij}\boldsymbol{c}_{j\to i} - 2(\theta_j - \theta_i).\mathbf{1}\|^2}{2\alpha_{ij}^2\sigma^2}\right),$$

where

$$\boldsymbol{c}_{i\to j} = [c_j(t^1_{ij,2}) + c_j(t^1_{ij,3}), \cdots, c_j(t^k_{ij,2}) + c_j(t^k_{ij,3})]$$

$$\boldsymbol{c}_{j\to i}, = [c_i(t^1_{ij,1}) + c_i(t^1_{ij,4}), \cdots, c_i(t^k_{ij,1}) + c_i(t^k_{ij,4})]$$

Having defined the above conditional distribution, the joint posterior distribution is given by [12]:

$$f_i(\theta_i) = \int p(\theta_i, \theta_j|c_{ij})\, d\theta_j \ \propto \ \int p(c_{ij}|\theta_i, \theta_j)p(\theta_i)p(\theta_j)d\theta_j$$

Generalizing the above equation for the whole network, we can calculate the marginal for each node enabling them to apply the desired corrections to their clocks. Below we introduce an efficient algorithm based on probabilistic graphical models to efficiently calculate the marginal for each node.

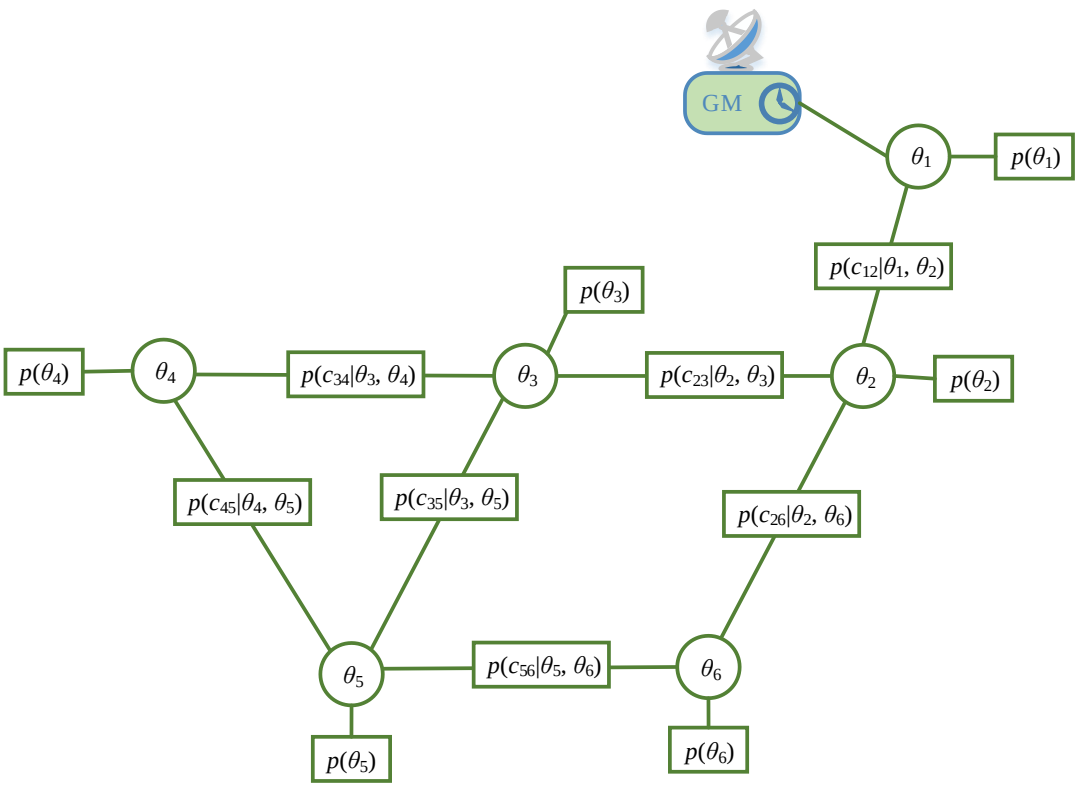

Figure 4-18: Factor graph corresponding to the exemplary network.

4.5.2 Factor Graphs and Inference

Factor Graphs (FG) are bipartite graphs used to represent the factorisation of probability distribution functions (PDFs). A FG comprises of a number of nodes, each denoting a variable (as in Figure 4-18 and several factor nodes, each being a function of their neighbor variables. In particular, the factorisation and graph structure in FGs can alleviate the computation load, e.g., that of marginal distribution through sum-product algorithm [14], or most probable state calculated using max-product algorithm. Moreover, FGs can preserve more information about the form of the distribution than either a Belief Network or a Markov Network (or Chain Graph) can do alone [16].

Employing FGs and drawing on the idea in [12][13] we construct the graphical model in Figure 4-18. With the harmonizer having the global view of the network and giving each node the relevant information about their neighboring nodes (e.g. Z so that each node can calculate the joint probability with its neighbor nodes), the network can be synchronized by each node receiving other nodes' opinion about its clock status. This manner of inference based on exchanging opinions is called Belief Propagation (BP) in literature. The following subsection is dedicated to BP and its properties.

4.5.2.1 Belief Propagation

Belief Propagation is a technique for exact inference of marginal for singly connected distributions (and near exact approximation of marginal when the graph is loopy). The algorithm is purely local meaning that each nodes cares only about its neighboring nodes and, therefore, does not take into account the global view of the network [16]. In brief, the goal here is to calculate the posterior probability of each node $p_i(\theta_i)$ in an efficient manner. The messages being passed among the neighboring nodes (both factor nodes and variable nodes) are shown in Figure 4-19. For the sake of simplicity we denote the $p(c_{ij}|\theta_i, \theta_j)$ with p_{ij}.

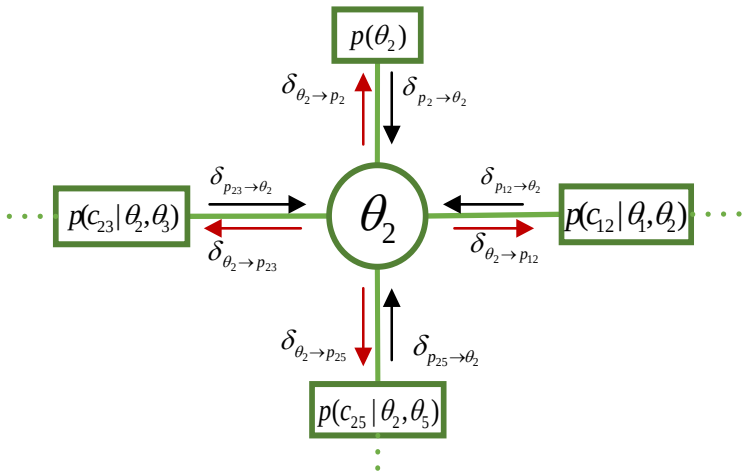

Figure 4-19: Belief propagation in the factor graph corresponding to the exemplary network.

The message from a variable vertex θ_i to a factor vertex p_{ij} is then calculated as

$$\delta_{\theta_i \to p_{ij_0}} = \prod_{p_{ij} \in \{ne(\theta_i) \setminus p_{ij_0}\}} \delta_{p_{ij} \to \theta_i}$$

Likewise, the message passed from a factor vertex p_{ij} to a variable vertex θ_i is given by

$$\delta_{p_{ij} \to \theta_i} = \int p_{ij} \delta_{\theta_j \to p_{ij}} d\theta_j$$

After several rounds of message passing using the above calculation the algorithm converges and each nodes can compute its posterior distribution. In Figure 4-19 for instance, the posterior of θ_2 is proportional to

$$p(\theta_2) \propto \prod_{p_{ij} \in \{ne(\theta_2)\}} \delta_{p_{ij} \to \theta_i} = \delta_{p_{12} \to \theta_2} \delta_{p_{23} \to \theta_2} \delta_{p_{25} \to \theta_2} \delta_{p_2 \to \theta_2}$$

The proof of convergence has been thoroughly discussed in [12][13][16].

4.5.3 State-of-the-Art Bayesian Algorithms

Bayesian approaches have been extensively used in literature to perform synchronisation between nodes [10][11]. They rely mainly on maintaining a prior knowledge of their own local clock and the revision thereof when new measurements are available. In this section, we will shortly introduce the basics of Bayesian filtering and its application in synchronisation, where Bayesian filtering reduces to Kalman Filtering (KF).

4.5.3.1 Bayesian Filtering

Let us imagine that the PDF of x_l in Figure 4-20 is to be computed. Mathematically it can be translated to:

$$p(x_l|Obs) = \int p(x_0, x_1, \cdots, x_n, x_l|Obs) dX,$$

where Obs is the vector of observations and X is a vector containing $x_0, x_1, \cdots, x_n$. After several mathematical manipulation [17], employing Bayes rule, and assuming Markov property:

$$p(x_l|Obs) = \frac{1}{p(Z)} \int p(x_0) \left[\prod_{i=1}^{n} p(x_i|x_{i-1}) p(o_i|x_i) \right] p(x_l|x_n) dx_n$$

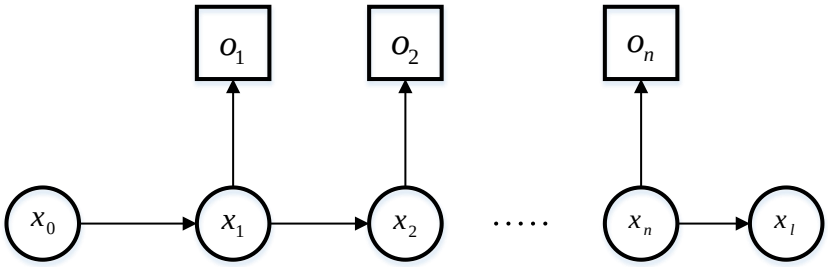

Figure 4-20: Bayesian network.

Replacing l with $n+1$ in the equation above leads to

$$p(x_{n+1}|o_n) = \frac{1}{p(Z)} \int p(x_n|Obs_{n-1}) p(o_n|x_n) p(x_{n+1}|x_n) dx_n$$

In the rich literature available on Bayesian filtering, the terms $p(x_n|Obs_{n-1})$, $p(o_n|x_n)$, and $p(x_{n+1}|o_n)$ are often referred to *a-prior*, *observation*, and *prediction step*, respectively.

It can be shown [17] that, when the distribution in the above equations follow the Gaussian distribution, the KF state and update equations can be derived from the above Bayesian filter equations. In the following subsection, we briefly introduce KF and later on we explain its application in synchronisation.

4.5.3.2 Kalman Filtering

The standard IEEE 1588 is well-known for achieving synchronisation. To this end, the PTP protocol, which is based on hardware time stamping, is employed. However, due to several error sources [11], there exists some degree of uncertainty in time stamping leading to imprecise synchronisation. As carried out in [10][11], KF can be utilized to combine the prior knowledge about the clock model with the measurement results of PTP to obtain a posterior belief about the clock offset and further apply the corresponding appropriate corrections. The state equations of KF can be given by

$$\boldsymbol{x}_{n+1} = \boldsymbol{A}\boldsymbol{x}_n + \boldsymbol{B}\boldsymbol{u}_n + \boldsymbol{\omega}_n$$

$$\boldsymbol{o} = \boldsymbol{H}\tilde{\boldsymbol{x}} + \boldsymbol{\kappa}$$

To clarify how KF is applied in the context of synchronisation, we expand the above equations as follows

$$\begin{bmatrix} \theta_{n+1} \\ \gamma_{n+1} \end{bmatrix} = \begin{bmatrix} 1 & \Delta T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \theta_n \\ \gamma_n \end{bmatrix} + \begin{bmatrix} -1 & -\Delta T \\ 0 & -1 \end{bmatrix} \begin{bmatrix} \mu_n^\theta \\ \mu_n^\gamma \end{bmatrix} + \begin{bmatrix} \omega_n^\theta \\ \omega_n^\gamma \end{bmatrix},$$

where θ_n and γ_n denote the offset and skew in n-th synchronisation time interval with the duration ΔT, respectively. μ_n^θ and μ_n^γ represent the correction applied in n-th synchronisation time interval on clock offset and skew, respectively. Finally, ω_n^θ and ω_n^γ are random process noise for the clock offset and skew respectively. The same approach for the observation leads to

$$\boldsymbol{o} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \tilde{\theta} \\ \tilde{\gamma} \end{bmatrix} + \begin{bmatrix} \kappa^\theta \\ \kappa^\gamma \end{bmatrix},$$

where $\tilde{\theta}$ and $\tilde{\gamma}$ are the precise value of the offset and skew, respectively. κ^θ and κ^γ denote the random measurement noise (time stamping uncertainty). Furthermore, the update equations are

$$\boldsymbol{K}_{n+1} = \boldsymbol{P}_{n+1|n} \left[\boldsymbol{P}_{n+1|n} + \boldsymbol{R}\right]^{-1}$$

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_{n+1|n} + \boldsymbol{K}_{n+1} \left(\boldsymbol{o} - \boldsymbol{x}_{n+1|n}\right)$$

$$\boldsymbol{P}_{n+1} = (\boldsymbol{I} - \boldsymbol{K}_{n+1}) \boldsymbol{P}_{n+1|n}$$

where $\boldsymbol{R}$ is measurement noise covariance matrix and

$$\boldsymbol{x}_{n+1|n} = \boldsymbol{A}\boldsymbol{x}_n + \boldsymbol{B}\boldsymbol{u}_n$$

$$\boldsymbol{P}_{n+1|n} = \boldsymbol{A}\boldsymbol{P}_n\boldsymbol{A}^T + \boldsymbol{Q}\,.$$

Employing the above state and update equations, we can synchronise the network nodes equipped with IEEE 1588.

4.5.4 Results and Analysis

In this section we describe the initial verification of the above presented synchronisation harmonizer algorithms via simulation for a suitable scenario.

We consider the network in Figure 4-21 as an exemplary scenario, where a number of BSs are backhauled by a mesh network. As can be seen the BSs are appended to the mesh network in Figure 4-15 constructing tree structures only at the edge of the communication network. We conduct two sets of simulations: Firstly, synchronising the whole network based only on FG and correspondingly BP algorithm. Then, we synchronise the mesh backhauling network based on FG while the tree networks at the edge of the network are being synchronised using KF. Performance of the harmonizer in a hybrid network (where the network comprises tree structure as well as mesh structure) is then evaluated by comparing the outcome of two abovementioned scenarios. The simulation parameters are set as included in Table 4-1:

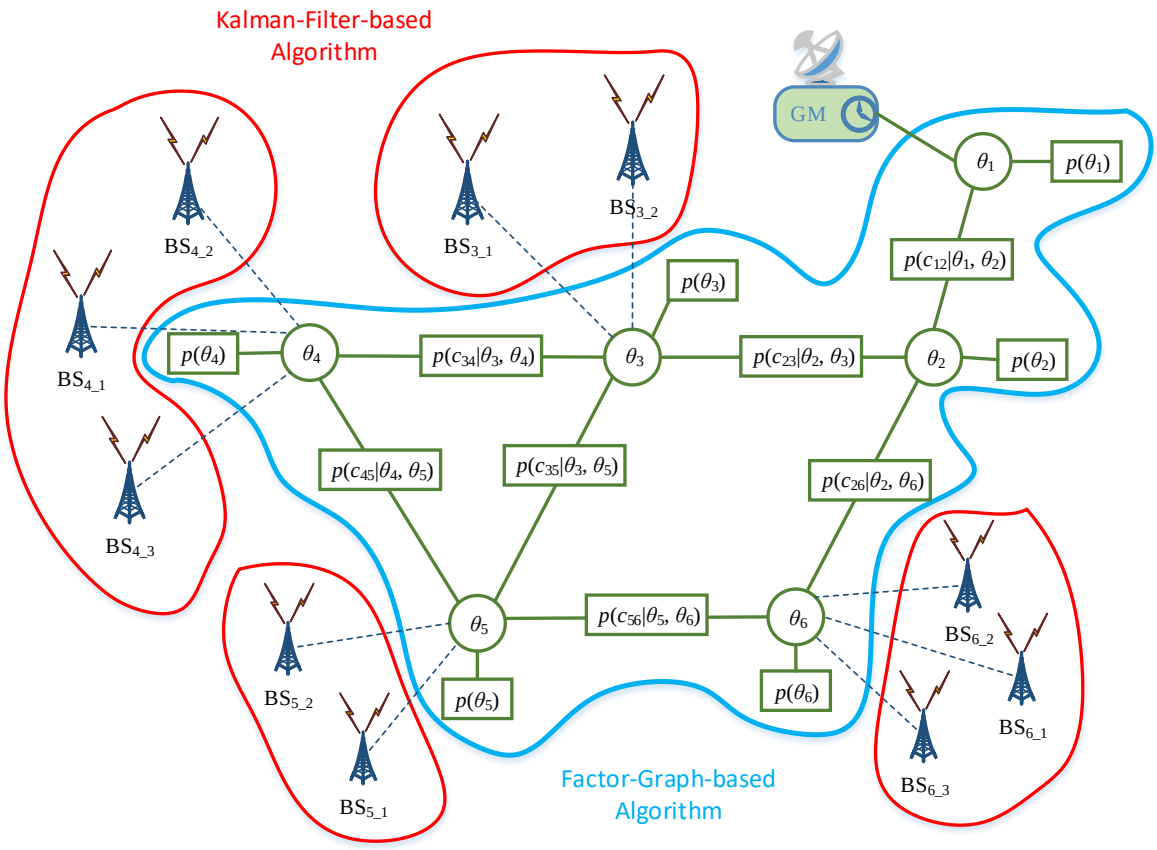

Figure 4-21: Hybrid synchronisation approach for a communication network containing mesh and tree structures.

Table 4-1: Simulation parameters.

Number of independent simulations	100
Initial random delays	[-25, 25] ns
Standard deviation of $\boldsymbol{T_k}$ and $\boldsymbol{R_k}$	4 ns
# of HW timestamped message exchange rounds for BP	10
Delay between each pair of nodes	300 ns
Initial PDF of the offset for each node	uniform

A) Belief Propagation (BP) only

Figure 4-22 represents the clock offset error estimation as well as its standard deviation for the scenario in Figure 4-21, where the BP has been applied to the whole network including the part with tree structure. As can be seen, the BP converges after 4 iterations with achieving synchronisation accuracy very close to the Grand Master (GM) node, with a 2-3 ns offset error. The convergence is guaranteed for the networks with at least one GM [13][18]. However, when the network contains loops, the value that BP converges to is considered to be approximate [14]. When there is no GM in the network, the nodes are still able to agree on a common local time among them, although not necessarily same as the reference time. Furthermore, the certainty in offset estimation can be measured by its standard deviation for each node. In this sense, BP is also showing an excellent performance as can be observed in the lower subfigures, of approximately 1 ns. In fact, the results in this simulation setup reveal the potential performance of BP in synchronization of the communication networks. However, the synch interval in which BP achieves convergence can be considered as its potential pitfall, what is attempted to be overcome in this work by introducing the hybrid network. That is, to lower the complexity of the BP by excluding particular nodes while applying BP on the nodes where high precision synchronisation is essential.

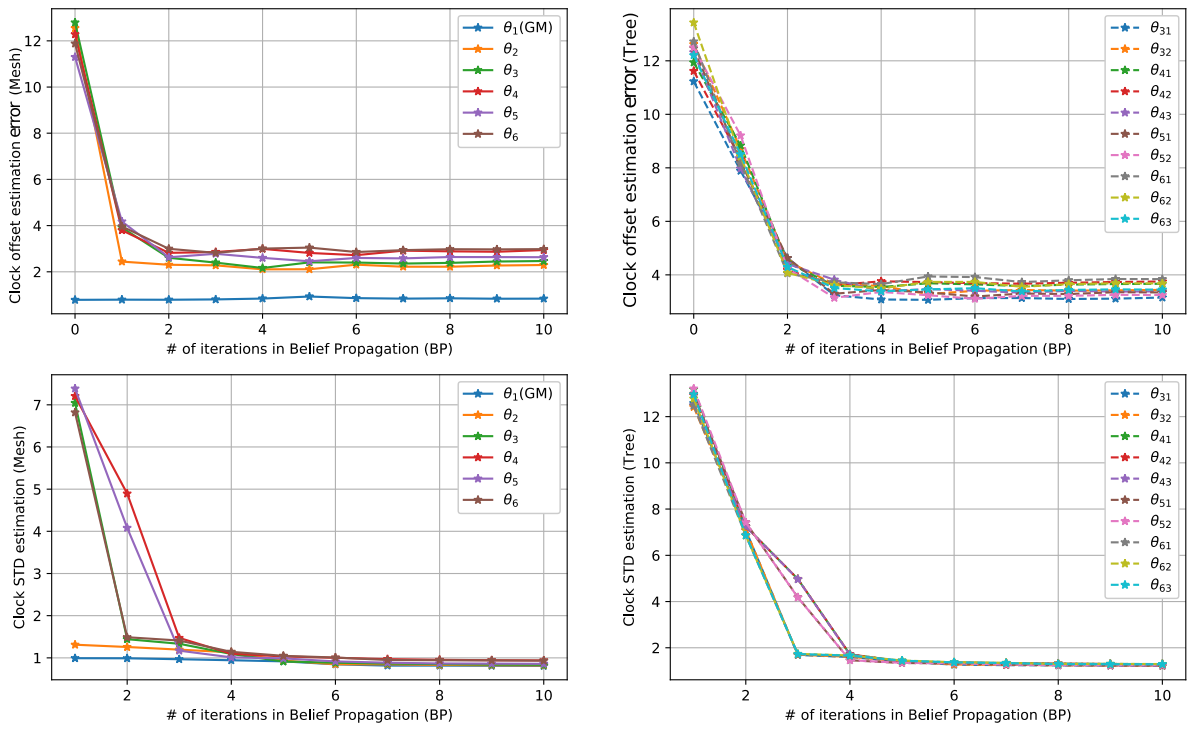

Figure 4-22: BP applied to the whole network including the tree structure part.

B) Hybrid approach with Belief Propagation (BP) and Kalman Filtering (KF)

Figure 4-23 shows the clock offset error estimation as well as its standard deviation for the scenario in Figure 4-21, where BP has only been applied to the mesh structure (the backhauling network) and KF is employed to synchronize the nodes with the tree structure at the edge of the network (leaves). As can be observed, the performance slightly deteriorates compared to the case where BP was applied on the entire network. However, we note that the KL estimation updates are significantly faster than that of BP. In fact, BP begins only when the nodes have already conducted several rounds of HW timestamped message exchange as in Figure 4-17 (in order to obtain the required statistics, namely the conditional probability distributions) whereas KF updates the estimation after each round of message exchange. In other words, since the KF is faster and runs independently (does not need any other information from the other parts of the network as BP needs), is able to conduct more iterations, thereby further decreasing the standard deviation shown in lower right subfigure of Figure 4-23.

In a nutshell, simulation results indicate that BP can be of great potential in high precision network synchronisation. Nevertheless, despite the excellent performance, complexity can cause trouble by prolonging the synch period. In particular, the time needed for the nodes to communicate to each other and pass messages can lead to deterioration in accuracy of synchronization. As a solution, hybrid approaches can be taken to alleviate the complexity. That is, applying BP on the critical parts of the network (e.g. the backhauling part who is responsible for distributing the clock to the edges) to achieve as high a precision as possible. Moreover, faster algorithms, e.g. KF, can be readily employed on the edges of the network where frequent synchronisation is required for numerous applications such as localization.

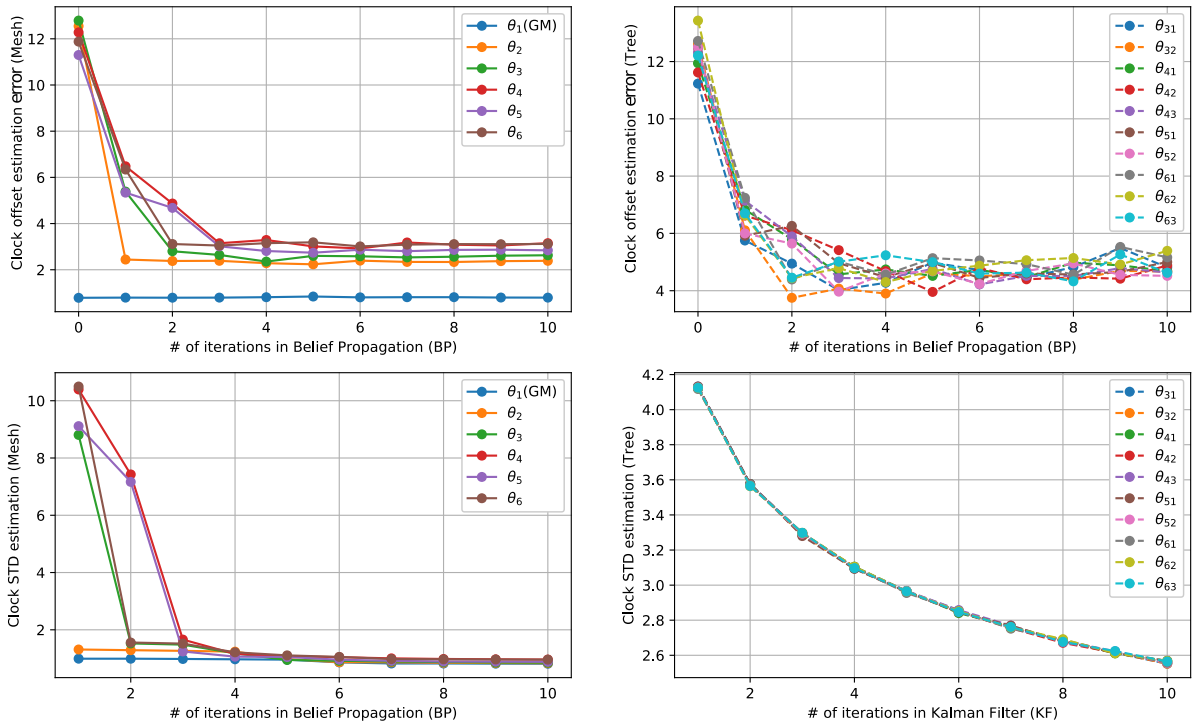

Figure 4-23: BP applied on the mesh structure and KF applied on the tree structure.

4.5.5 Conclusion & future works

We presented two algorithms for the synchronisation of communication networks, each extensively discussed and shown to have their up- and down- sides. One based on Factor Graphs able to achieve extremely accurate synchronisation with high complexity (additional message passing as well as computational costs) while the other can deliver strong performance in tree structure network. Later on,

we combined them to reduce the complexity of the BP while preventing the synchronisation quality from declining. Simulation results show that the proposed hybrid network can achieve high precision synchronisation with the lower complexity.

We only dealt with phase synchronisation here, however frequency synchronisation cannot be ignored while designing a sustainable synchronisation algorithm. As future work, one could consider incorporating frequency synchronisation, or alternatively, skew estimation into algorithm to further ameliorate the synchronisation quality.

Another objective is to perform joint synchronisation and localisation, both directly intertwined with each other. In particular, localisation with e.g. centimetre level accuracy would require high precision (and frequent) synchronisation in the local level which we believe can be achieved by further developing the methods presented in this work.

4.6 Technical Component 16: Over-the-air synchronisation for fronthaul networks

Massive multiple input multiple output (mMIMO) is one of the most promising wireless physical layer technologies to address the massive capacity requirement demanded by 5G systems. Massive MIMO exploits the use of large antenna arrays at the base station (BS) to simultaneously serve multiple users through spatial multiplexing over a channel.

Distributed mMIMO or distributed antenna system (DAS) with spatially separated antennas is considered for improving indoor coverage with a not so large number of antennas [19]. Distributed multi-user MIMO (MU-MIMO) unifies small cells and mMIMO approaches. A DAS is formed by coordinating a large number of remote radio units (RRUs), distributed over a certain coverage region, through a wired backhaul network connected to a central server. One of the biggest challenges in such distributed massive MIMO networks is the synchronisation of the RRUs.

The two common methods for radio synchronisation are the time-based synchronisation using a periodic pulse per second (PPS) and trigger-based/signal synchronisation based on a shared trigger architecture. The periodic PPS signal is used for establishing a common time base among the radios. However, the timed-based synchronisation would require each time the system is started up a calibration procedure, controlled by a host controller, to establish a common time along RRUs. Moreover, the latency introduced by the clock cycled prevents the immediate triggering of any process. Therefore, we opt for a trigger-based synchronisation as explained in the following.

4.6.1 System Architecture

EUR deployed a cloud radio access network (C-RAN) using OAI software and inexpensive commodity hardware, as shown in Figure 4-24. The testbed consists of the following 3 main entities: (i) the remote radio unit (RRU) which is a radio transceiver and contains the RF processing circuitry; (ii) the radio aggregation unit (RAU) which connects multiple RRUs to a baseband unit (BBU), and serves as a data processing unit; (iii) the radio cloud centre (RCC), which is responsible for the centralised baseband processing and controls multiple RAUs.

A set of 20 RRUs is deployed on the ceilings of the corridors on levels -3 and -4 of the Eurecom building. The RRUs on each floor are connected by Gbit Ethernet to a switch which are, in turn, connected to a central server over optical 20 Gbit Ethernet. An additional high power commercial remote radio head (RRH) is connected to the C-RAN server through a common public radio interface (CPRI) gateway. A frequency reference unit outputs ten high-precision 10 MHz frequency reference outputs on each floor. The RRUs consist of an UP board from Intel, a B200 mini from Ettus Research, an RF frontend designed by Eurecom and Power over Ethernet (PoE) technology, as shown in Figure 4-25.

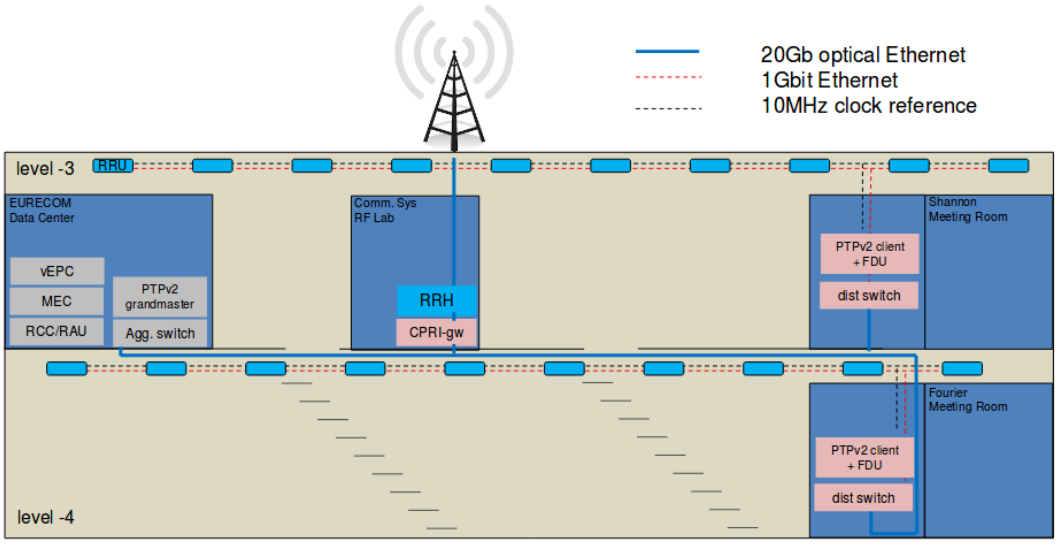

Figure 4-24: OpenAirInterface 5G DAS testbed.

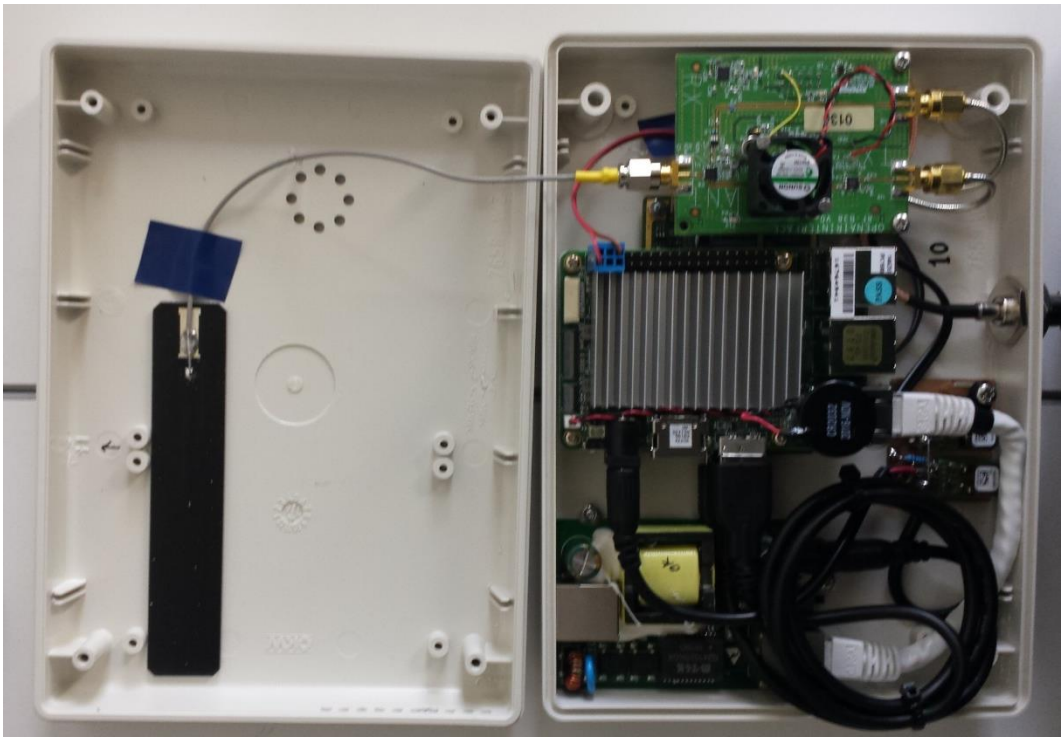

Figure 4-25: Custom Eurecom indoor RRUs.

4.6.2 Synchronisation and Calibration

There are three levels of synchronisation: (i) time synchronisation to ensure that the frames are aligned between the different RRUs up to within a sample, (ii) frequency synchronisation to ensure that the RRUs stay synchronised in time and phase and (iii) phase synchronisation to enable coherent transmission and precoding. Our system is based on LTE TDD configuration 1, which has two UL, two DL, and one special sub-frame every 5 ms.

4.6.3 Time synchronisation

Time across all RRUs must be synchronised to stay within the accuracy of one sample of the A/D and D/A converters.

Time synchronization is achieved by using over-the-air trigger-based synchronisation using a “master-slave” protocol, where one RRU acts as the master and the other RRUs synchronizing to it, similar as an UE would synchronize to the network. However, the primary synchronisation sequence (PSS) used for UE synchronisation would not provide the required accuracy as it only occupies ~1MHz. Therefore, we have added a demodulation reference symbol (DMRS) in OFDM symbol 3 of the special sub-frame 1, just after the PSS. As soon as the initial sync is done the frames are aligned and the slave RRUs start to connect to the RAU. When the RAU knows that slave is running it sends a resynchronisation command to the slave RRU to change its frame number to the right one.

4.6.3.1 Frequency Synchronisation

In order to have all the RRUs form a single DAS, we have to provide a reference for frequency and time synchronisation. A shared 10 MHz oven-controlled crystal oscillator (OCXO) reference provides frequency disciplining for the internal voltage-controlled oscillator (VCO) used to generate the system local oscillator (LO) and A/D and D/A channels, both of which must be synchronised across the entire RRU array.

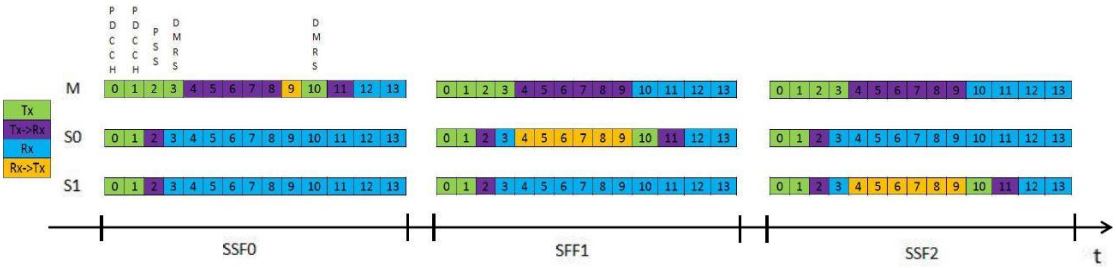

Figure 4-26. Synchronisation-Calibration Framework.

Sharing the common 10 MHz reference among RRUs allows a phase-coherent LO to be synchronized using a fractional-N frequency approach. During synthesis, as the reference is divided, the phase may be locked on either rising or falling edges producing a constant but arbitrary phase offset on each channel. Due to a poor PLL design on the B200 USRPs we realized that the phase does not lock, rendering our system beamforming-incapable. To deal with this phase incoherence we disabled the Voltage Controlled Temperature Compensated Crystal Oscillator (VCTCXO) at each RRU and we replaced it with a frequency multiplier component that generates a 40 MHz signal which is directly fed into the RF chip of the B200.

4.6.3.2 Phase Synchronisation

Beamforming places additional requirements on the system. In addition to sample time and sample clock alignment, the system must maintain a known phase relationship between each RF chain. However, because each radio has an independent synthesizer circuit (PLL-VCO) for both Tx and Rx, the phase can be considered phase-coherent but not phase-aligned. Through periodic calibration, alignment can be achieved by digitally adjusting the real and imaginary signal component I-Q-phase.

In order to calibrate the C-RAN testbed we need to collect channel measurements between the master and the slave RRUs. This is achieved by using the framework shown in Figure 4-26. In this example, there are three RRUs, one master RRU “M” and two slave RRUs “S0” and “S1”. In the first special sub-frame (SSF) in a TDD configuration 1, frame symbols 3 and 10 are reserved DMRS symbols. The slave RRUs sacrifice symbol 2 in order to switch from Tx to Rx mode, so that PSS is omitted and only the first two PDCCH symbols are transmitted in the DL. At each special sub-frame 1 every 10ms, only one RRU transmits the calibration symbol. If the RRU which is in transmit mode is a master RRU, then all

the active slave RRUs receive, decode and estimate the DL channel estimates at symbol 10. On the other hand, if a slave RRU transmits the calibration symbol, only the master RRU collects and estimates the UL channel estimates. Thus, a bidirectional calibration symbol exchange for a pair of RRUs takes up to 20ms. We assign a number at each RRU, *tag*, such that each RRU enables its transmit mode if and only if *frame* mod *p* = *tag*, where *frame = {0, 1, …, 1023}* is the frame number and *p* is the number of active RRUs in the testbed. In contrast to the frequency with which the master RRU transmits the calibration symbol, the synchronization symbol is broadcasted to the slave RRUs every 10ms.

After acquiring channel estimates between the master and slave RRUs we use the reciprocity calibration method described below.

4.6.4 Over-the-air Reciprocity Calibration

TDD reciprocity calibration and RRU synchronization are the two key factors to enable distributed multi-user MIMO. They compensate asymmetric Tx/Rx paths as well as unknown phase offsets between RRUs. This allows the eNB to obtain reliable DL Channel State Information based on UL channel estimates in the considered TDD systems.

A number of reciprocity calibration algorithms are considered in the literature as shown in Figure 4-27 (left side, top-to-bottom):

- Argos [20] considers bi-directional transmissions between the reference antenna and the other antennas.
- Rogalin *et al.* [21]] consider bidirectional transmissions between each pair of antenna elements.
- Avalanche [22] uses a scheme where calibrated antenna elements calibrate the uncalibrated ones group-wise, i.e., first one antenna calibrates with the first calibrated one, a group of two calibrates with the new reference antenna group, and so on.

Our work is based on a recently developed framework for fast calibration generalising all these methods [23]. In this framework, bi-directional transmissions of pilots are performed between RRU groups as shown in Figure 4-27 (right side). When a group of RRUs transmits, all other groups are considered in receiving mode. This includes varying group sizes and a possibly arbitrary number of channel uses. Therefore, all of the above reciprocity calibration algorithms can be used.

4.6.5 Preliminary Experimental Results

RCC and RRU run OAI on a 10 MHz LTE channel. The DL/UL channel estimates extracted from the DMRS calibration symbols are sent via packetised I/Q samples to the RAU over the IF4.5 fronthaul protocol on commodity Ethernet. IF4.5 corresponds to the split-point at the input (TX) and output (RX) of the OFDM symbol generator (i.e. frequency-domain signals). According to NGFI, IF4 is “Resource mapping and IFFT” and “FFT and Resource demapping”. IF4p5 therefore refers to simply compressed transmitted or received resource elements in the usable channel band.

As the time synchronisation happens in two steps by first acquiring frame synchronization and second synchronizing the frame number via the fronthaul. As this can happen every frame, this process takes only 20-40 ms and is thus very quick compared to the base station start.

The RAU receives inter-RRU reference signal measurements and deduces the needed calibration information to form a distributed MIMO transmitter. The system has always to be under calibrated status, therefore, calibration procedures need to be repetitively performed. Through real-time measurements, we determine the frequency with which we have to repeat these procedures without interrupting the main system operations.

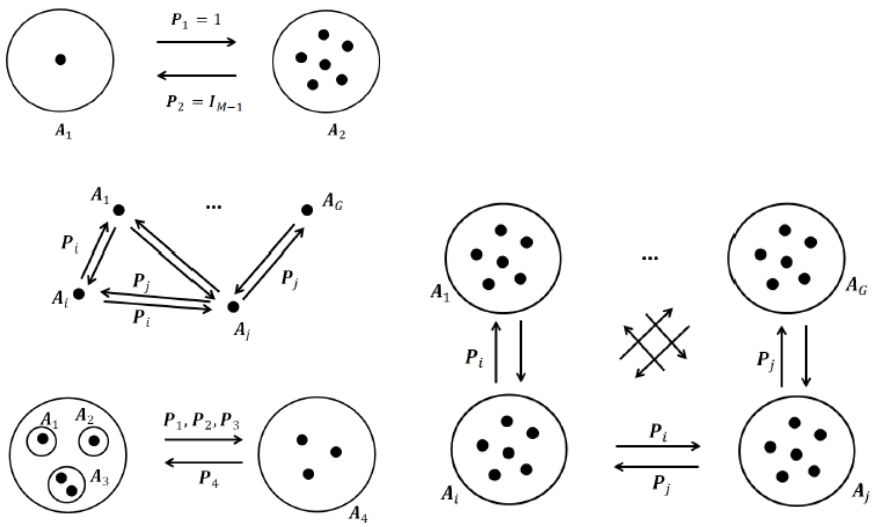

Figure 4-27. The existing calibration methods (left, top-to-bottom: Argos, Rogalin, Avalanche) and the recently-developed framework (right).

Figure 4-28. Group partitions Interleaved, Neighbours and Random (top to bottom).

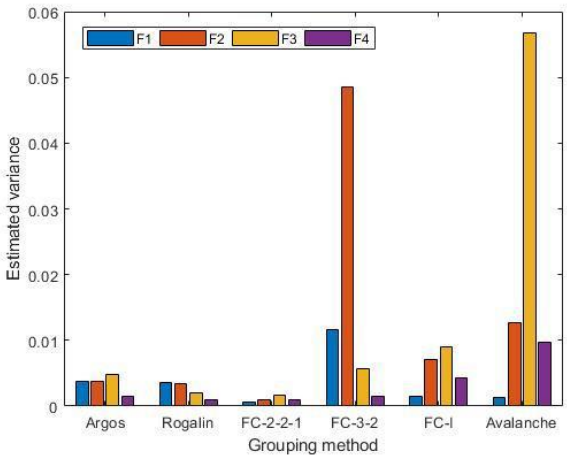

Figure 4-29: Variance of time-domain calibration coefficients for different groupings and 5 RRUs.

We consider the following calibration algorithms/groupings:

- Argos.
- Rogalin.
- Avalanche.
- FC-I (fast calibration with Avalanche-like grouping).
- FC-II (fast calibration with equally partitioned groups).

For FC-I, the RRUs (as depicted in Figure 4-24) follow the Avalanche grouping scheme. For FC-II, we consider the following group partitions for equally partitioned groups, as shown in Figure 4-28 from top to bottom: Interleaved, Neighbors, and Random.

Since the calibration coefficients can only be estimated (at a given RRU, the exact calibration coefficients cannot be known), we measure the variance of the continuously updated calibration coefficients over time. The calibration coefficients themselves are complex factors in the unit circle (unitless).

Figure 4-29 shows the variance of time-domain calibration coefficients for different grouping methods for 5 RRUs. The individual bars show the calibration coefficient variances per antenna. Argos and Rogalin perform very good, but can be time consuming as individual antennas are calibrated one by one. For the groupings with equally partitioned groups (for instance in the 2-2-1 scheme, first, one antenna synchronizes to the reference antenna, then two more antennas synchronize with this first reference group), it is visible that the calibration coefficient variance depends on the number of antennas synchronizing at the same time. Comparing the FC-I approach to Avalanche, we see that the improved calibration scheme results in lowered calibration coefficient variance. The FC-2-2-1 (FC-II) method with small RRU groups performs best in comparison to the other methods.

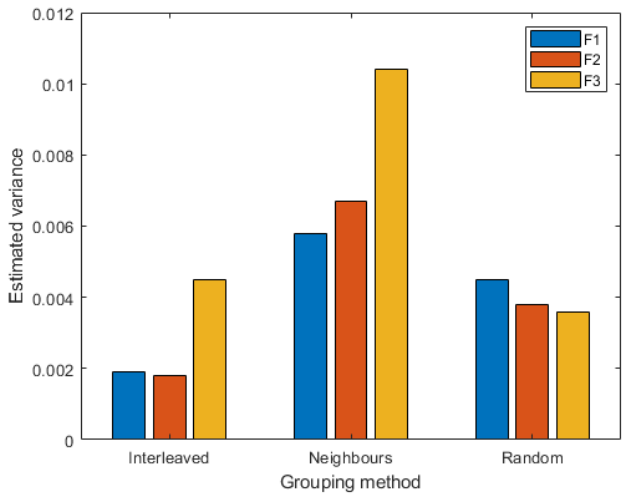

Figure 4-30. Variance of time-domain calibration coefficients for FC-II groupings and 4 RRUs.

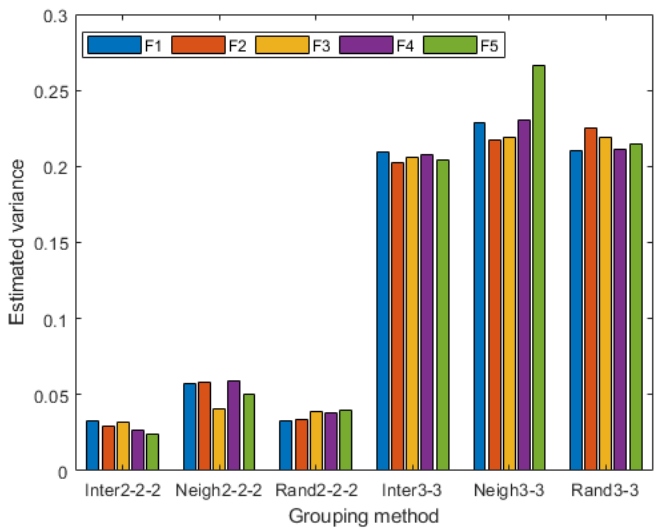

Figure 4-31: Variance of time-domain calibration coefficients for FC-II groupings and 6 RRUs.

Figure 4-30 shows the variance of time-domain calibration coefficients for the FC-II grouping schemes and 4 RRUs. In this scenario, groups synchronise with each other. It is apparent that short antenna distances between antennas of different groups improve the performance. Hence, the neighbours grouping method performs worst because antennas on the outside have a maximum distance to the reference group. Therefore, random grouping is better, since on average antennas will be closer to a reference group antenna.

Finally, Figure 4-31 also shows the variance of calibration coefficients for FC-II, but 6 RRUs. It is again observable that the interleaved grouping performs best. Furtherm0re, we see again the effect that smaller but more numerous antenna groups perform better than larger groups. Thus, there is a trade-off between group size and speed of calibration, since more calibration steps incur a higher calibration setup time.

5 Integrated Demonstrators

5.1 Introduction

In this section we describe three integration activities that validate the main design goals of WP4, namely:

- Goal 1: Demonstrate the viability of having a single transport network that can jointly transport latency sensitive fronthaul flows and backgroung traffic.
- Goal 2: Demonstrate the viability of setting up end-to-end connectivity services through a multi-domain and multi-technology transport in a matter of seconds, which is fully aligned to the 5GPPP KPI of reducing service provisioning time from 90 hours to 90 minutes.
- Goal 3: Demonstrate the viability of the Synchronization Harmonizer that is a novel centralized control function that can steer clock distribution paths in a multi-technology transport network.

Regarding Goal 1, we present in Section 5.2 a first integration activity that uses IEEE Time Sensitive Networking (TSN) for the fronthaul network. This study evaluates the performance of Ethernet TSN networks based on IEEE 802.1Qbv and IEEE 802.1Qbu for carrying real fronthaul traffic and benchmark it against Strict priority and Round Robin scheduling policies. We demonstrate that both 802.1Qbv and 802.1Qbu can be well used to protect high-priority traffic flows even in overloaded conditions.

Regarding Goal 2, in Section 5.3 we describe the design of a multi-domain transport network testbed built by joining the lab environments of four 5G-PICTURE partners (**I2CAT**, **UPB**, **UTH**, **ZN**), which we use to demonstrate the provisioning of end-to-end connectivity services using a hierarchical SDN control plane originally defined in WP5 and extended here. The results of our experiments show that end-to-end connectivity services across three different domains can be established on demand in less than 30 seconds.

Finally, regarding Goal 3, in Section 5.4 we describe an integrated synchronisation testbed featuring wired and wireless technologies, which we use to validate the ability of the Synchronisation Harmoniser Technical Component to quickly steer the paths followed by clock distribution protocols. Due to the problems acquiring the wireless synchronisation devices, only an initial description of the demonstrator is reported, and the final results will be included in deliverable D6.3.

5.2 Goal 1: Time-Sensitive Networking for 5G Fronthaul Networks

5.2.1 IEEE TSN for fronthaul networks

For the data link layer, the IEEE 802.1 Audio Video Bridging (AVB) task group work focuses on enabling isochronous and deterministic low-latency services over legacy Ethernet. However, this was intended for multimedia streaming applications. In order to widen the area of applications, the IEEE 802.1 TSN Task Group (TG) was founded. The IEEE 802.1 TSN TG focuses mainly on physical and link layer techniques to achieve guaranteed delivery of data with bounded low latency, low delay variation and low loss. A comprehensive survey covering both fixed and wireless ultra-low latency communications is presented in [24]. An overview of Ethernet and its evolution to various fields of application is provided in [25]. A categorisation of the relevant IEEE TSN standards is provided in Table 5-1.

Table 5-1: TSN Standards Overview.

Category	Standards
Time Synchronization Providing network wide precise synchronization of the clocks of all entities at Layer 2.	IEEE 802.1AS & IEEE 802.1AS-Rev (Network Timing & Synchronization)
Latency & Jitter Separating traffic into traffic classes and efficiently forwarding & queuing the frames in accordance to these traffic classes.	IEEE 802.1Qav (Credit Based Shaping) IEEE 802.1Qbv (Scheduled Traffic) IEEE 802.3br & IEEE 802.1Qbu (Frame Preemption) IEEE 802.1Qch (Cyclic Queuing) IEEE 802.1Qcr (Asynchronous Traffic Shaping)
Reliability & Redundancy Maintaining network wide integrity by ensuring path redundancy and ingress queue policing.	IEEE 802.1CB (Frame Replication & Elimination) IEEE 802.1Qca (Path Control & Reservation) IEEE 802.1Qci (Per-Stream Filtering)
Resource Management Providing dynamic discovery, configuration and monitoring of network in addition to resource allocation & registration.	IEEE 802.1Qat & IEEE 802.1Qcc (Stream Reservation) IEEE 802.1Qcp (YANG Models) IEEE 802.1CS (Link-Local Reservation)

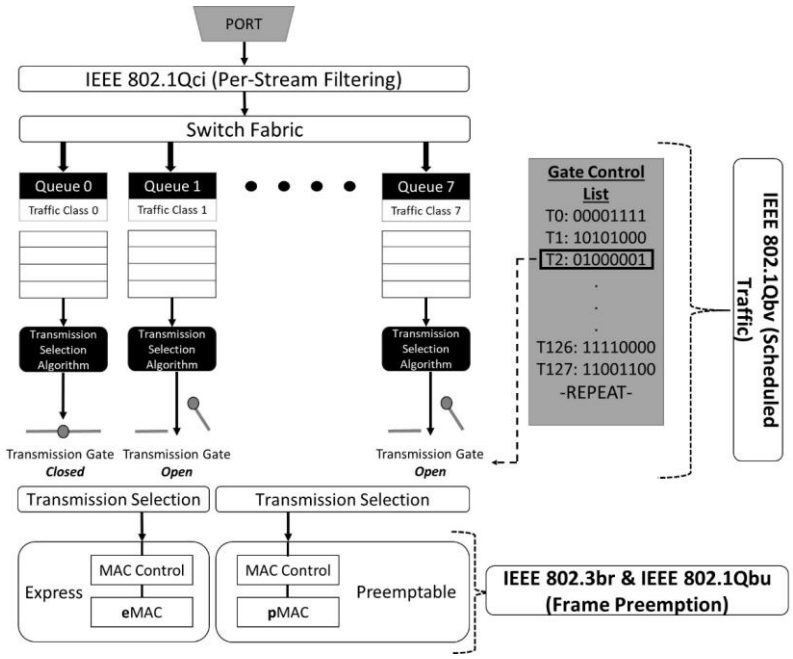

Figure 5-1: IEEE TSN Scheduled Traffic & Frame Pre-emption.

Resource management aspects of TSN are covered by amendments like 802.1Qcc, 802.1Qdd. TSN synchronization is covered in IEEE 802.1AS, ongoing work is addressed in 802.1AS-Rev. The ability to provide delay guarantees is supported by techniques like Scheduled Traffic (IEEE 802.1Qbv) and Frame Preemption (IEEE 802.3br, IEEE 802.1Qbu). These standards explain how frames belonging to a particular traffic class or having a particular priority are handled by TSN-enabled bridges. IEEE 802.1Qbv introduces a transmission gate operation for each queue as depicted in Figure 5-1. The transmission gates open/close according to a known time schedule. Scheduled Traffic ensures that the transmissions are controlled by a Gate Control List (GCL) which consists of multiple schedule entries.

For instance, in the scenario depicted in Figure 5-1, the GCL entry for T2 indicates that the gates for the queues 1 and 7 are open (1), while all other gates are closed (0). Based on these schedules, selected traffic types can be allowed to pass through to the transmission selection block which provides access to the medium. Frame Preemption on the other hand, allows the ongoing transmission of a lower priority frame to be preempted by a higher priority frame and thus ensures lower latency for high priority frames. It maps frames onto two separate MAC service interfaces namely express MAC (eMAC) and preemptable MAC (pMAC) as seen in Figure 5-1.

Express frames can thus preempt preemptable frames by either interrupting the frame transmission or by preventing the start of a pMAC frame transmission. In case of an interruption, the pMAC frame transmission resumes after the transmission of the express frame has completed. A live demo presenting preemption for low-Latency mobile X-haul 100G Ethernet was presented by ADVA in ECOC 2018.

5.2.1.1 802.1CM Profile for the Fronthaul

Based on the area of application, TSN Profiles have been specified to explain which standards, protocols, features and options should be applied for a given use-case. The existing TSN Profiles are 802.1BA for AVB networks, IEC/IEEE 60802 TSN Profile for industrial automation, P802.1DG for automotive in-vehicle Ethernet communications and IEEE 802.1CM TSN for mobile fronthaul networks. IEEE 802.1CM resulted from a collaborative effort of the CPRI and IEEE 802.1. It describes how to meet the stringent fronthaul requirements in an Ethernet-based bridged network which can support not only fronthaul traffic but also other concurrent traffic types. In 802.1CM both CPRI and eCPRI splits are supported (Class 1 and Class 2 respectively). In both cases the following data flows are considered: a) User Data; b) Control and Management Data and c) Synchronization Data. The relevant requirements (for these data flows) are defined by the CPRI Specification V7.0 and by the eCPRI Transport Network Specification V1.1 respectively. For example, for class 2 (eCPRI) the maximum end-to-end one-way latency is 100 µs for high priority user plane data traffic between eREC and eRE. The maximum tolerable Frame Loss probability for control plane data flows is 10^{-6}, and the internal time error requirements for eRE/RE synchronisation varies between 15 to 30 ns, depending on the case and category. Moreover, 802.1CM mentions the components that contribute to the worst-case latency for a single hop from a bridge to a bridge:

Input queuing delay + Interference delay (Queuing delay + Self-queuing delay) + Frame transmission delay + LAN propagation delay + Store-and-forward delay.

Further, P801.Cmde is now investigating several enhancements to the fronthaul profiles. Three types of fronthaul data flows are defined:

- High Priority Fronthaul (HPF) data, which has 100 µs maximum end-to-end one-way latency (Class 1 IQ data and Class 2 user plane data);
- Medium Priority Fronthaul (MPF) data, which has 1 ms maximum end-to-end one-way latency, (Class 2 user plane data and Class 2 control plane data);
- Low Priority Fronthaul (LPF) data, which has 100 ms maximum end-to-end one-way latency (control data for both class 1 and Class 2).

Furthermore, two profiles are specified being applicable to class 1 (CPRI) and class 2 (eCPRI):

- Profile A: exploits strict priority queuing to 52erform service differentiation between high priority traffic (User data or/and IQ data) and low priority traffic (control and management data). The maximum frame size for all traffic flows is 2000 octets (MAC Protocol Data Unit (PDU)) on each port.
- Profile B: exploits frame preemption (802.3br and 802.1Qbu) together with strict priority scheduling in order to differentiate between high priority (fronthaul traffic) and low priority

preemptable traffic (non-fronthaul traffic). The maximum frame size for fronthaul traffic is still 2000 octets (MAC PDU) on each port, while the maximum frame size for non-fronthaul traffic might vary. 802.1CM also discusses how the time synchronisation requirements can be met for precision time protocol (PTP) enabled devices satisfying for example the ITU-T G.8275.1 telecom profile and ITU-T G.8272, ITU-T G.8273 depending on the deployment case.

5.2.1.2 Performance Evaluation

The target of our performance evaluation is to investigate how system and statistical parameters affect the performance of a TSN-enabled network when carrying fronthaul traffic. The performance metrics we focus are the average forwarding latency and average delay-variation (jitter) of the high priority traffic competing with lower priority traffic and background traffic. Since eCPRI implementations are still not available, in the following experiments, we focus on NGFI split 4.5. This is a custom defined format for packetisation implemented in OAI v2019.w30 [26], and is similar to 3GPP option 7-1. Such functional split requires placing extra DFT/IDFT modules collocated with RRHs and can drastically decrease the fronthaul capacity requirement roughly by a factor of 2 [27]. The employed split transports I/Q samples in the frequency domain, i.e. after removal of the cyclic prefix and FFT, and before resource element (de-)mapping, while beamforming is not employed. Like also NGFI split 5, the capacity of split F4.5 depends only on the number of antennas carriers as well as the radio bandwidth, and thus there is no need for standard-specific processing, e.g., radio resource (de-)mapping, at the cell site.

It is thus part of the cell-related fronthaul processing (as opposed to user-specific processing), exhibiting a constant bandwidth requirement since the I/Q samples related to the full cell capacity need to be transmitted, irrespective of the cell load [27]. Since the fronthaul capacity scales with the cell sectors and antenna elements and not with the traffic load, we do not consider the traffic load. Regarding packets format from data sample pluses we refer to [28] for more details. To reduce the fronthaul bandwidth, the samples are compressed using 8-bit A-law compression originally defined in ITU-T G.711 standard with benefits of decreased throughput and round-trip-time. The packets are encapsulated and transported over UDP/IP, while annotated with 802.1Q VLAN tags to mark different flows and QoS requirements with the Priority code point. On the TSN network part we use prototype TSN switches equipped with eight 1 Gigabit ports each.

We use a traffic generator to emulate background traffic. In order to generate IEEE 802.1Qbv scheduled traffic we use Ixia traffic generator from Keysight Technologies, applying IEEE 802.1AS with hardware time-stamping for verifying time-synchronisation. When necessary we also use the *iperf* tool to generate background traffic. The forwarding latency is measured as the time interval between the start of sending the k-th packet from the RRU and the end of receiving the k-th at the BBU. Our performance evaluation is divided into two parts. In the first part, we compare the performance of TSN 802.1Qbv against the Strict Priority (SP) and Simple Round Robin (SRR) schemes for carrying scheduled and non-scheduled traffic. The results are used to develop an intuition about TSN and in particular the performance of TSN 802.1Qbv in comparison to other scheduling mechanisms. The non-scheduled traffic is generated by endpoints that are not TSN-enabled and the traffic is continuous.

Scheduled traffic on the other hand is generated by TSN enabled endpoints and the traffic is scheduled in accordance to the TSN-802.1Qbv schedules deployed on the switches. The network scenario comprises only TSN Ethernet switches, the Ixia traffic generator and *iperf* tool. In the second part we carry real fronthaul traffic over Ethernet with TSN. The network scenario additionally consists of several RRU(s) and one BBU, see Figure 5-2).

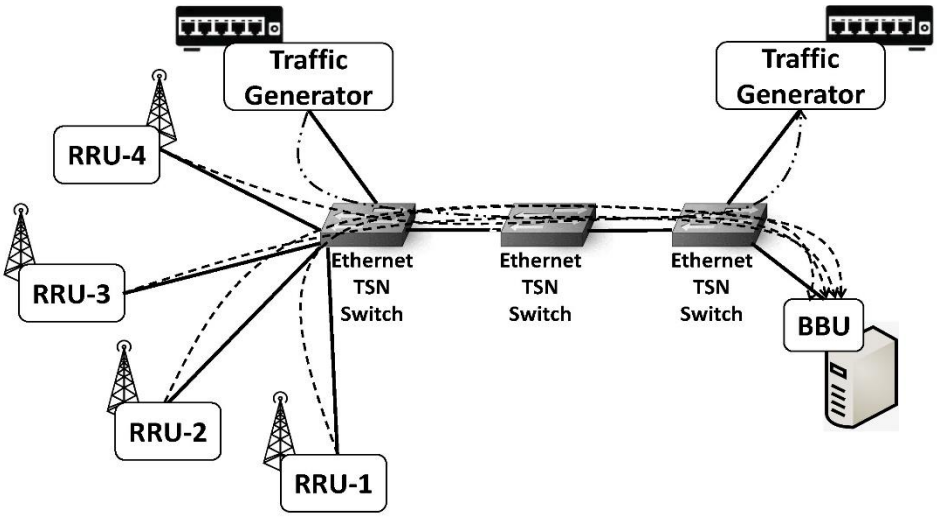

Figure 5-2: Experimentation network topology.

5.2.1.3 Evaluation Results

Figure 5-3 (a) shows the performance evaluation of IEEE 802.1Qbv. Two traffic flows are generated (both continuous and non-scheduled), one with high priority (VLAN priority 7) and one with low priority (VLAN priority 0). We consider two scenarios namely underload and overload. In the underload scenario, both traffic flows have a data rate of 592 Mbps which is approximately 60% of the line-rate (1 Gbps). In the overload scenario, both the traffic flows have a data rate of 987 Mbps (approximately 100% of the line-rate). In both scenarios and for all traffic flows, the packet length were set to 1500. As mentioned in III, with IEEE 802.1Qbv, one can assign time schedules for the opening/closing of the transmission gates. In our experiments we consider a cycle-time of 10 ms and vary the share of the cycle time for which the transmission gates for high-priority traffic can remain open. After the cycle time (10 ms) has passed the next cycle starts and the schedules are repeated again. In case of SP scheduling, the average latency for the high-priority traffic is the lowest in both the underload and overload scenario with a value of 18.2 us (reason: high-priority traffic is always given a priority above the low-priority). The SRR mechanism gives an equal opportunity to both traffic flows irrespective of their priority. We notice that the average latency values for both the lower and higher priority traffic is the same being close to 6k us. Figure 5-3 (a) shows that for IEEE 802.1Qbv, the average latency for high-priority traffic decreases as the duration of opening the transmission gates increases. It is as low as that of SP. However, the average latency increases when the transmission gate is open for a lower duration. This is expected because in this scenario, when the high-priority arrives and finds the transmission gate closed, it needs to wait till the gate re-opens in the next cycle. The key finding from this experiment is the tunability feature of 802.1Qbv being able to assign specific duration for which a particular traffic flow can be scheduled. When IEEE 802.1Qbv is enabled, parameters selection is based on the traffic aware shaper algorithm used in the case of scheduled traffic or the asynchronous traffic shaper traffic if nodes are not synchronised.

Figures Figure 5-3 (b) and Figure 5-3 (c) show the average latency and jitter for SP and 802.1Qbv for scheduled traffic (i.e. the sender and the switch are using synchronized transmission) in presence of continuous background traffic. In this experiment we also show the effect of varying packet size. For Ethernet with TSN (TSN-QBV 'A'), we fix the schedule such that the high-priority traffic is allowed to pass for 70% of the cycle-time (10 ms) and the low-priority traffic passes for the remaining cycle-time. We ensure that the background traffic generated via the *iperf* tool is of low-priority i.e. best-effort traffic (VLAN priority 0) and is sent continuously at line-rate (986 Mbps) (100% of line-rate). The scheduled traffic which is generated with the Ixia traffic generator matches that of the TSN-QBV 'A' schedule i.e. the high-priority traffic is generated for 7 ms in a cycle-time of 10 ms. Both for the high-priority and low-

priority traffic case a packet rate of 1 packet per 10ms (1 packet per cycle) is set. The packet sizes are varied from a minimum of 100 Bytes to a maximum of 1500 Bytes. Since the traffic is scheduled and the packets of the respective priority arrive when the gates for the corresponding priority class are open, TSN has the lowest average jitter, the minimum jitter being 60 ns even in presence of background traffic. The background traffic does not affect the high-priority traffic. This is because the TSN schedule prevents the interference of background traffic when the transmission gates for high-priority traffic are open. In case of SP, the minimum jitter (4 µs) and minimum latency (23 µs) are much higher than the maximum jitter (390 ns) and maximum latency (15 µs) in case of TSN. In case of the SRR scheme even higher values are observed. Since the average latency values are very high, we omit this curve for scaling purposes. The minimum average latency for SRR is 2 ms. Thus, when the TSN schedule on the switch can be set in accordance to the traffic flow, very low average jitter and latency values can be achieved and the traffic can be well protected even in presence of background traffic.

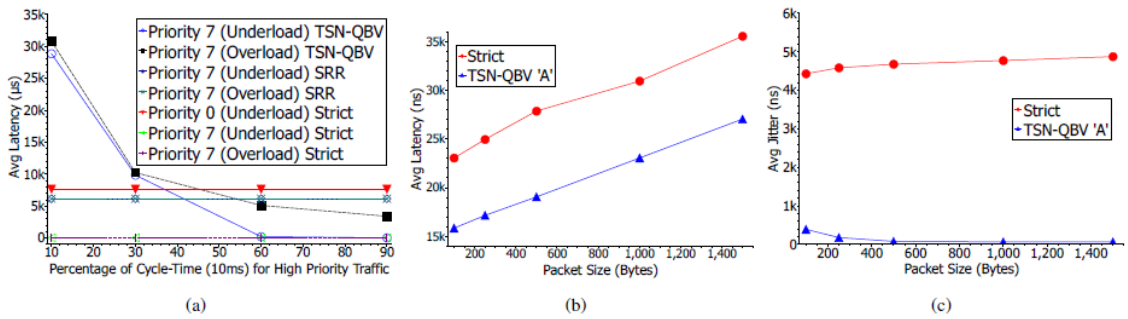

Figure 5-3: Evaluation Results: TSN 8021Qbv compared with Strict Priority and Simple Round Robin (SRR).

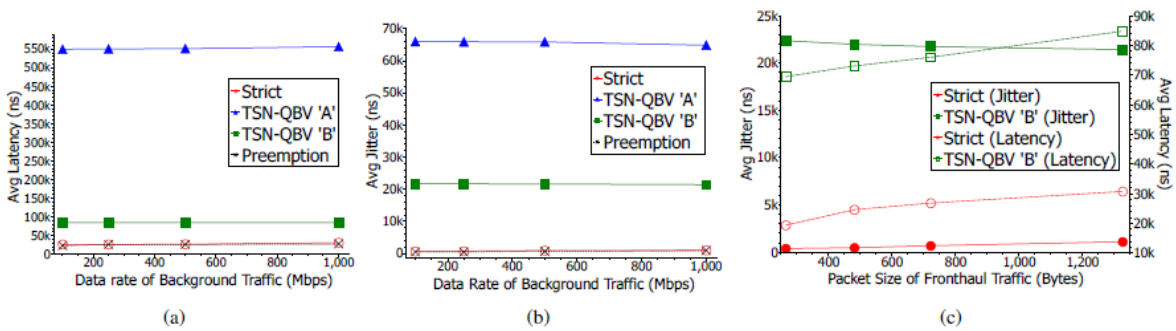

Figure 5-4: Evaluation results for fronthaul traffic over (a), (b) varying background traffic and (c) fronthaul packet sizes.

For the next set of experiments we evaluate a more realistic scenario, where we generate real fronthaul traffic between the RRU and BBU over Ethernet with TSN-802.1Qbv, Frame Preemption (TSN-802.1Qbu) and Strict Priority. We also applied the SRR scheme but as the performance values were much higher we omit the results in this paper.

Figure 5-4 (a) and Figure 5-4 (b) show the impact of background traffic on the Fronthaul traffic performance. Here we consider only one RRU and one BBU which are connected over 3 hops i.e. with two Ethernet switches in between. We vary the data rate of the background traffic from a minimum of 100 Mbps to a maximum of 1 Gbps. The packet size for the background traffic is kept constant at 100 Bytes. The fronthaul traffic sent by the RRU has a packet size of 1322 Bytes and is transmitted with an average data rate of 95 Mbps. For Ethernet with TSN we use two different schedules. One is the same as the previous experiment namely TSN-QBV 'A' and the other one is named TSN-QBV 'B' where the high-priority traffic is allowed to pass for 90% of the cycle-time (10 ms) and the low-priority traffic passes in the remaining cycle-time. We additionally evaluate the performance of 802.1Qbu (Frame Preemption)

in this experiment. Note that Frame Preemption is used independently and without the presence of 802.1Qbv. We observe that in case of SP and Frame Preemption, the difference between the maximum and the minimum average latency is around 375 ns. In case of TSN this difference is only 133 ns. This is because the background traffic is completely blocked during the time when the high-priority traffic is allowed to pass. However, the lowest average latency and jitter values are observed in case of preemption. The time taken to preempt a packet ideally depends on the size of the packet being preempted which in our case is 100 Bytes (background traffic). The minimum average latency in case of Preemption is 23 µs while the minimum latency in case of SP is 26us. This difference is even more significant when the preempted packet is as large as 1500 Bytes wherein the average latency and jitter in case of Preemption is significantly lower than for standard Ethernet or TSN-Qbv. However, for the sake of uniformity in comparison we do not demonstrate the results for this scenario. Also, in case of TSN, lower latency and jitter values can be noticed for TSN-QBV B compared to TSN-QBV A which is consistent with the results depicted in Figure 5-4 (a). The higher latency and jitter values of TSN compared to SP and Preemption is expected because the fronthaul traffic is not scheduled and the packets are transmitted by the switch only according to the time-schedule and not immediately upon arrival. However, this is acceptable because it still falls within the acceptable range mentioned by IEEE 802.1CM for High Priority fronthaul data which is 100 µs.

Figure 5-5 shows the impact of the packet sizes of the fronthaul traffic on the average latency and jitter. For this experiment, we use the same network scenario as in the previous experiment. However, the background traffic is kept constant at a data rate of 1 Gbps. The packet sizes for the fronthaul traffic is varied based on the Resource Blocks assigned. We assume fronthaul traffic with 6, 15, 25 and 50 resource blocks which correspond to packet sizes of 266 Bytes, 482 Bytes, 722 Bytes and 1326 Bytes.

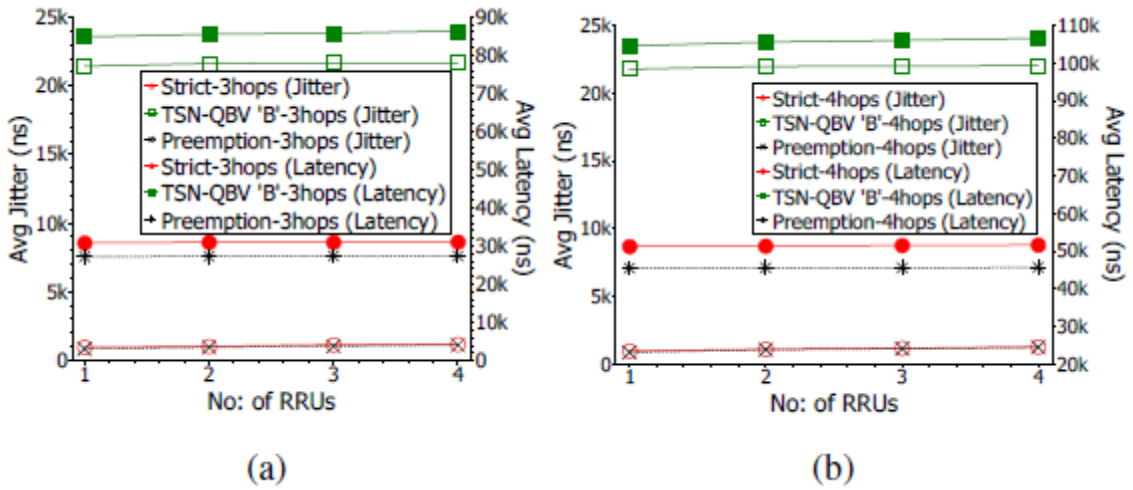

Figure 5-5: Jitter and latency for different number of RRUs and hops, (a) 3 hops and (b) 4 hops.

We observe that the average latency increases with increasing packet sizes for both TSN and Strict priority.

Figure 5-5 (a) and Figure 5-5 (b) show the impact of increasing the number of RRUs associated to one single BBU (and thus the impact of multiplexing multiple traffic flows from multiple RRUs) on the average latency and jitter. The results in Figure 5-5 (a) correspond to the network scenario with only two switches between the RRUs and the BBU i.e. 3 hops. We increase the number of hops to 4 to observe the impact on the average latency and jitter. These results are depicted in Figure 5-5 (b). In both the cases, the fronthaul packets are 1322 Bytes long and the average data rate is 95 Mbps. We notice that in both the cases, the average latency and average jitter remain more or less constant even as the number of RRUs are increased. Also, the average jitter value remains constant as the number

of hops increase from Figure 5-5 (a) to Figure 5-5 (b) while the average latency increases. Note that in both experiments, the fronthaul traffic from all the RRUs is given the same high priority. Using TSN, one can also assign different priorities to the different traffic flows from the different RRUs. By doing so, one can prioritize specific traffic flows and can accordingly assign schedules for resource sharing and protection of all traffic flows. However, with Strict priority, this might not be the case. If different priorities are assigned to different traffic flows originating from different RRUs, the Strict priority mechanism would always prioritize only the highest priority traffic and all other traffic flows would observe a significant increase in the latency and jitter.

5.3 Goal 2: Multi-domain Transport for end-to-end connectivity services

In this section we present an integrated demonstrator that illustrates how the 5G-PICTURE vision laid out in deliverable D4.1 [2], and depicted in Figure 5-6, of a having a multi-domain and multi-technology transport network across which we can interconnect RAN and Core Network functions can be implemented.

In order to fully demonstrate how an end-to-end Mobile Network service can be orchestrated through a multi-domain transport network, one needs to make use of the 5G OS functionality defined and developed in WP5. Thus, the multi-domain testbed that we present in this section has been integrated with the 5G OS developed in WP5. To report on this joint work between WP4 and WP5 we have adopted the following approach.

i. In this deliverable we will report how the end-to-end connectivity across multiple domains is achieved, and will benchmark connectivity provisioning times.

ii. In deliverable D5.4 [35] we report how the end-to-end service is provisioned, which includes the physical and virtual RAN and Core Network functions, in addition to the connectivity aspects reported here.

The first point to address when defining such a multi-domain solution is the need of a control plane that can orchestrate connectivity. For this purpose we make use of the hierarchical multi-domain control plane first defined in the 5G-XHaul project [36] and further developed in 5G-PICTURE in deliverable D5.2 [37]. Several functions of this control plane have been further extended during this work in order to accommodate the various transport technologies defined in WP4, as we describe later.

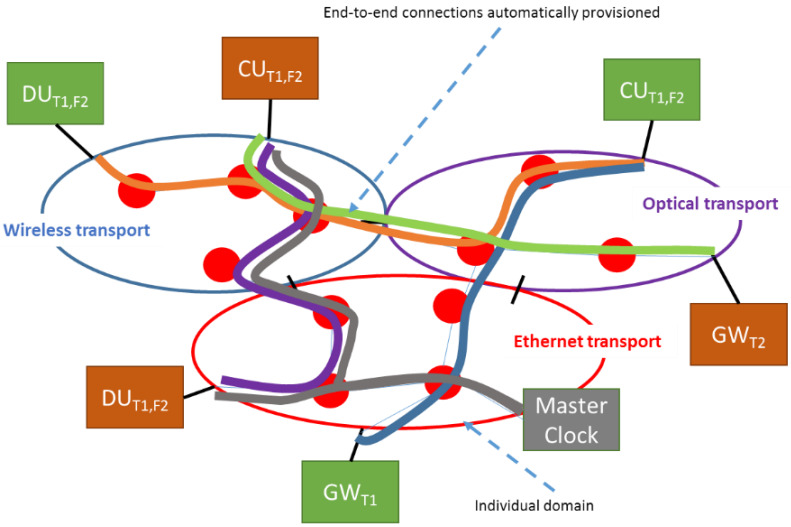

Figure 5-6: 5G-PICTURE vision of end-to-end connectivity between Mobile Network Functions.

Even though the hierarchical control plane was already reported in D5.2, we provide here a summarized description for the sake of completeness.

5.3.1 A hierarchical SDN control plane based on the Control Orchestration Protocol (COP)

The 5G-PICTURE architecture is composed of three main functions in the dataplane, namely the Transport Nodes (TNs), depicted as circles in Figure 5-7; the Edge Transport Nodes (ETNs), depicted as squares in Figure 5-7; and the Inter-Area Transport Nodes (IATNs), depicted as triangles in Figure 5-7. In a nutshell TNs are simple forwarding nodes that only how to forward packets across connections defined in their domain, ETNs bind host (virtual) network functions to the appropriate per-domain connections in order to provide end-to-end connectivity, and IATNs stitch connections between domains.

The control plane of the 5G-PICTURE transport network is composed of a hierarchy of logical controllers, as illustrated in Figure 5-7. The Top controller is responsible for orchestrating the required connectivity across different areas or domains. The Level-0 controller is responsible for the provisioning and maintenance of transport tunnels in the TNs to connect the ETNs and IATNs of a given area. A set of Level-0 controllers are logically organized under a Level-1 controller. The latter is technology-agnostic, is in charge of maintaining connectivity between the corresponding areas. Finally, ETNs and IATNs, which lie at the edges of transport areas, are directly controlled by Local Agents which are the glue between their datapaths and the Top controller with which they interact.

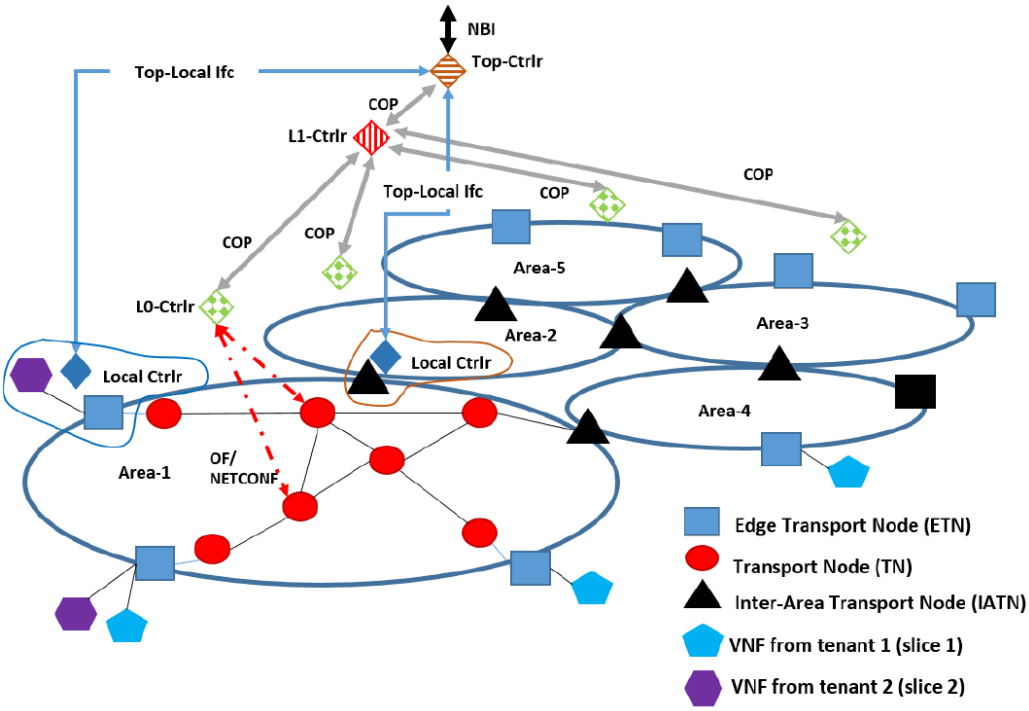

Figure 5-7: Proposed hierarchical Control Plane.

The main principle adopted in the design of the higher layers of the 5G-PICTURE control plane is the separation of responsibilities between the L1 controller and the Top controller, whereby the Top controller interfaces with the ETNs, e.g. in order to provision a new VNF, and with the IATNs in order to stitch domains. On the other hand the L1 controller's job is to act as an aggregator of L0 controllers, thus interacting only with these controllers, which end up programming the TNs of each domain.

A local ETN agent's main responsibility is to maintain mappings from virtual entity addresses to the remote ETNs hosting them, whenever these entities are attached to the same Layer 2 segment as at

least one virtual entity hosted locally at the agent's ETN. For these remote ETNs of its interest, it also maintains mappings to the respective tunnel IDs that must be used for forwarding encapsulated traffic towards them. Here we note that in cases where a tunnel traverses multiple areas, only the first tunnel ID is stored, the one leading to the first IATN along the route. This is in accordance with abstracting out unnecessary information; the ETN does not have to care about whether the destination ETN lies in the same or another area, in fact it does not know anything about areas. The ETN Local Agent also maintains mappings from local ports to the respective L2SIDs, and from local virtual entity addresses to local ports.

To understand the required steps in the establishment of an end-to-end connection we review now the detailed interactions between the Top Controller, the L1 controller and the L0 controller. The interested reader can refer to deliverable D5.2 [37] for a detailed description about the interactions between the Top Controller and the Local Agents that program the ETN and IATN functions, which we skip here for the sake of brevity.

5.3.1.1 Top – L1 and L1 – L0 controllers interactions

The interactions between Top controller and L1 controller, as well as between L1 controller and L0 controllers, make use of the Control Orchestration Protocol (COP), following the initial architecture defined in 5G-XHaul [36]. COP is a REST-based protocol, which defines a set of data models to allow REST endpoints to offer network related services. COP has been proposed as a research-oriented transport API, technology and vendor independent, that permits to abstract technology specifics of a given transport domain. It provides a multi-layer hierarchical control plane approach using YANG and RESTconf.

The following two main services are offered by the L1 controller towards the Top controller:

i. A topology dissemination service, able to retrieve topologies from individual L0 controllers and then aggregate them into an end to end topology.
ii. A path provisioning service, able to receive a path provisioning request involving nodes in different areas and resolve it into separate path requests for each of the involved areas. If there are multiple paths between two areas, the L1 controller performs routing at the area level.

In addition, the L0 controller offers the same two services towards the L1 controller:

i. A topology dissemination service, in which the L0 controller exports the topology specific to its domain in COP format. We note that the L0 controller might choose to report only a summarized version of its topology, where the only critical information to be exposed to the L1 controller are the nodes connecting to an ETN or an IATN.
i. A path provisioning service, whereby the L0 controller receives a request to connect to TNs under its control, and the L0 controller responds with the corresponding tunnel identifiers.

Since the L1 controller only interacts with the L0 controllers, not ETNs or IATNs, the COP topology will only report TNs in its list of nodes. However, the Top controller needs to be able to resolve an ETN or IATN into a TN in order to issue a path request to the L1 controller. Since the mapping between ETN/IATNs and TNs is expected to be something fairly static, we opt to manually provision the L1 controller with this information. In particular, we enable an additional REST endpoint in the L1 controller that allows to specify information about IATNs and ETNs connecting to one of the L0 domains under the control of this L1 controller. Then, the L1 controller exposes the IATN/ETN information as a COP edge. This information is sufficient for the Top controller to match ETN/IATNs with TNs and issue a path request.

5.3.1.2 End-to-end path establishment

The basic operation that requires interactions among all controllers in the hierarchy is that of an end-to-end path establishment between two ETNs that are located in different areas. The time sequence of interactions required for this procedure is summarised in Figure 5-8 for an example with two adjacent areas.

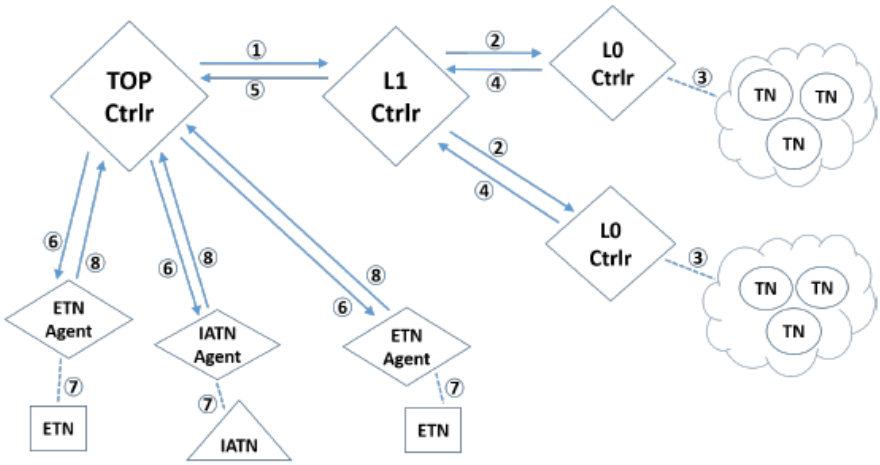

Figure 5-8: Interactions among controllers for end to end path.

At step 1, the Top controller sends a request to the L1 controller, using COP/REST for establishing a path between two ETNs. The L1 controller examines the request and finds out that the path must traverse two adjacent areas. It then breaks the original request into two sub-requests, intended for the L0 controllers of these areas. Essentially, these subrequests, sent in parallel via COP/REST at step 2, are for provisioning of connections between each ETN and the IATN stitching these areas. The L0 controllers build the connections by programming the required flows at their subordinate TNs during step 3, using OpenFlow, NETCONF or another standard protocol. At steps 4 and 5, confirmation of path establishment travels from the L0 controllers to the L1 controller, and from there to the Top controller. The procedure is not yet finished, because the ETNs and the IATN need to be instructed as well. Therefore, at step 6 and using the respective REST interfaces, the Top controller informs the ETN Local Agents of the tunnel IDs they must associate to the remote ETN, and the IATN Local Agent of the tunnel translation it must perform. These messages are sent at step 6. At step 7, the Local Agents install the required flows into the underlying datapaths, typically using OpenFlow. Finally, at step 8, the Local Agents send confirmations to the Top controller. After this point, the tunnel is ready for use.

5.3.2 Experimental validation of multi-domain and multi-technology end-to-end connectivity provisioning

In this section we describe an experimental evaluation carried out to evaluate the 5G-PICTURE multi-domain and multi-technology connectivity framework.

An initial implementation of some of the control plane functions, e.g. (L0, L1 and Top Controllers) was already available from the 5G-XHaul project (5G-XHaul D3.3 [38]), but significant technical developments were required to carry out the experiments that we describe in this section. In particular:

i. A new implementation of an L0 controller based on OpenDayLight Fluorine [39] was carried out by **I2CAT**. This implementation features a COP server module to interface with the L1 controller and a custom traffic engineering module used to manage and deploy end-to-end paths inside a domain using OpenFlow 1.3 compliant devices. This L0 controller was handed out to UTH and UPB to steer the traffic in their respective domains.

ii. A COP compliant interface was developed by **ZN** for their Dynamic Slicing Engine (DSE), as part of this work. In addition to a COP interface the DSE offers slicing functions that can benefit a transit provider to offer on-demand slices to its customer. This slicing technology and the corresponding COP interface are described in deliverable D5.4 [35].

iii. An implementation of the L0 controller tailored to the TSON technology introduced in D4.1 has been developed by **UNIVBRIS**. Due to timing issues we have however not been able to integrate TSON in the multi-domain multi-technology testbed that we introduce in this section. Instead we report on this individual integration in Section 3.2 We leave as future work the demonstration of a TSON domain inside a multi-domain end-to-end service provisioning.

iv. An initial implementation of the L1 controller originally developed by **I2CAT** in the 5G-XHaul project was significantly enhanced in this work in the following way: i) integration and interfacing with the ZN L0 controller, ii) scalability extensions as the original 5G-XHaul implementation was limited to only two domains, and iii) several bug fixes.

v. Extensions to the ETN data-plane pipeline in order to be able to route packets incoming with the VLAN identifying the end-to-end service.

vi. Extensions to the Local agents by **UTH** in order to parse the configuration of the updated ETN pipeline.

vii. Extensions to the Top Controller originally developed in 5G-XHaul by **UTH** in order to orchestrate an end-to-end service with the modifications in the various components.

We now introduce the testbed used in our evaluation that involved inter-connecting the labs of four 5G-PICTURE partners, namely **I2CAT**, **UTH**, **ZN**, and **UPB**. Notice that although ZN and UPB are not formally active within WP4, their labs were integrated in this testbed because the same testbed is used to validate the WP5 5G OS capabilities in deliverable D5.4 [35]. This testbed is illustrated in Figure 5-9.

We can see in Figure 5-9 each of the four domains inside a coloured circle (yellow for I2CAT, red for UTH, cyan for ZN and purple for UPB). Adhering to the 5G-PICTURE vision each domain features different switching technologies, namely:

- The **I2CAT** domain consists of four embedded wireless devices providing wireless backhaul connectivity using the Technology Component 11 (Table 1-1 in deliverable D4.2 [42]) and the WP3 Gateworks programmable platform reported in deliverable D3.2 [40]. A picture of the I2CAT domain is included in the top left of Figure 5-9.
- The **UTH** domain is implemented within the NITOS testbed, and uses 2 NITOS nodes as forwarding devices. A picture of the NITOS domain is included in the top-right corner of Figure 5-9.
- The **ZN** domain is implemented using EdgeCore Ethernet switches, which implement an OpenFlow 1.3 interface and ZN's proprietary network slicing technology. Special drivers were created to map vendor specific details to a common API to be used by the Dynamic Slicing Engine. A picture of the ZN domain is included in the bottom-left corner of Figure 5-9.
- The **UPB** domain is composed of x86 computers implementing software switching based on Open vSwitch using an OpenFlow 1.3 interface. A picture of the UPB domain is included in the bottom-right corner of Figure 5-9.

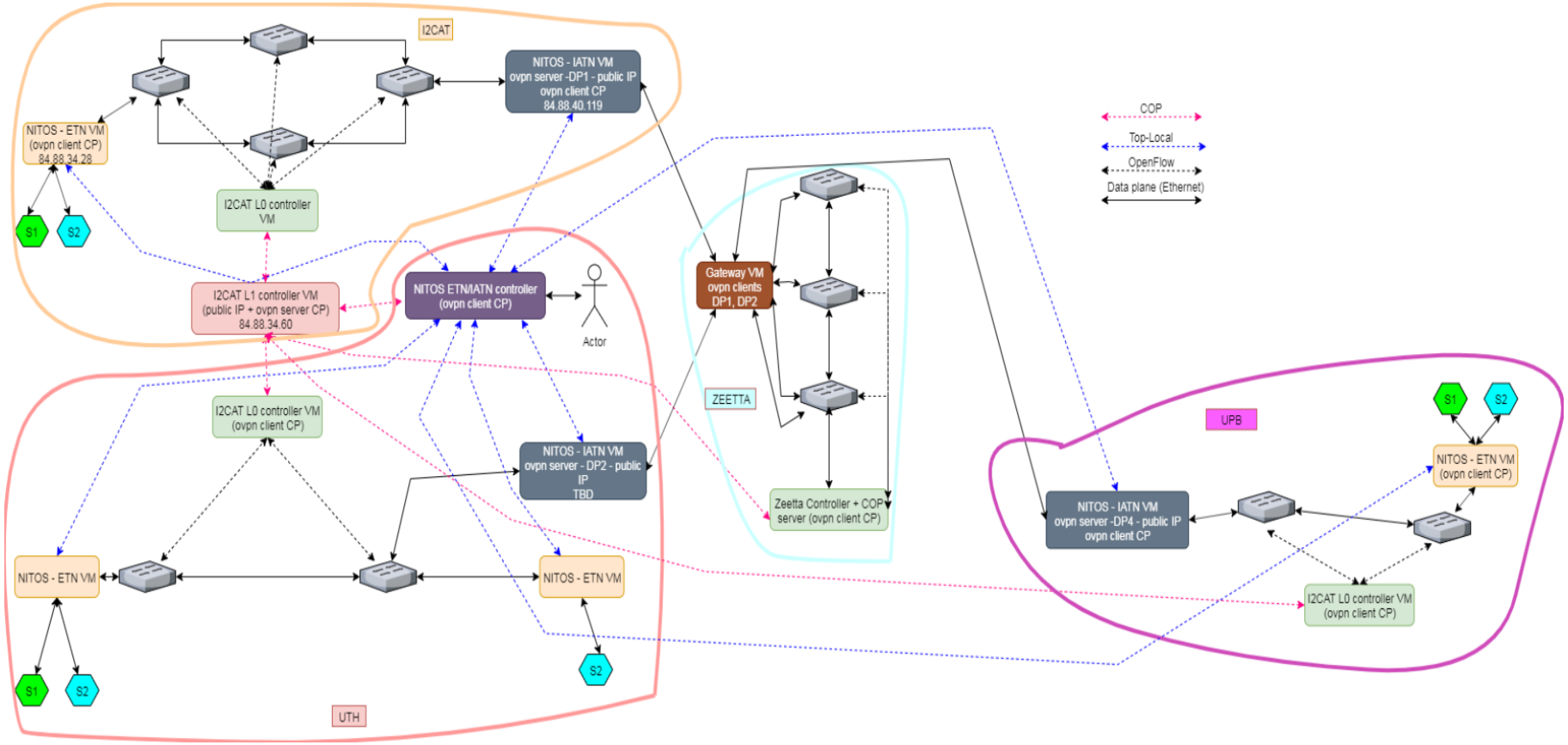

Figure 5-9: Multi-domain multi-technology testbed.

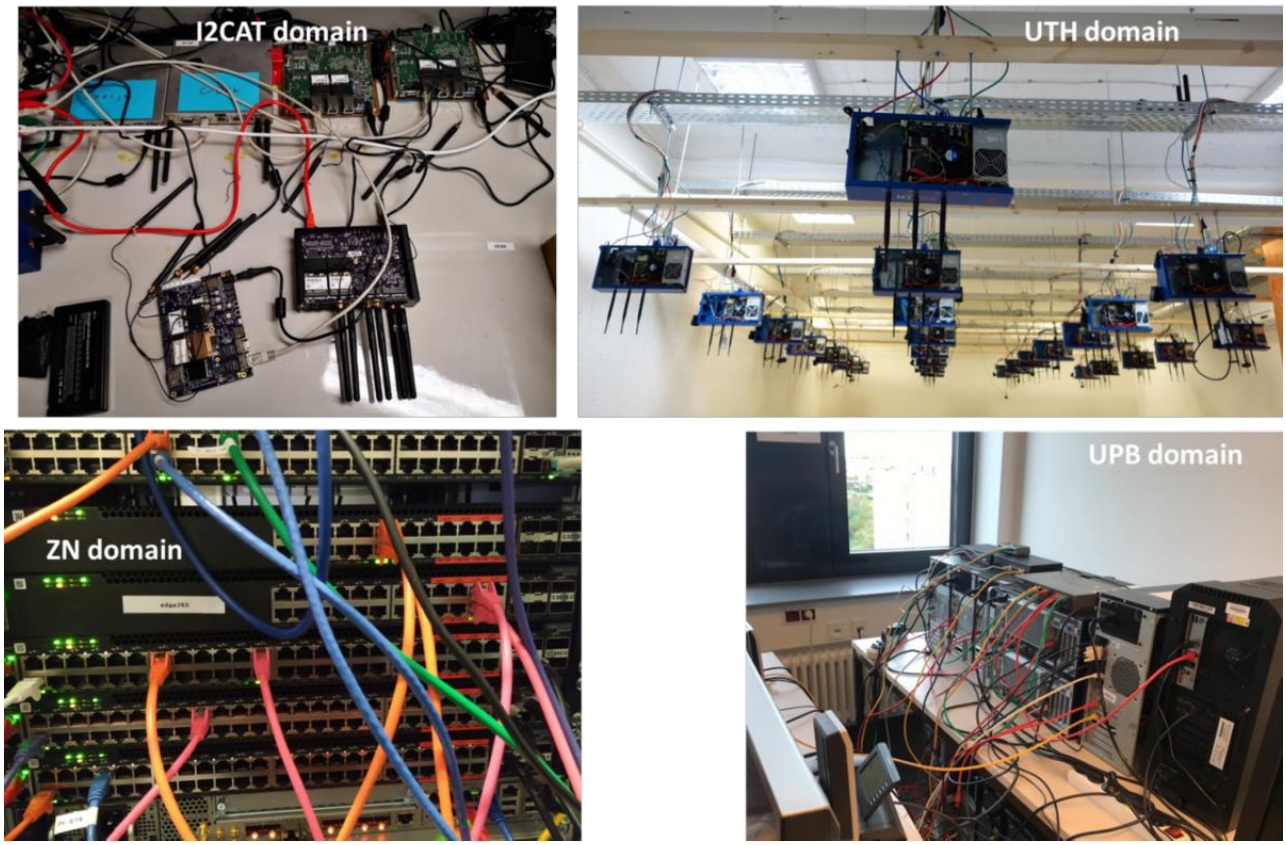

Figure 5-10: Individual testbed interconnected.

In order to interconnect the different domains we used a static set of OpenVPN (OVPN) tunnels [41] provisioned over the Internet. Naturally the end-to-end performance of our testbed is affected by the Internet performance between our labs, whereas in a real situation operators would lease dedicate transport connections between domains. We chose to use VPN tunnels over the Internet because a dedicated interconnect through GEANT was not available to all partners, and because the focus of this experiment is in the control plane performance and its interfaces, which should not be significantly affected by the reduced performance offered by the Internet connection. In order to design the VPN interconnects we adopted the following design:

- An OVPN server was deployed inside a Virtual Machine in the **I2CAT** domain. This OVPN server creates and routes a private network between all the control plane functions deployed in each domain, namely the Local, L0, L1 and Top controllers.
- For the data-plane interconnect we used dedicated OVPN connections between the IATN function in each domain and a Gateway node deployed inside the **ZN** domain. This gateway node bridges all the OVPN client interfaces interconnecting to each domain with interfaces connecting to a particular port of the Transport Nodes inside the ZN domain. In this way all domains have a single IATN connecting to the ZN domain, which acts as a transit provider, and can therefore provide end-to-end reachability. Refer to Figure 5-9 to understand the role of the Gateway function in the ZN domain. There are three switches in the ZN testbed with 'interface 2' on each switch pointing to one of the three sites.

The control plane functions, i.e. the L0, L1 and Top controllers are deployed as Virtual Machines and hosted in the respective domains, thus running an OVPN client to connect to the control plane OVPN server hosted by I2CAT (refer to Figure 5-9, where L0s are depicted with a green box, the L1 with a red box, and the Top Controller with a purple box). In the data-plane the ETN and IATN functions are also

deployed as Virtual Machines (yellow and grey boxes in Figure 5-9) running an OVPN client to connect to the control plane network, and an OVPN server in the IATNs to connect to the Gateway interconnect function in the ZN domain. An NFV infrastructure is available in each domain to host all these network functions, which we assume to be pre-provisioned for the purpose of our experiment. In addition, the VNF end-points connecting to the ETN functions are emulated in our experiment, i.e. they are not real network functions but rather virtual end-points that allow us to verify end-to-end connectivity by running tools like *ping* and *iperf*. In D5.4 we demonstrate the provisioning of a full service running real VNFs.

Finally it is worth highlighting in Figure 5-9 that a human actor triggers the provisioning of end-to-end connections through the Top Controller. This human actor is replaced by the 5G OS Multi-Domain Orchestrator (MDO) in D5.4.

To conclude this section we look at the performance (throughput and delay) of the various control plane and data plane interfaces available in our testbed. Table 5-2 illustrates the average delay in milliseconds between the Top Controller hosted in the UTH domain and the distributed control plane entities. We can see that there are 61 ms on average between the Top Controller in UTH and the L1 controller in I2CAT, which means that this will be a fixed delay when provisioning end-to-end connections. The other important interface is the interface between the Top Controller and the Local Agents in the IATN and ETN nodes of the respective domains. In this regard we see an almost negligible delay between the Top Controller and the IATN and ETN hosted in UTH, since both are in the same domain, and an average delay of 60 ms and 54 ms between the Top Controller and the IATN and ETN functions hosted by I2CAT and UPB.

Table 5-2: Control Plane interface delays.

Control plane interface	Avg. delay (ms)
MDO ↔ Top	2
Top ↔ L1	61
Top ↔ **UTH**-ETN	0.25
Top ↔ **UTH**-IATN	0.24
Top ↔ **i2CAT**-ETN	60
Top ↔ **i2CAT**-IATN	60
Top ↔ **UPB**-ETN	54
Top ↔ **UPB**-IATN	54

Data plane performance testing between the different domains, i.e. "**I2CAT** – **UTH**", "**I2CAT** – **UPB**" and "**UPB** – **UTH**" showed stable performances between 50 Mbps and 100 Mbps. This performance is determined by the internet bandwidth available at the

5.3.2.1 Multi-domain end-to-end topologies

The first step in the validation of our multi-domain testbed is to validate that the L1 controller is able to extract the end-to-end topology of the different domains through the COP interface exposed by each L0 controller. Figure 5-11 depicts the output of the GET TOPOLOGY end-point on the L1 controller including the individual topologies of each domain, which for the sake of readability are summarized. The four domains appear identified by the url parameter containing the IP address of the L0 controller for each domain, where all L0 controllers are in the IP subnet set up to support the connectivity between control plane components.

```
[
    {
        "id": 1,
        "port": 6000,
        "copTopology": {...},
        "url": "http://192.168.201.102"
    },
    {
        "id": 2,
        "port": 8181,
        "copTopology": {...},
        "url": "http://192.168.201.107"
    },
    {
        "id": 3,
        "port": 8181,
        "copTopology": {...},
        "url": "http://192.168.201.108"
    },
    {
        "id": 4,
        "port": 8181,
        "copTopology": {...},
        "url": "http://192.168.201.109"
    }
]
```


Figure 5-11: Aggregated multi-domain topology exposed by the L1 Controller. Individual per-domain topologies are hidden for readability.

Figure 5-12 depicts the detailed topology of the **i2CAT** domain, which is composed by node and edge objects. The figure also highlights the full description of a node, where we see the different interfaces encoded as edge_node objects, and the description of a link including the source and target nodes and their respective ports (localIfId and remoteIfId).

Using the aggregate multi-domain topology the L1 controller is able to provision end-to-end connections, as we will benchmark in the next section.


```
"id": 7,
"domain": "Nitos_Wireless_1571744393254",
"node_id": "openflow:1",
"name": "s1",
"nodetype": "i2cat",
"edge_end": [
    {
        "id": 18,
        "name": "s1",
        "edge_end_id": "openflow:1:LOCAL",
        "switching_cap": null,
        "peer_node_id": null
    },
    {
        "id": 19,
        "name": "s1-eth2",
        "edge_end_id": "openflow:1:2",
        "switching_cap": null,
        "peer_node_id": null
    },
    {
        "id": 20,
        "name": "s1-eth1",
        "edge_end_id": "openflow:1:1",
        "switching_cap": null,
        "peer_node_id": null
    },
    {
        "id": 21,
        "name": "tap0",
        "edge_end_id": "openflow:1:7",
        "switching_cap": null,
        "peer_node_id": null
    }
]
```

```
{
    "name": "s1:1->s2:1",
    "metric": null,
    "localIfidId": "openflow:1:1",
    "remoteIfidId": "openflow:2:1",
    "edge_id": "openflow:1:1",
    "switching_cap": null,
    "max_recv_bw": null,
    "source": "openflow:1",
    "target": "openflow:2"
},
```


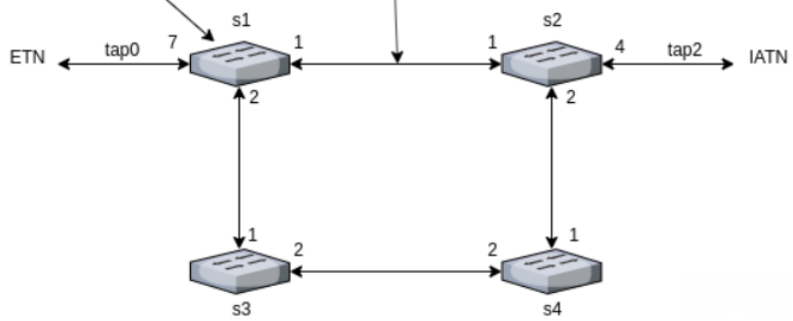

Figure 5-12: Detail of the topology in the I2CAT domain obtained by the L1 controller through the COP module

5.3.2.2 Validating and Benchmarking establishment of end-to-end connectivity

We start by validating the correct behavior of our end-to-end connectivity solution, by provisioning and end-to-end connection between the ETNs of the I2CAT and UTH domains, and capturing the packets as they traverse the various domains. This is illustrated in Figure 5-13 (**I2CAT** domain), Figure 5-14 (**ZN** domain) and Figure 5-15 (**UTH** domain).

Following the 5G-PICTURE architecture we see that packets leave the I2CAT ETN encapsulated with a Provider Backbone Bridge (PBB) header, hence obfuscating the host VNF MAC addresses from the Transport Nodes in the various domains. The outer PBB header carries a VLAN that identifies the path that packets need to follow across domains, i.e. the path identifier. This outer VLAN changes in each domain and corresponds in our experiment to VLAN 181 in the I2CAT domain, 167 in the ZN domain and 169 in the UTH domain. Notice that it is the IATN function between each pair of domains that translates this outer VLAN into the appropriate VLAN for the next domain.

We can also see how the inner Ethernet header carries a VLAN, referred to as L2SID (Layer 2 Segment ID) that identifies the network service in the ETNs. This L2SID turns out to be equal to 10 in our experiment, and as we can see it is carried end-to-end throughout all domains. Notice that the L2SID is the piece of state that allows the ETN on the receiving end to deliver incoming packets to the proper network service.

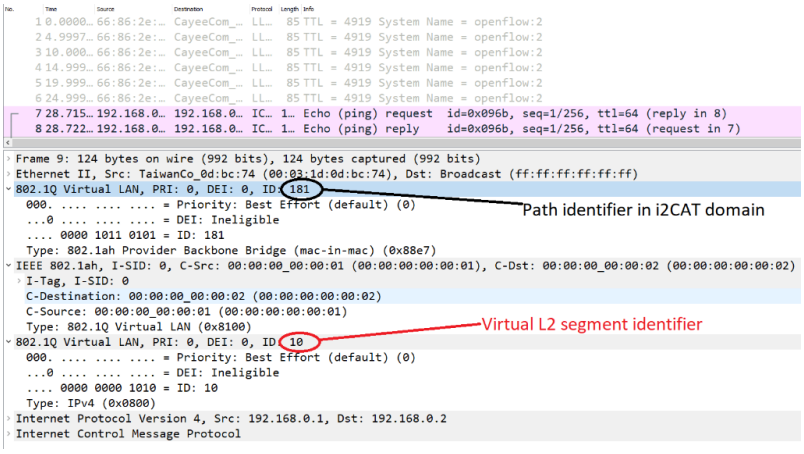

Figure 5-13: Wireshark capture in I2CAT domain.

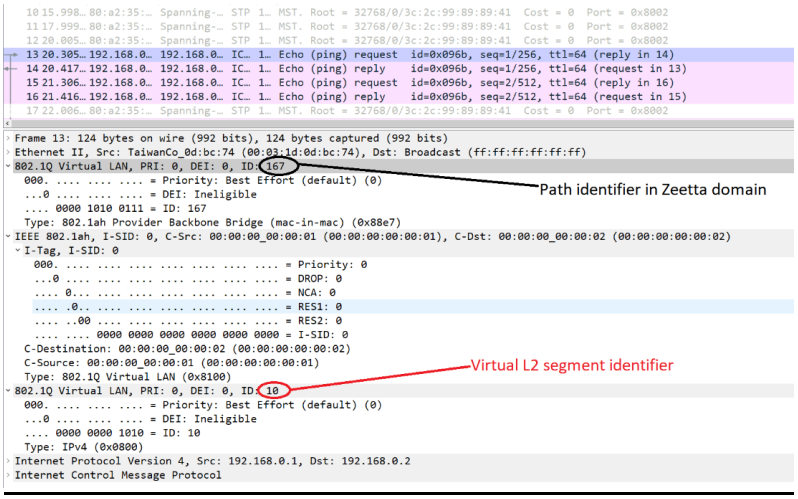

Figure 5-14. Wireshark capture in Zeetta domain

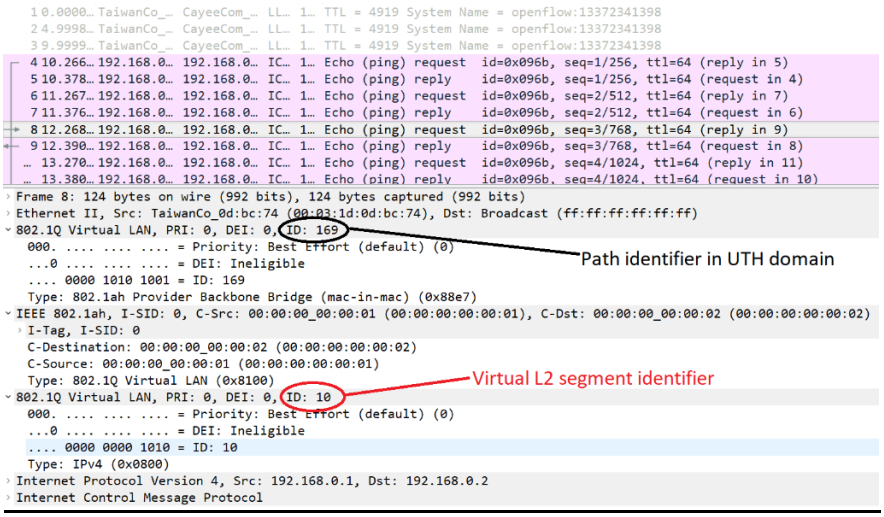

Figure 5-15. Wireshark capture in UTH domain.

We now benchmark the time required to provision automatic end-to-end connections in our multi-domain testbed using the 5G-PICTURE hierarchical connectivity solution. Figure 5-16 depicts the CDFs obtained through 20 consecutive experiments, of the provisioning times measured at the L1 controller between each pair of domains in our testbed, i.e. "**I2CAT** – **UPB**", "**UTH** – **I2CAT**" and "**UTH** – **UPB**". The provisioning times are defined as the time between the instant the Top Controller requests an end-to-end service call to the L1 controller, and the time the L1 controller returns the call with the identifiers (VLANs) of all the involved connections. Notice that this is not the overall end-to-end connection establishment time as the Top Controller still needs to parse the response and program accordingly the affected ETN and IATN end-points. For this purpose, Figure 5-17 depicts the CDF, obtained across 40 different experiments, of the end-point provisioning times. The times reported in Figure 5-17 should be added to those reported in Figure 5-16 to obtain the complete end-to-end provisioning times from the 5G OS perspective.

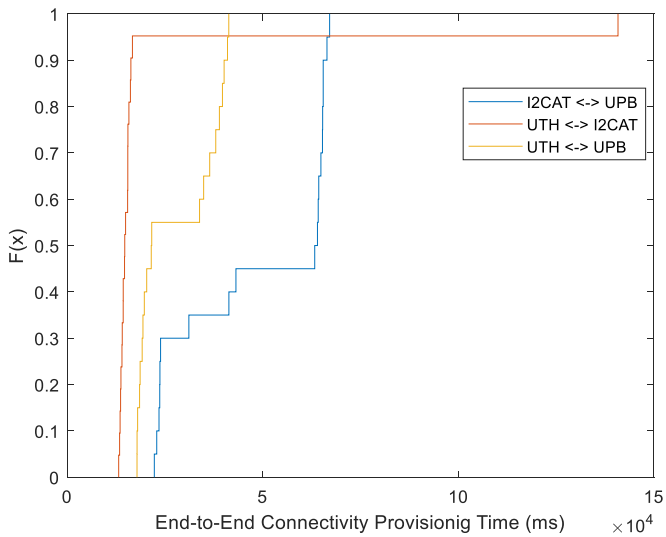

Figure 5-16: Measured end-to-end L1 provisioning times.

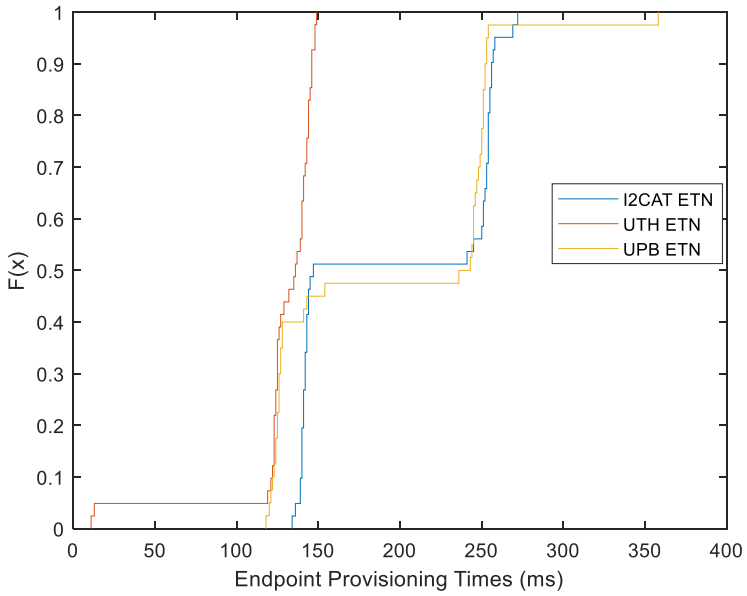

Figure 5-17: Measured per-domain ETN end-point provisioning times.

Looking at Figure 5-16 and Figure 5-17 we can see how the end-to-end provisioning times from the L1 controller clearly dominate spanning approximately from 3 to 15 seconds, whereas the end-point provisioning times span approximately between 50 and 350 milliseconds being smaller in the UTH domain. Clearly the end-to-end provisioning process is the more complex part of the signalling flow, since the L1 controller needs to break down the service-call request and individually contact the various involved L0 controllers. However the previous fact does not justify a difference of almost two orders of magnitude in the provisioning times reported in Figure 5-16 and Figure 5-17. To better understand this difference we plot separately the measured L0 provisioning times in each domain, i.e. the time from the instant the L1 controller requests a path establishment to an L0 controller until this creates the path and replies to the L1 controller.

Figure 5-18 depicts the CDF of the L0 provisioning times at the I2CAT, UTH and UPB domains, which we can see span approximately between 50 and 350 ms, being the lowest in the I2CAT domain because the L1 controller is also located at I2CAT. However, Figure 5-19 depicts the L0 provisioning times at the ZN domain that span between 2 and 15 seconds. Therefore, the behaviour of the L0 controller in the ZN domain is the reason for the observed end-to-end provisioning times in the order of seconds, since all end-to-end paths need to traverse the ZN domain.

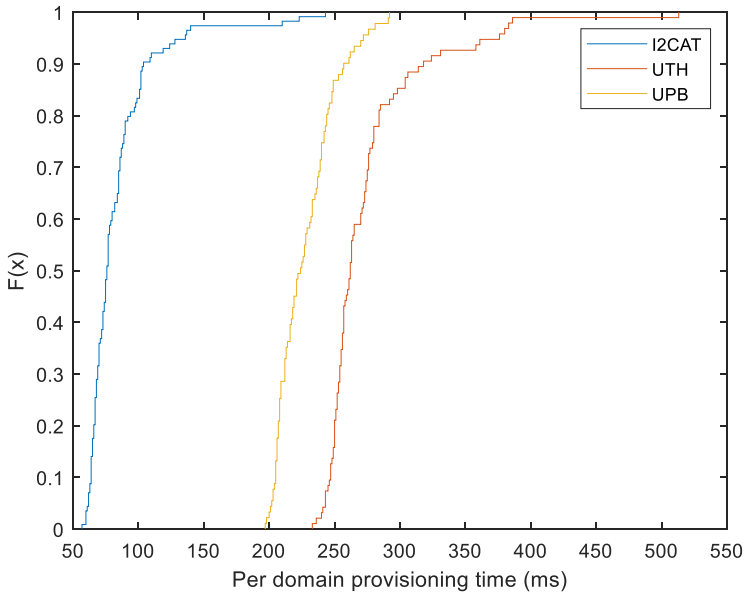

Figure 5-18: I2CAT, UTH, UPB L0 provisioning times.

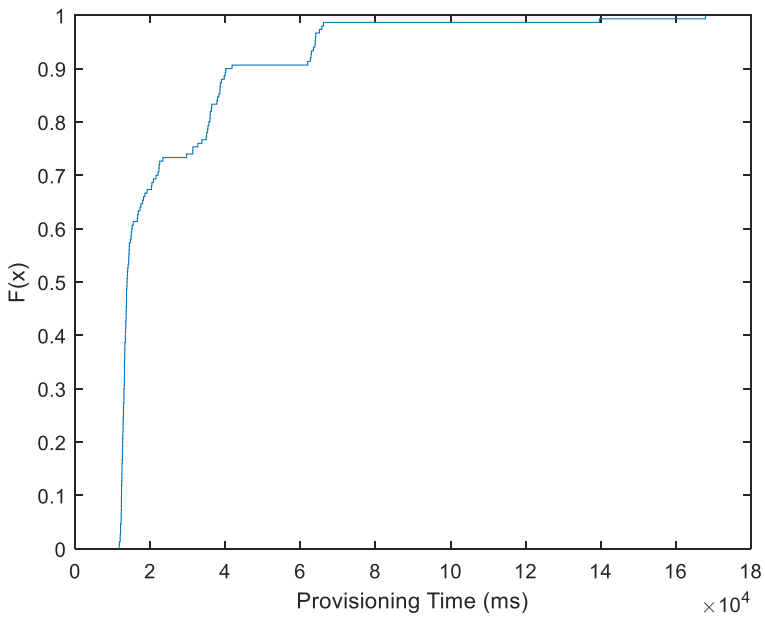

Figure 5-19: ZN L0 provisioning times.

reason why there is so much difference in the L0 provisioning times between the ZN domain and the other domains is the fact that fundamentally different L0 control plane technologies are being used in each case. The I2CAT, UTH and UPB domains all use the same L0 controller solution provided by I2CAT, which essentially implements a path provisioning service based on OpenFlow 1.3 and a COP adapter over OpenDayLight Fluorine. Hence, this L0 controller directly programs a path over the hardware or software transport nodes available in the respective domains. The L0 controller in the ZN domain is however fundamentally different due to Zeetta Network's Dynamic Slicing Engine technology. In this case, every time a path request comes from the L1 controller,

The previous results illustrate that 5G-PICTURE has been able to deliver on its vision of previsioning end-to-end connectivity across network functions connected to a multi-domain and multi-technology transport network in an autonomous manner. The designed control plane is able to establish end-to-end connectivity services involving up to three domains, including a transit provider, in less than 20 seconds in all cases. In addition, the proposed 5G-PICTURE architecture is flexible enough to incorporate heterogeneous control plane solutions, such as Zeetta's DSE, TSON or I2CAT'sa L0 controller, through the development of a simple COP adapter.

The multi-domain testbed introduced in this section is the basis of the end-to-end automated service provisioning, including connectivity and compute, reported in deliverable D5.4 [35].

5.4 Goal 3: Synchronisation through a multi-technology transport network

As part of the integrated demonstrator development in WP4, a small scale demonstration of the synchronisation harmonizer algorithms (presented in Section 4.5) is planned to be reported in deliverable D6.3. The primary goal is to demonstrate the optimal selection of the synchronisation path under realistic conditions, for transport networks using different technologies. For this purpose, we are considering a scenario where one slave (i.e. ordinary clock) is connected to different grandmaster clocks (GC) through one wired and one wireless path, as depicted in Figure 5-20.

The standard IEEE 1588v2 PTP protocol would be employed on all nodes for the underlying synchronisation messages exchange. The synchronization harmonizer algorithms would be implemented in Python and would run independently on a local PC, with the PTP management client (PMC) used as an interface to the clock nodes. Ingress and egress hardware timestamps obtained from the PTP synchronization messages would be used as an input to the synchronisation harmonizer. The best performing synchronisation path in terms of lowest clock offset error and standard deviation would then be determined based on the algorithms described in Section 4.5. Subsequently, the slave would be synchronized to the Grandmaster Clock with the best statistical performance as yielded by the synchronisation harmonizer, rather than relying on the BMCA defined in IEEE1588v2. The BMCA algorithm could be implicitly overriden by, e.g., changing the slave's ClockClass or Priority fields as a result of the synchronisation harmonizer output.

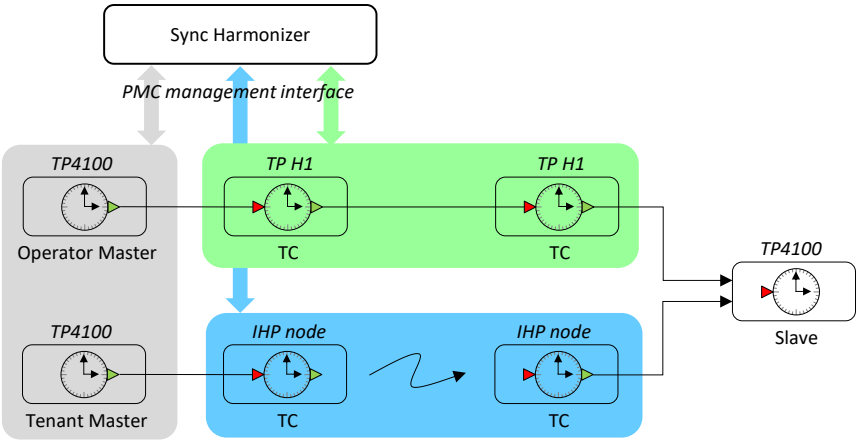

Figure 5-20: Small-scale demonstration scenario for the synchronisation harmonizer, containing one wired and one wireless synchronisation path through transparent clocks of different transport technologies.

The hardware demonstrator setup in its current state is depicted in Figure 5-21. Initial tests have successfully performed for the wired path, using a pair of the TransPacket H1 TC nodes provided by the partner TP.

5.4.1 Description of the tests and results

In order to validate the synchronisation harmonizer function, a number of tests will be performed with the above mentioned HW setup (Figure 5-20).

For this purpose, we would configure the two TP4100 (annotated as Operator Master and Tennant Master) as PTP master nodes with a specific PTP profile that allows for a high rate exchange of PTP messages, e.g., telecom-2008 with 64 messages per second. The receiving TP4100 would correspondingly be configured as a PTP telecom-2008 slave node. Subsequently, the wired and wireless synchronisation paths will be established in parallel, connecting the Operator Master and the Tennant Master to the Slave node, correspondingly. By performing packet sniffing on the two synchronization paths, the PTP messages that are being exchanged between the nodes will be collected and their hardware timestamps will be extracted, serving as an input to the synchronisation harmonizer function. Using these HW timestamps, the Belief Propagation and the Kalman Filtering based approaches described in section 4.5 would be run in a Python implementation on a local PC and would be thus tested in realistic conditions. Assuming a static scenario with a wired path of fixed cable length as well as a static wireless link, the measurement of the clock offset for both paths will be of primary interest, as the mean path delay would be approximately constant. Clock offset measurements for both paths in parallel will be performed in a 60 minute interval in order to collect a sufficient number of timestamps.

Since the synchronisation harmonizer function is not implemented locally on the master and slave nodes, but rather as an experimental Python implementation on a PC, the post-processing of the collected timestamps will be performed off-line after the measurement interval. This has the added benefit of allowing further modifications and improvements of the synchronisation harmonizer function based on the same set of measured data. As the main outcome of the performed measurement, the estimated clock offset error and its standard deviation will be compared for both synchronization paths and for both synchronization approaches. The switching of the synchronisation path as the main output of the synchronization harmonizer algorithms, will be plotted alongside these results. This test would ultimately allow us to evaluate the measured statistical performance of the clocks, and the decision mechanism of the synchronisation harmonizer function in realistic conditions.

Figure 5-21: Hardware demonstrator setup for the synchronisation harmonizer.

5.4.2 Future plans

With respect to the ongoing development of the wireless path, it will be established using a pair of transparent clock capable mmWave nodes in a fixed point-to-point link, operating at 60 GHz. For this purpose, we would ideally employ BWT's Typhoon nodes, which were recently unavailable due to being allocated to the Interim Review Railway demonstration in Barcelona. Due to the problems acquiring the wireless synchronisation devices, only an initial description of the demonstrator is reported, and the final results will be included in deliverable D6.3.

6 Summary and Conclusions

This deliverable concludes the work in WP4 providing two main contributions:

- A complete evaluation of the Technical Components identified in D4.1 [2] that had not been reported in D4.2 [1], and an update on some of the other Technical Components. With these results WP4 has delivered a total of 16 novel Technical Components in the areas of RAN, transport and synchronization that represent a significant step towards the vision of programmable and disaggregated 5G network.
- Three integrated demonstrators that validate the main novel concepts put forward by WP4. The first demonstrator validates that a single transport network can be used to transport latency sensitive fronthaul flows and background traffic, using Ethernet TSN technology. The second demonstrator validates the ability to provision end-to-end connectivity services on demand through a multi-technology transport network in only a matter of seconds. The third demonstrator validates the concept of a Synchronization Harmonizer that is a novel centralized control plane function that can steer clock distribution paths across a multi-technology transport network.

Several of the integrated demonstrators and Technology Components described in this and the previous WP4 deliverables have been used by the 5G OS developed in WP5 in order to orchestrate end-to-end services. These include:

- The multi-domain transport testbed that is used to support the demonstration of two end-to-end services consuming wireless, transport and compute resources in deliverable D5.4 [35].
- The TSN based fronthaul demonstrator that is used to illustrate the joint orchestration of virtual RAN functions and TSN flows in deliverable D5.4 [35].
- The joint access and backhaul Wi-Fi Technical Component, which is used to support and end-to-end virtual Wi-Fi service in deliverable D5.4 [35].
- The TSON control plane functions developed in WP4, which are used to demonstrate the 5G OS MDO concept in deliverable D5.4 [35].

Some of the reported Technical Components, such as TSON or the joint access backhaul Wi-Fi devices, have also used the programmable platforms developed in WP3. Therefore, we conclude that we have been able to accomplish the overarching goal of WP4 that was developing novel functions in the RAN, transport and synchronization domains that would make use of the WP3 programmable platforms and could be orchestrated by the 5G OS.

7 References

[1] 5G-PICTURE, "D4.2 Complete design and initial evaluation of developed functions", Deliverable D4.2, Nov 2018

[2] 5G-PICTURE, "D4.1 State of the art and initial function design", April 2018

[3] 5G-PICTURE, "D3.1 Initial report on Data Plane Programmability and infrastructure components", March 2018

[4] Garcia-Villegas, E., Sesto-Castilla, D., Zehl, S., Zubow, A., Betzler, A., & Camps-Mur, D., June). "SENSEFUL: An SDN-based joint access and backhaul coordination for Dense Wi-Fi Small Cells", 13th International Wireless Communications and Mobile Computing Conference (IWCMC), pp. 494-499, 2017

[5] ptp4l – PTP Boundary/Ordinary Clock, https://linux.die.net/man/8/ptp4l

[6] Setting time stamping policy at the driver level, https://linux.die.net/man/8/hwstamp_ctl

[7] ETSI TS 136 211 V14.2.0 (2017-04). "LTE; Evolved Universal Terrestrial Radio Access (E-UTRA);. Physical channels and modulation."

[8] J. Eidson and K. Lee, "IEEE 1588 standard for a precision clock synchronization protocol for networked measurement and control systems," in Sensors for Industry Conference, 2002. 2nd ISA/IEEE, 2002.

[9] S. Ruffini, P. Iovanna, M. Forsman and T. Thyni, "A novel SDN-based architecture to provide synchronization as a service in 5G scenarios," IEEE Communications Magazine, vol. 55, pp. 210-216, 2017.

[10] B. Lv, Y. Huang, T. Li, X. Dai, M. He, W. Zhang and Y. Yang, "Simulation and performance analysis of the IEEE1588 PTP with kalman filtering in multi-hop wireless sensor networks," Journal of networks, vol. 9, p. 3445, 2014.

[11] G. Giorgi and C. Narduzzi, "Performance analysis of Kalman-filter-based clock synchronization in IEEE 1588 networks," IEEE Transactions on Instrumentation and Measurement, vol. 60, pp. 2902-2909, 2011.

[12] M. Leng and Y.-C. Wu, "Distributed clock synchronization for wireless sensor networks using belief propagation," IEEE Transactions on Signal Processing, vol. 59, pp. 5404-5414, 2011.

[13] B. Etzlinger, H. Wymeersch and A. Springer, "Cooperative synchronization in wireless networks," IEEE Transactions on Signal Processing, vol. 62, pp. 2837-2849, 2014.

[14] F. R. Kschischang, B. J. Frey, H.-A. Loeliger and others, "Factor graphs and the sum-product algorithm," IEEE Transactions on information theory, vol. 47, pp. 498-519, 2001.

[15] Y.-C. Wu, Q. Chaudhari and E. Serpedin, "Clock synchronization of wireless sensor networks," IEEE Signal Processing Magazine, vol. 28, pp. 124-138, 2010.

[16] D. Barber, "Bayesian Reasoning and Machine Learning", Cambridge University Press, 2012.

[17] A. L. Barker, D. E. Brown and W. N. Martin, "Bayesian estimation and the Kalman filter," Computers \& Mathematics with Applications, vol. 30, pp. 55-77, 1995.

[18] J. Du and Y.-C. Wu, "Distributed clock skew and offset estimation in wireless sensor networks: Asynchronous algorithm and convergence analysis," IEEE Transactions on Wireless Communications, vol. 12, pp. 5908-5917, 2013.

[19] A. A. M. Saleh, A. Rustako, and R. Roman, "Distributed antennas for indoor radio communications," IEEE Transactions on Communications, vol. 35, no. 12, pp. 1245–1251, December 1987.

[20] Clayton Shepard, Hang Yu, Narendra Anand, Erran Li, Thomas Marzetta, Richard Yang, and Lin Zhong, "Argos: practical many-antenna base stations." In Proceedings of the 18th annual international conference on Mobile computing and networking (Mobicom '12). ACM, New York, NY, USA, 2012.

[21] R. Rogalin et al., “Scalable Synchronization and Reciprocity Calibration for Distributed Multiuser MIMO,” in IEEE Transactions on Wireless Communications, vol. 13, no. 4, pp. 1815-1831, April 2014.

[22] H. Papadopoulos, O. Y. Bursalioglu and G. Caire, “Avalanche: Fast RF calibration of massive arrays,” 2014 IEEE Global Conference on Signal and Information Processing (GlobalSIP), Atlanta, GA, 2014, pp. 607-611.

[23] X. Jiang, A. Decunringe, K. Gopala, F. Kaltenberger, M. Guillaud, D. Slock, and L. Deneire, “A framework for over-the-air reciprocity calibration for TDD massive MIMO systems,” IEEE Transactions on Wireless Communications, vol. 17, no. 9, pp. 5975–5990, Sep. 2018.

[24] A. Nasrallah, A. Thyagaturu, Z. Alharbi, C. Wang, X. Shao,M. Reisslein, and H. Elbakoury, “Ultra-low latency (ull) networks: A comprehensive survey covering the IEEE TSN standard and related ull research”, 2018.

[25] J. Sommer, S. Gunreben, F. Feller, M. Kohn, A. Mifdaoui, D. Saß, and J. Scharf, “Ethernet–a survey on its fields of application,” IEEE Communications Surveys & Tutorials, vol. 12, no. 2, pp. 263–284, 2010.

[26] N. Nikaein, M. K. Marina, S. Manickam, A. Dawson, R. Knopp, and C. Bonnet, “Openairinterface: A flexible platform for 5g research,” ACM SIGCOMM, vol. 44, no. 5, pp. 33–38, 2014.

[27] C. Chia-Yu, S. Ruggero, N. Navid, S. Thrasyvoulos, and B. Christian, “Impact of packetization and functional split on c-ran fronthaul performance,” in Proc. Of IEEE ICC, 2016.

[28] C. Chang, N. Nikaein, R. Knopp, T. Spyropoulos, and S. S. Kumar, “Flexcran: A flexible functional split framework over ethernet fronthaul in cloud-ran,” in 2017 IEEE International Conference on Communications (ICC), May 2017, pp. 1–7.

[29] K. Lee, S. Hong, S. J. Kim, I. Rhee, and S. Chong, “SLAW: Self-Similar Least-Action Human Walk,” IEEE/ACM Trans. Netw., vol. 20, no. 2, pp. 515-529, Apr. 2012.

[30] E. Liotou, et al, “QoE-SDN APP: A Rate-guided QoE-aware SDN-APP for HTTP Adaptive Video Streaming,” IEEE Journal on Selected Areas in Communications, Vol.36, No.3, Mar. 2018

[31] D. Naddef and G. Rinaldi, “Branch-and-cut algorithms for the capacitated VRP”, In book: The Vehicle Routing Problem, Society of Industrial and Applied Mathematics, 2001.

[32] E. Pateromichelakis, et al, "Joint routing and scheduling in dense small cell networks using 60 GHz backhaul," IEEE ICCW, London, 2015.

[33] L. Tassiulas and A. Ephremides, “Stability Properties of Constrained Queuing Systems and Scheduling Policies for Maximum Throughput in Multihop Radio Networks”, IEEE Tran. On Automatic Control, 37(12):1936–1948, Dec. 1992.

[34] E. Pateromichelakis, K. Samdanis, Q. Wei and P. Spapis, “Slice-Tailored Joint Path Selection & Scheduling in mm- Wave Small Cell Dense Networks,” IEEE Globecom, Singapore, Dec. 2017.

[35] 5G-PICTURE, “D5.4 Integrated prototype (across tasks and work packages)”.

[36] 5G-XHaul, https://www.5g-xhaul-project.eu/

[37] 5G-PICTURE, “D5.2 Auto-adaptive hierarchies”, May 2019

[38] 5G-XHaul, “D3.3 5G-Xhaul algorithms and services Design and Evaluation”, August 2018

[39] OpenDayLight Fluorine. Available at: https://www.opendaylight.org/what-we-do/current-release/fluorine

[40] 5G-PICTURE, “D3.2 Intermediate report on Data Plane Programmability and infrastructure components“, November 2018

[41] OpenVPN. Available at: https://openvpn.net/

[42] 5G-PICTURE, “D4.2 Complete design and initial evaluation of developed functions”, Nov. 30th 2018.

8 Acronyms

Acronym	Description
3GPP	Third Generation Partnership Project
5G	Fifth Generation
5G-OS	5G Operating System
AAA	Adaptive Antenna Array
AP	Access Point
API	Application Programming Interface
BBU	Base Band Unit
BC	Boundary Clock
BH	Backhaul
BS	Base Station
CDF	Cumulative Distribution Function
CoMP	Coordinated Multi-Point
COP	Control Orchestrator protocol
COTS	Commercial Off-The-Shelf
CPRI	Common Public Radio Interface
C-RAN	Cloud-RAN
CU	Centralised Unit
DAS	Distributed Antenna System
DL	Downlink
DMRS	Demodulation Reference Symbol
DSCP	Differentiated Services Code Point
DSE	Dynamic Slicing Engine
DU	Distributed Unit
eCPRI	enhanced CPRI
EE	Energy Efficiency
eMAC	Express MAC
eMBB	enhanced Mobile Broadband
ETN	Edge Transport Node
FG	Factor Graphs
FFT	Fast Fourier Transform
FH	Fronthaul
Flex-E	Flexible Ethernet
FPGA	Field-Programmable Gate Array
FT	Fourier Transform

GCL	Gate Control List
HPF	High priority Fronthaul
HPN	High Performance Networks
I/Q	In-phase/Quadrature
IATN	Inter-Area Transport Node
IP	Internet Protocol
JSON	JavaScript Object Notation
KF	Kalman Filtering
KPI	Key Performance Indicator
LO	Local oscillator
LP	Linear Program
LPF	Low Priority Fronthaul
LTE	Long Term Evolution
LUT	Look Up Table
MAC	Medium Access Control
MDO	Multi-Domain Orchestrator
MEC	Mobile Edge Computing
MLME	Media Access Control (MAC) Sublayer Management Entity
mMIMO	massive Multiple-Input Multiple-Output (MIMO)
mmWave	millimetre Wave
MPF	Medium Priority Fronthaul
NIC	Network Interface Card
NPUI	Network Processor Unit
NR	New Radio
NTP	Network Time Protocol
OAI	OpenAirInterface
OC	Ordinary Clock
OCXO	Oven Controlled Crystal Oscillator
OFDM	Orthogonal Frequency-Division Multiplexing
OFDMA	Orthogonal Frequency-Division Multiple Access
ONOS	Open Networking Operating System
OPP	Open Packet Processor
OS	Operating System
OSM	Open Source MANO
PBB	Provider Backbone Bridge
PDCCH	Physical Data Control Channel
PDCP	Packet Data Convergence Protocol

PHY	Physical
PLL	Phase-locked loop
pMAC	preemptable MAC
PPS	Pulse-Per-Second
PSS	Primary Synchronization Sequence
PTP	Precision Time Protocol
QoS	Quality of Service
QoT	Quality of Transmission
RAN	Radio Access Network
RAU	Radio Aggregation Unit
RCC	Radio Cloud Center
REST	Representational State Transfer
RF	Radio Frequency
RRH	Remote Radio Head
RRU	Remote Radio Unit
SC	Service Call
SDN	Software-Defined Networking
SE	Spectral Eifficiency
SINR	Signal to Interference and Noise Ratio
SLAW	Self Similar Least-Action Walk
SLL	Side-lobe level
SP	Strict Priority
SRR	Simple Round Robin
ST	Service Toplogy
STS	System Timestamping
SW	Software
TDD	Time-Division Duplexing
TDM	Time Division Multiplexing
TDMA	Time Division Multiple Access
TG	Task Group
TN	Transport Node
TSN	Time Sensitive Networking
TSON	Time Shared Optical Network
UDP	User Datagram Protocol
UE	User Equipment
UL	Uplink
UMi	Urban Micro

VCO	Voltage-Controlled Oscillator
VCTCXO	Voltage Controlled Temperature Compensated Crystal Oscillator
VLAN	Virtual LAN
VNF	Virtual Network Function
VPN	Virtual Private Network
WDM	Wavelength Division Multiplexing