5G Programmable Infrastructure Converging disaggregated network and compUte Resources

D3.2 Intermediate report on Data Plane Programmability and infrastructure components

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: 33 months

Call: H2020-ICT-2016-2

Date of delivery: 30th November 2018

Topic: ICT-07-2017

Version 1.0

Project co-funded by the European Commission

Under the H2020 programme

Dissemination Level: Public

Grant Agreement Number:	762057
Project Name:	5G Programmable Infrastructure Converging disaggregated network and compUte REsources
Project Acronym:	5G-PICTURE
Document Number:	**D3.2**
Document Title:	Intermediate report on Data Plane Programmability and infrastructure components
Version:	1.0
Delivery Date:	30th November 2018
Responsible:	Consorzio Nazionale Interuniversitario per le Telecomunicazioni (**CNIT**)
Editor(s):	Salvatore Pontarelli (**CNIT**), Stefan Zimmermann(**ADVA**)
Authors:	Jens Bartelt (**AIR**), August Betzler (**i2CAT**), Giuseppe Bianchi (**CNIT**), Steinar Bjørnstad (**TP**), Marco Bonola (**CNIT**), Daniel Camps (**i2CAT**), Chia-Yu Chang (**EUR**), Jay-Kant Chaudhary (**TUD**), Nebojsa Maletic (**IHP**), Marcus Ehrig (**IHP**), Jesús Gutiérrez (**IHP**), Paris Flegkas (**UTH**), Nikos Makris (**UTH**), Eduard García-Villegas (**i2CAT**/**UPC**), Arash Beldachi (**UNIVBRIS-HPN**), Joan J. Aleixendri (**i2CAT**), Matty Kadosh (**MLNX**), Vaia Kalokidou (**UNIVBRIS-CSN**), Peter Legg (**BWT**), Salvatore Pontarelli (**CNIT**), Marco Spaziani (**CNIT**), Anna Tzanakaki (**UNIVBRIS-HPN**), Raimena Veisllari (**TP**), Stefan Zimmermann(**ADVA**).
Keywords:	Dataplane programmability, programmable network platform, optical technologies, wireless technologies.
Status:	Draft
Dissemination Level	Public
Project URL:	http://www.5g-picture-project.eu/

Revision History

Rev. N	Description	Author	Date
0.1	Initial Draft, initial contributions to Section 2.4	Salvatore Pontarelli (CNIT), Marco Bonola (CNIT), Giuseppe Bianchi (CNIT)	27/06/2018
0.1	Contribution of the Flex-E description and the X-Ethernet description (Sections 4.3 and 4.4)	Kostas Katsalis (HWDU)	11/09/2018
0.1	Contribution on the RF processing/model-ling for simulation (Section 4.5)	Vaia Kalokidou (UNIVBRIS-CSN)	13/09/2018
0.1	Contribution on BB Processing in Active Antenna Distributed Unit (Section 4.6)	Jens Bartelt (AIR)	17/09/2018
0.1	Contribution to Time Sensitive Ethernet (Section 2.7)	Steinar Bjørnstad (TP)	18/09/2018
0.1	Contribution to Section 3.1	Peter Legg (BWT)	26/09/2018
0.1	Contribution on the FlexRAN platform (Section 2.5)	Chia-Yu Chang (EUR)	27/09/2018
0.1	Contribution on APIs for the GateWorks Ventana platform (Section 3.2)	Daniel Camps, Eduard García, Joan J. Aleixendri, August Betzler (i2CAT/UPC)	29/09/2018
0.1	Contribution on Section 4.2	Steinar Bjørnstad (TP), Jim Zou (ADVA)	01/10/2018
0.1	Contribution on OAI Interfaces for Physical Network Functions (Section 3.3)	Nikos Makris (UTH), Paris Flegkas (UTH)	03/10/2018
0.2	First merge	Salvatore Pontarelli (CNIT)	03/10/2018
0.2	Contribution on the development of P4 compiler for Spectrum device (Section 3.4)	Matty Kadosh (MLNX)	08/10/2018
0.2	Contribution on TSON, Sections 2.1, 2.2. and 2.3	Arash Beldachi (UNIVBRIS-HPN), Anna Tzanakaki (UNIVBRIS-HPN)	12/10/2018
0.2	Contributions on P2MP MAC processor and on LoS MIMO (Sections 2.6 and 4.7)	Marcus Ehrig (IHP), Nebojsa Maletic (IHP),	23/10/2018
0.2	NETCONF/YANG service development framework for ADVA's programmable edge device platforms (Section 2.8), and Section 4.1	Stefan Zimmermann (ADVA), Jim Zou (ADVA)	23/10/2018
0.3	Second merge (at Technical Meeting in Oslo)	Stefan Zimmermann (ADVA)	23/10/2018
0.4	Revised version with improved formatting, numbering, and additional contributions from i2CAT and IHP	Eduard García (i2CAT/UPC), Nebojsa Maletic (IHP), Jesús Gutiérrez (IHP), Stefan Zimmermann (ADVA)	30/10/2018
0.5	Further revised version with merged and updated bibliography, and additional contri-butions from ADVA (Section 4.8)	Stefan Zimmermann (ADVA), **ALL**	01/11/2018

0.6	Revised contribution on TSON, Sections 2.1, 2.2. and 2.3	Arash Beldachi (**UNIVBRIS-HPN**)	09/11/2018
0.7	Internal Review phase	Jens Bartelt (**AIR**)	10/11/2018
0.8	Revised version after internal review phase	Salvatore Pontarelli (**CNIT**)	12/11/2018
0.8	Further revised version after internal review phase, including additional corrections from ADVA	Stefan Zimmermann (**ADVA**)	26/11/2018
0.9	Final revised document	Anna Tzanakaki (**UNIVBRIS**), Jesús Gutiérrez (**IHP**)	28/11/2018
1.0	Submission to the EC	Jesús Gutiérrez (**IHP**)	30/11/2018

Table of Contents

List of Figures

List of Tables

Executive Summary

This document corresponds to deliverable D3.2, entitled “Intermediate report on Data Plane Programmability and infrastructure components” of the H2020 5G-PICTURE project. The deliverable provides the detailed functional definition of the programmable platforms that are being developed within 5G-PICTURE’s WP3 activities, the implementation results and a preliminary evaluation of the performance achievable with the developed platforms.

To exploit the programmability of these platforms a set of suitable hardware abstractions has been developed. The description of these abstractions, an initial performance assessment and their functional evaluations are reported as well in this deliverable.

The final description of the developments implemented in each of the programmable platforms, together with the performance evaluations and will be provided in deliverable D3.3, with due date November 2019.

1 Introduction

As stated in deliverable D3.1, the 5G-PICTURE vision of a disaggregated hardware/software (HW/SW) allocation of network functionalities in different nodes of the network providing different resource types requires a significant increase in the programmability and flexibility of these nodes. We focused on the development of programmable platforms as an enabler to deploy into the network different functionalities that can be specified using certain high level description and be deployed at run-time on specific platforms. The network operating system should be able to estimate the performance achievable with the different platforms and to allocate in the most suitable node the network function that must be executed. This approach will permit the 5G network to support a wide set of complex network functionalities, which can be dynamically changed and moved among different network nodes depending on the network conditions (traffic, faults, link availability, etc.) and on the required Service Level Agreements (SLAs). Furthermore, programmability is also a mandatory element to permit the adoption of new protocols/services that could come up in the future.

The other element on which the WP3 tasks are focused on is efficiency of the developed programmable platforms. While it is fairly simple to provide programmability and flexibility sacrificing performance, it is much more challenging to achieve a greater programmability level without sacrificing the performance. Therefore, the 5G-PICTURE project focused the effort of WP3 in designing efficient programmable platforms. In the previous deliverable several common aspects that the programmable platforms of the different network domains (ethernet, optical, and radio access) has been identified: (i) the programmable network platforms must provide clear programming models that will allow development of network functions (NFs), decoupling the definition of the function from the specific platform-dependent implementation. Possible programming models are for example the Domain Specific Languages (DSLs), such as the P4 language for programmable dataplanes or the OpenCL language for the definition of digital signal processing (DSP) radio functions; (ii) the control/data plane separation enabled by the use of SDN technologies that is dominating the wired network scenario can be extremely useful also in the radio and optical network domains; (iii) a set of hardware abstractions and/or interfaces are needed to provide an API that can be used for configuring the programmable platforms.

This deliverable presents the current status of the work related to the development of the interfaces, programming models, and hardware abstractions of the programmable network platforms that are under development in the 5G-PICTURE project.

Organisation of the document

This deliverable is structured as follows: Section 2 presents the detailed functional definitions of the programmable platforms that are under development and an initial evaluation of the performances achievable with the proposed platform. A definitive assessment of the performances will be provided in the deliverable D3.3.

Section 3 presents the hardware abstractions that are used to decouple the details of the programmable platforms implementation from its configuration capability.

Section 4 illustrates the different hardware technologies developed in 5G-PICTURE that provide basic building blocks of the 5G network architecture, comprising novel optical as well as RF and baseband (BB) processing technologies.

Finally, section 5 contains the summary and conclusions.

2 Functional definition of programmable platforms: implementation results and preliminary evaluation

This section reports the progresses of Task 3.1. The main goal of this task is to emerge with domain-specific processors which pre-implement (very efficiently, e.g. in HW) functional primitives reusable by multiple network functions. In the first phase of Task 3.1 the consortium focused on the architectural definition of the programmable platforms. The second phase of this task focused on the detailed implementation of these platforms and an initial evaluation of the performance achievable with the associated technologies. In particular, for the backhaul (BH), the platforms will provide a highly *programmable stateful dataplane* with latency and throughput performances able to sustain the need of the 5G network. For the fronthaul (FH), reconfigurable HW platforms will be used to provide both *SDN-based programmability* and efficient allocation of the processing tasks with flexible *functional splitting* between the radio units (RUs) and the base station (BS).

2.1 Time-Shared Optical Network (TSON) Platform

The 5G-PICTURE solution uses the Time-Shared Optical Network (TSON) technology [20], which provides programmable converged fronthauling/backhauling functions and offering suitable multi-technology, multi-protocol interfaces based on FPGA-based programmable HW.

2.1.1 Generic TSON functionality/architecture, capabilities, and evaluation results

This subsection briefly recall the generic TSON functionality already introduced in deliverable D3.1 [1] and presents the resource allocation capabilities and the initial TSON evaluation results. In addition, a description of the Real-Time Bandwidth Variable Optical Transmitter is provided in this section.

2.1.2 Generic TSON functionality/architecture

TSON is a multi-wavelength fully bi-directional synchronous and novel frame based flexible system. It delivers a flexible and statistically multiplexed optical network infrastructure able to support on-demand guaranteed time-shared multi-granular services. Its network implementation consists of field-programmable gate array (FPGA) optoelectronics platforms integrated with advanced optical components to enable high performance processing and transparent switching and transport [20] [21] [22]. TSON is a contention-less solution through the deployment of a central resource allocation engine of route, wavelength, and time assignment, responsible to set-up paths of sub-wavelength granularity.

The TSON solution includes two different types of nodes, the edge and the core nodes, incorporating different functionality and level of complexity. TSON edge nodes provide the interfaces between wireless, Passive Optical Network (PON) and Data Centre (DC) domains to the optical TSON domain and vice versa. Figure 2-1 shows the TSON edge architecture. The ingress TSON edge nodes are responsible for traffic aggregation and mapping, while the egress edge nodes include the reverse functionality. To send and receive data, each TSON edge node uses four SFP+ transceivers, two 1310 nm 10 km reach for end-point server traffic and control, and two DWDM 80 km reach transceivers at 1544.72 nm and 1546.12nm. The 1310 nm interfaces can be used to support both data and control traffic either separately or combined depending on whether out-off band or in-band control is adopted. For more information on the TSON overall architecture and interfaces the interested reader is referred to 5G-XHaul deliverable D2.2 [23].

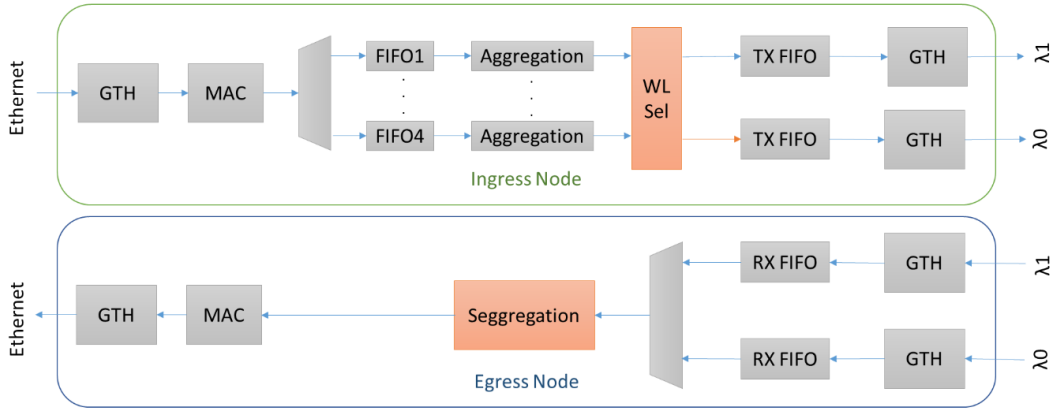


Figure 2-1: TSON edge node architecture.

When the ingress part of the edge node receives traffic, the FPGA waits to finish the processing of the current frame, and then starts to transmit time-slices by assigning optical bandwidth to these. The optical bandwidth allocated to the different services is not fixed but can be elastically defined based on the requirements of each service. Therefore, TSON can support elastic time and optical bandwidth allocation. As an example, in Figure 2-1, when the input traffic comprises Ethernet frames at the ingress part of the edge node, the Ethernet frames received at the 10 Gb/s receiver are passed to the 10GE Medium Access Control (MAC); Then MAC discards the preambles and Frame Check Sequence (FCS), transmits the data to the Receiver (RX) in a first in first out (FIFO) manner and indicates whether the packet is good or not; the RX FIFO receives the data, waits for the good/bad indication from the MAC, and sends it to the DEMUX block if there is any valid data. The DEMUX analyses the Ethernet frame information (i.e. Destination MAC address, Source MAC address, etc.) and puts them in a different FIFO. After that, the FIFO does not send any data until the AGGREGATION gives a command; the register file of AGGREGATION, containing the Time-slice Allocation information, is updated by the Lookup Table (LUT) (this table stores information related to time-slice Allocation and the fast-optical switch that is incorporated in the edge TSON node). The AGGREGATION module waits until the burst-length Ethernet frames are ready in the FIFO and the time-slice allocation is available, then it transmits the bursts with a suitable wavelength through the TX FIFO. For the egress part of the edge TSON node, when the 10 Gb/s receiver receives a burst (time-slice), it drops the burst in the RX FIFO Lambda0/Lambda1; after the burst is completely received, the SEGREGATION block segregates the burst to Ethernet frames and transmits them to a TX FIFO. Every time the TX FIFO receives a complete Ethernet frame it sends it to the 10GE MAC. Finally, MAC passes the data to the 10 Gb/s transmitter and transmits them out.

The TSON core nodes do not carry out any data processing but need to switch the traffic optically. Therefore, the FPGA-based TSON core node controls the fast-optical switches to setup the path with a client's request. These nodes switch transparently the optical frames to the appropriate output port utilising the fast-optical switching which in the core node as was also in the edge node case. The TSON core nodes adopt the wavelength selective architecture and as such require one switch per wavelength, to direct the incoming optical time-sliced signals towards the appropriate output ports, as defined by the control plane. The dimension of the space switch is defined by the number of fibres that are interconnected through the node. The TSON core node uses the same type of high performance FPGA boards for the ((Pb,La)(Zr,Ti)O3) (High-Speed PLZT Optical Switches) control. The FPGA LUTs are filled in by the control plane, through customised Ethernet communication carrying PLZT switching information, to be able to change the switch state per time-slice on the PLZT switches with the aim to establish and maintain optical paths across the TSON domain. The basic functions for the operation of TSON domains have been implemented in internal modules, within the Software Defined Networking (SDN) controller, that cooperates for the on-demand provisioning of connectivity between TSON core and edge nodes. The interested reader can find a detailed description of the TSON control plane in the 5G-XHaul deliverable D3.1 [24].

TSON works in two different modes: Ethernet and Elastic bandwidth allocation mode. The Ethernet mode is for Ethernet-based packets without any time-slicing. Elastic bandwidth allocation is based on time-slicing as described earlier in this section. Even though the latest TSON implementation is based on Xilinx Virtex7 board (156.25 MHz clock frequency), supporting multiple 10 Gb/s (for control and transport) DWDM SFP+ transceivers, the architecture can support rates beyond the 10 Gb/s (i.e. 25, 40, 100 Gb/s). For the optical layer, TSON relies on fast optical PLZT switches [22] having 10 ns switching speed as well as a set of active and passive components including Erbium Doped Fibre Amplifiers (EDFAs), MUX/DEMUXes, etc. TSON is designed and implemented as a novel frame-based, time multiplexing network solution, offering dynamic connectivity with fine granularity of bandwidth.

Although natively TSON allows handling of Ethernet frames, its configuration can support a broad range of framing structures and communication protocols including CPRI and Open Base Station Architecture Initiative (OBSAI), either natively or through their packetised versions.

2.1.3 TSON resource allocation capabilities

TSON is fully SDN enabled, and the parameters of TSON nodes are programmable by the controller, such as Quality of Transmission (QoT), overhead that is programmable from 0 to 39 KB, time-slice size that is programmable from 80 B to 31.25 KB, time-slice numbers in a frame that are programmable from 4 to 100, and time-slice allocation. Thus, when the client (e.g. operator, service provider) requests to setup a network service, after evaluating its requirements such as bit-rate, connectivity type, Quality of Service (QoS), and QoT by the network controller, the TSON nodes chose the corresponding QoT overhead size, Time-slice size, time-slice

number and time-slice allocation to fulfil such requests to compose a dedicated network function slice of the node. In addition, TSON supports programmable traffic flow control (i.e. VLAN, Dest MAC, Src MAC).

TSON offers a hierarchy of three levels of resource granularity: connections, frames, and time-slices, as illustrated in Figure 2-2. Connection refers to a sub-wavelength light path establishment between any two end points in the TSON domain. To improve statistical multiplexing of data units, each connection lasts for a number of frames and the minimum duration of 1ms. Each frame is divided into time-slices as the smallest units of network resources, i.e. the actual sub-lambda resources. The frame length and the number of time-slices inside a frame define the minimum granularity achievable by the TSON network [23]. The TSON framework offers a very flexible optical platform that supports sub-wavelength switching, frame lengths, varying from 64 ns to 25.6 µs and variable bit rates, spanning from of 30 Mb/s up to several Gb/s, with 30 Mb/s step.

2.1.4 Generic TSON Evaluation

Figure 2-3 and Figure 2-4 show the TSON evaluation results obtained using Ethernet as an interface technology. This evaluation is based on latency and jitter analysis.

Figure 2-3 displays the impact of latency as a function of time-slots. As expected, the latency increases as the time-slice size increases because a higher number of packets are accumulated in a given time-slot. However, when the TSON node is in the Ethernet mode the minimum latency is 1.747 µs, while when it is in Elastic bandwidth allocation mode, the latency ranges from 9.234 µs to 118.233 µs.

On the other end, Figure 2-4 shows the impact of jitter as a function of time-slices. The frame size is set to 1500B with 5 Gb/s traffic. As the time-slice size grows, the percentage of traffic not received in a given time (e.g., 2 µs) decreases, but the percentage of traffic with more latency increases, thus increasing the jitter.

The minimum latency is 1.747 µs, while, in Elastic Time Division Multiplexing (TDM) mode, the latency ranges from 9.234 µs to 118.233 µs when the node is in Ethernet mode. In addition, as the time-slice duration increases, so does the jitter, and the number of packets not received at certain times increases.

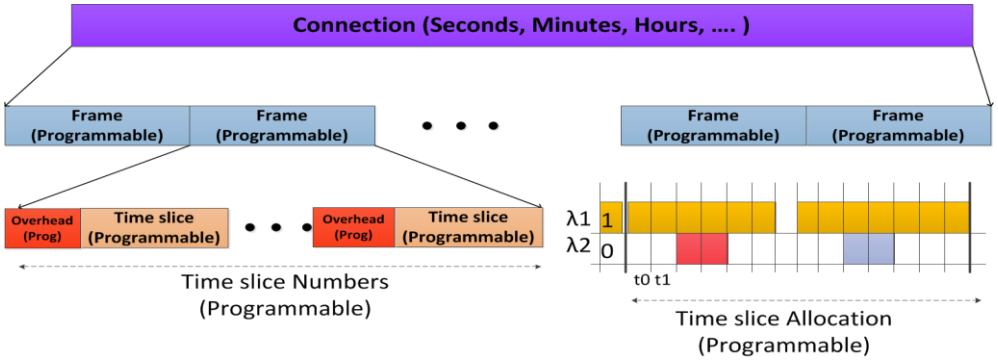


Figure 2-2: Structure of connection, frame and burst.

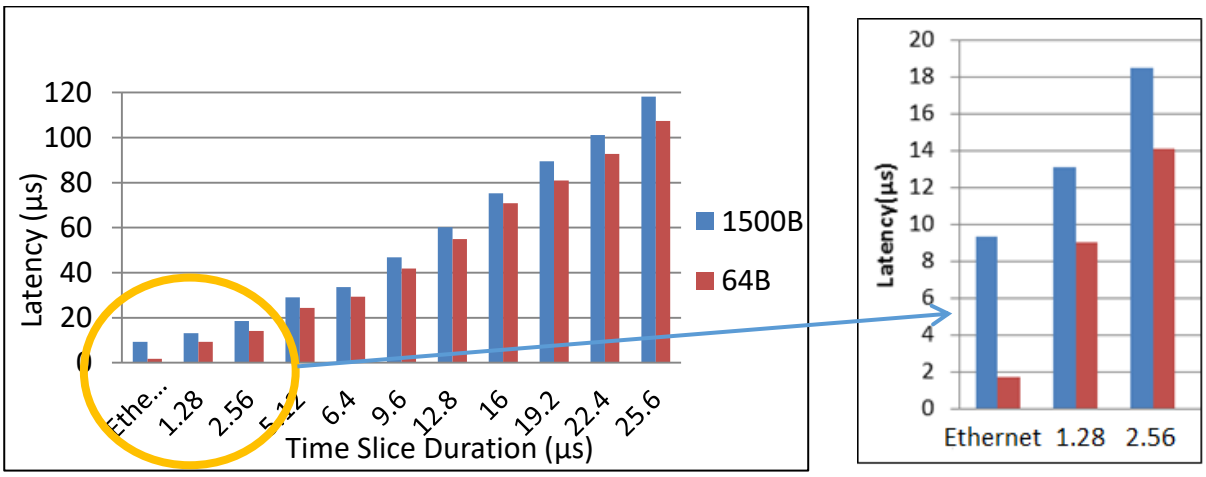


Figure 2-3: Latency vs. Time slice duration (1500B, 5 Gb/s traffic).

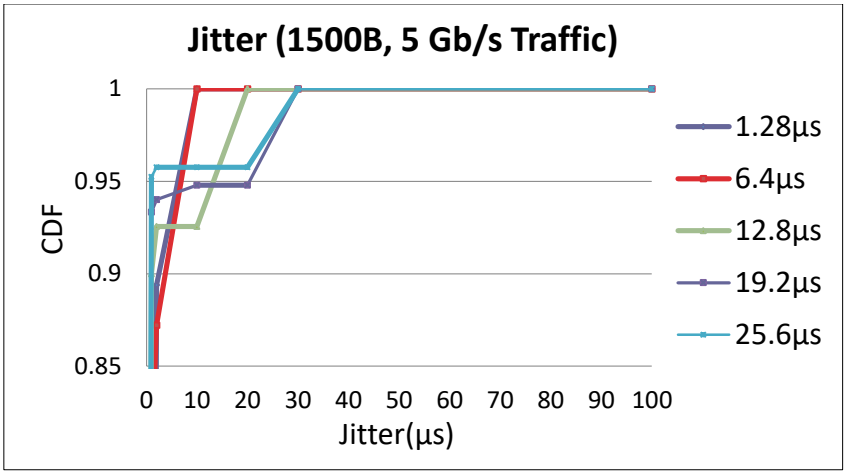


Figure 2-4: Jitter (1500B, 5 Gb/s traffic).

2.2 TSON extensions provided in support of FH and BH services

TSON was proposed as a suitable optical transport technology in support of BH and FH services. However, the initial TSON implementation needed several extensions to become suitable to support FH and BH services. These extensions that are currently being further enhanced were originally implemented in the framework of the Phase 1 5G-PPP project 5G-XHaul and included synchronisation, support of multi Ethernet clients for backhaul, and CPRI integration for FH services.

2.2.1 TSON Synchronisation

To achieve high synchronisation accuracy over the network, accurate timestamps are required. Although TSON has its own synchronisation scheme [23], the TSON network requires global frame synchronisation among TSON nodes to meet the precision needed in the 5G-PICTURE project environment. This solution takes advantage of the IEEE-1588 v2 protocol [25] across heterogeneous domains. This solution needs to be at least a Transparent Clock (TC) for a given domain/node. This introduces the following requirements:

- The domain/node can measure the residence time of a 1588v2 packet inside the domain/node. This indicates: 1588 packets are recognised at the ingress, a timestamp is generated in the node, at the egress port the time is again measured, a residence time is computed "as egress-ingress time".
- The residence time is copied in a subsequent "Follow up" packet sent by the 1588 Master clock node.
- The synchronisation end-point will subtract the residence time from the follow up to this measurement of RTT in order to remove per domain jitter.

Applying the approach described above to a TSON domain means that the residence time is computed at the egress TSON node as the difference between its current time and the time when the packet enters the TSON ingress source node. Therefore, the TSON nodes need to be synchronised being, in principle, a different synchronisation to that of 1588 v2. It is a synchronisation local to TSON that requires it to be able to compute the residence time in a meaningful way. In this scenario, the ingress time needs to be conveyed to the TSON egress node. Therefore, the time stamps need to be transmitted in-band to calculate the residence time.

As TSON is FPGA based programmable HW, Xilinx 10 Gigabit Ethernet Subsystem IP core [26] supports high accuracy IEEE-1588-v2 1-step and 2-step timestamping on a 10GBASE-R network interface. Figure 2-5 shows the block diagram of the 10G Ethernet MAC subsystem. The subsystem IP core has three main sub IP cores: a MAC, a Physical layer - Physical Coding Sublayer (PCS)/ Physical Medium Attachment (PMA), and a Timer for synchronisation. The time stamper unit for the transmit-side and the receiver-side are implemented inside the MAC and the PCS/PMA sub IP cores, respectively.

This subsystem synchronises the 1588 system timer in the selected format from the system timer clock domain into the subsystem clock domain for each of the transmit and receive data paths. This guarantees high accuracy for the 1588 timestamps. The selected format is available in one of two Time-of-Day (ToD) and Correction Field timestamp formats. The subsystem supports both In-band and out-of-band timestamping methods. The system timer could be delivered to the FPGA board via available SubMiniature version A (SMA) user clock on the board.

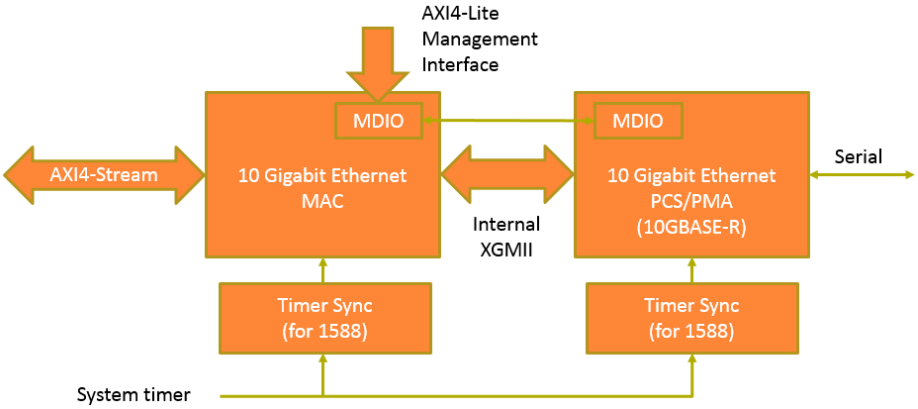


Figure 2-5: 10 Gb Ethernet High-Level Block Diagram.

Xilinx subsystem IP core does not support multiple IEEE-1588 v2 enabled subsystem instances [27]. In addition, as we mentioned before, the residence time needs to be computed at the egress TSON node as the difference between its current time and the time when the packet enters the TSON ingress source node. Therefore, the TSON nodes need to be synchronised and this in principle is different synchronisation than the synchronisation carried out by IEEE-1588 v2 even though the timing format – i.e. Time of Day (ToD) and Correction Field – is the same as in IEEE-1588 v2.

To resolve this problem and the Xilinx IP core limitation, a new subsystem, called HPN subsystem, has been developed to support multiple IEEE-1588-v2 enabled subsystem. As it was mentioned before, the Xilinx approach uses a time stamper unit for the transmit-side and the receiver-side inside the MAC and the PCS/PMA sub IP cores, respectively. Unlike the Xilinx approach, the developed HPN subsystem uses a separate developed time stamper. Figure 2-6 shows the HPN subsystem architecture. The HPN time stamper follows the same features as the Xilinx subsystem. The time stamper unit is located between the MAC and PCS/PMA IP cores, uses the Timer Sync clock and follows the IEEE 1588 protocol. In addition, the time stamper considers the physical layer delay for stamping. Figure 2-7 shows the captured waveform of both transmit and the receiver sides after time stamping for loop-back case. In this scenario, the ingress and egress time stamps are both the same as the ones we used in the loop-back approach and therefore the residence time is zero.

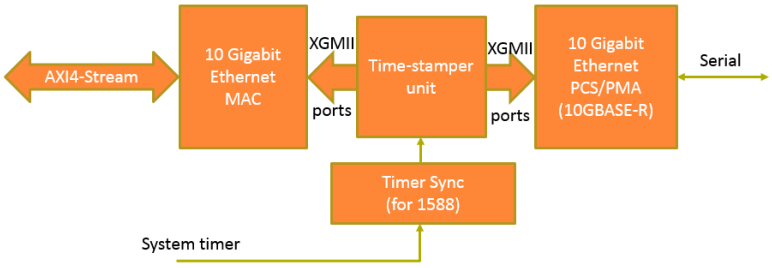


Figure 2-6: HPN Subsystem for synchronisation.

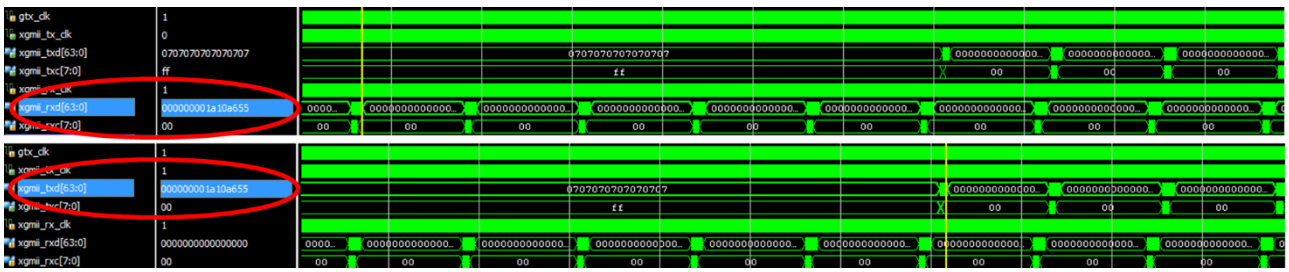


Figure 2-7: Time stamp with HPN Subsystem for ingress and egress source node for loop-back scenario.

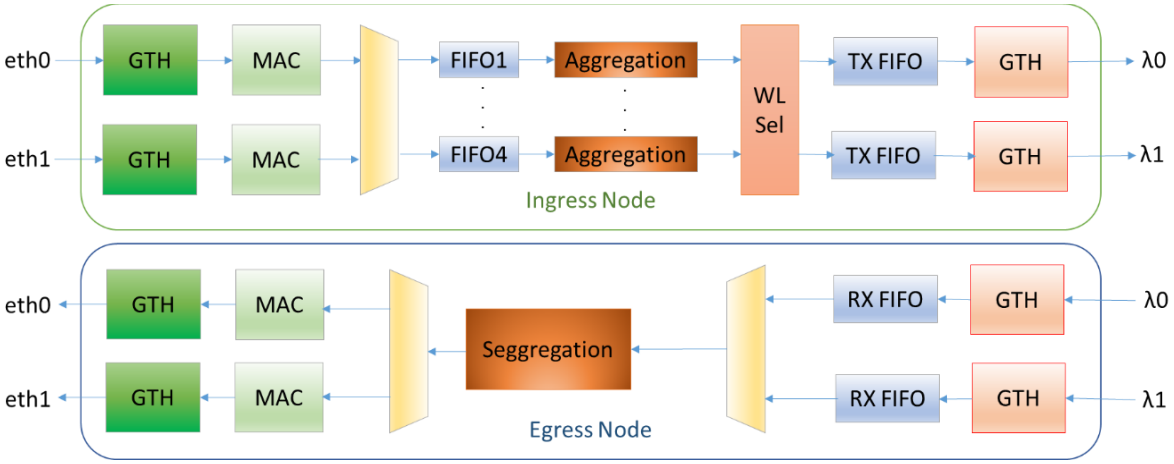


Figure 2-8: TSON extension for backhaul services with multi Ethernet clients.

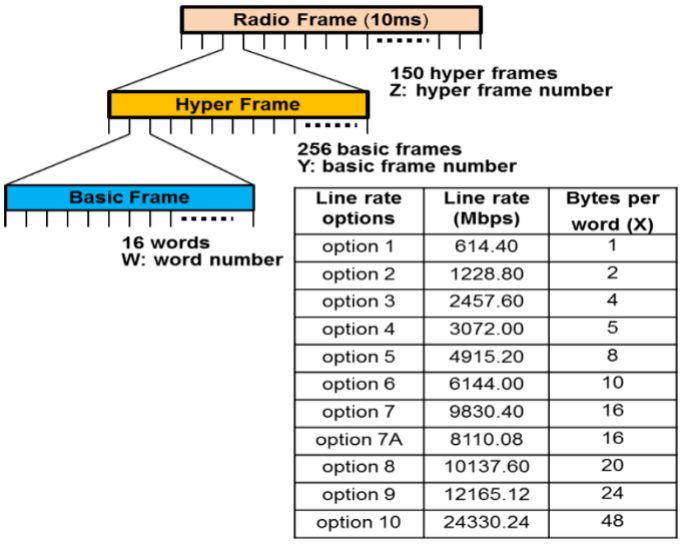
2.2.2 Backhaul Services over TSON – Supporting Ethernet

Extension of the TSON clients are only limited to the number of available transceivers on FPGA boards. We extended the TSON Ethernet client ports to support BH services.

Figure 2-8 shows the extended TSON solution to support BH service requirements in 5G. The multi-line client TSON follows the one client TSON legacy when the clients are using a different wavelength (i.e. λ). In addition, the Ethernet clients can use a common wavelength if the total bandwidth of the clients is less than 10 Gb/s. In this scenario, the packets are delivered to the related Ethernet clients at the egress nodes based on their header information (VLAN tag for 5G-XHaul [24]).

2.2.3 FH Services over TSON – CPRI extension

Figure 2-9 shows the CPRI frame structure that sends sampled In-phase Quadrature (IQ) data in a frame format. The CPRI radio frame is 10 ms. CPRI line rate information is sent in Z.Y.W.X format between the RU and the Baseband Unit (BBU), where Z is the hyper frame number, Y is the basic frame within a hyper frame, W is the word number within a basic frame, and X is the byte number within a word. A single basic frame duration is 260 ns (1/3.84 MHz) which is compatible to a Universal Mobile Telecommunications System (UMTS) chip length. Each basic frame consists of 16 words, and the word length depends on the CPRI line rate: 256 basic frames make a hyper frame, and 150 hyper frames make a radio frame. CPRI supports topologies such as tree, ring, and chain, each link between RU and BBU is a fixed-bandwidth TDM connection.



Line rate options	Line rate (Mbps)	Bytes per word (X)
option 1	614.40	1
option 2	1228.80	2
option 3	2457.60	4
option 4	3072.00	5
option 5	4915.20	8
option 6	6144.00	10
option 7	9830.40	16
option 7A	8110.08	16
option 8	10137.60	20
option 9	12165.12	24
option 10	24330.24	48

Figure 2-9: Frame Structure.

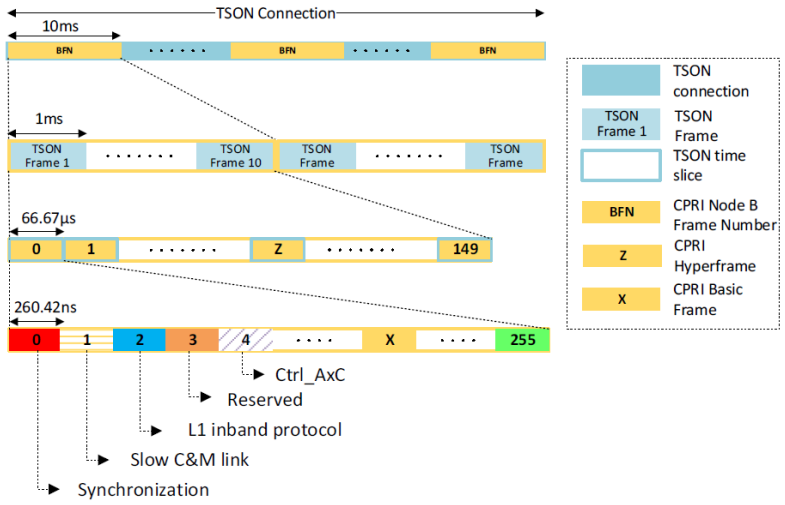


Figure 2-10: CPRI frame structure over TSON.

TSON's extension to enable native CPRI can support up to line rate option 5 with the current implementation as in 5G-XHaul the BBU and the Remote Radio Head (RRH) are using this option. The higher line rate option could use the same methodology for the TSON-CPRI integration. TSON can support varying service related requirements as its operational characteristics can be dynamically modified. Figure 2-10 illustrates how a heavy CPRI flows can be embedded into the TSON framing structure.

We have configured the Xilinx GTH Transceiver IP core for the CPRI protocol option 5. The reference clock frequency and data width for this IP core is 122.88 MHz and 32-bit respectively. The IP configuration uses 8B/10B encoding/decoding. Therefore, 4-bit control signals are available to indicate if data are special characters (comma symbols) or regular data for both the receiving and transmitting side. Comma symbols are used for synchronisation. The term of synchronisation here refers to finding the alignment of the 8b/10b codes within a bit-stream. Figure 2-11 shows the TSON edge node and CPRI integration high level architecture. In this architecture, the CPRI transceiver receives the CPRI frames and passes them to the *CPRI_TSON* interface unit in the ingress side. The *CPRI_TSON* interface is responsible for the Clock Domain Crossing (CDC) from 122.88 MHz to 156.25 MHz, which involves design the clock, and mapping the 32-bit data and 4-bit control to 64-bit corresponding to the TSON data width. Once the CPRI frames reach the aggregation block, according to the LUT information of the Time-slice Allocation, the aggregation block calculates the frames needed to construct the burst, aggregates the frames, waits for the valid Time-slice, and finally sends the burst into a different wavelength FIFO. The egress side has the reverse functionality.

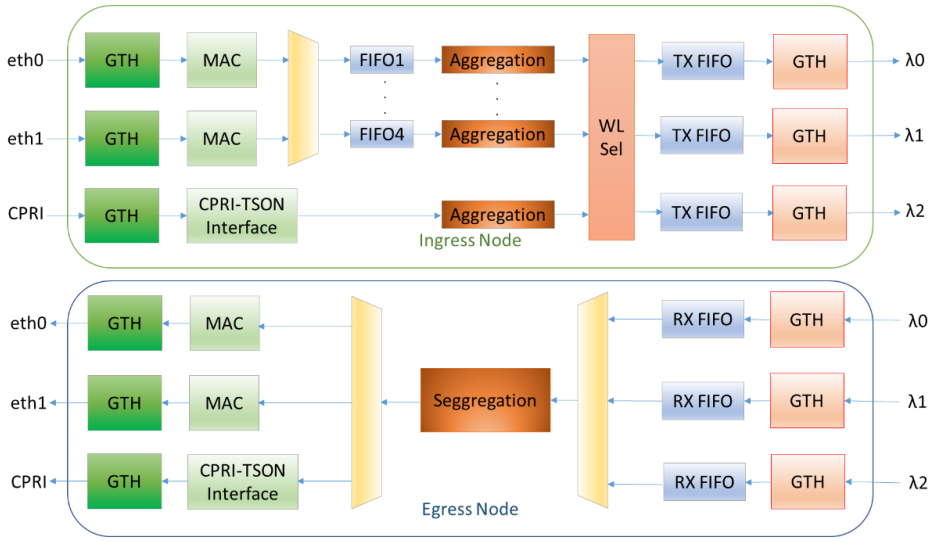


Figure 2-11: TSON edge node and CPRI integration high level architecture.

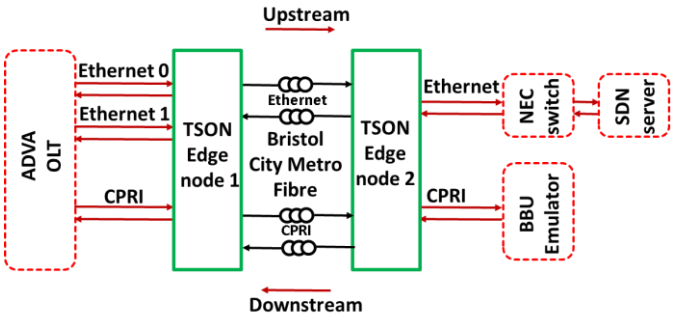


Figure 2-12: TSON 5G-XHaul implementation architecture for 5G-XHaul final demonstration.

2.2.4 TSON evaluation results

The performances of the TSON architecture developed in the final 5G-XHaul project demonstration has been evaluated during the 5G-PICTURE project and the results are shown in this subsection.

Figure 2-12 shows the TSON architecture used in the final 5G-XHaul project demonstration featured a combination supporting both optical BH and FH services. The configuration contains two TSON edge nodes employing Xilinx VC709 evaluation boards. TSON edge node 1 is connected to an ADVA Optical Line Terminal (OLT) with three clients comprising two Ethernet and one CPRI ports and TSON edge node 2 is connected to the SDN server and BBU emulator with two clients including one Ethernet and one CPRI ports, respectively. In the upstream scenario, TSON Edge node 1 receives the Ethernet and CPRI traffic streams from the ADVA OLT, aggregates two Ethernet traffic streams into one flow and sends both CPRI and the aggregated Ethernet packets to the TSON Edge node 2. TSON Edge node 2 receives the aggregated Ethernet and CPRI traffic and directs it to the SDN server and the BBU emulator, respectively.

In the downstream scenario, TSON edge node 2 receives Ethernet and CPRI traffic from the SDN server and the BBU emulator and sends them to the TSON edge node 1. TSON edge node 1 receives the traffic, the Ethernet traffic is segregated to two ports, based on the VLAN and CPRI traffic it maps the CPRI client, then both Ethernet and CPRI traffic is sent to the ADVA OLT.

In the 5G-PICTURE project we have evaluated the TSON implementation architecture for both FH and BH services. Three different scenarios are considered for the TSON experimental evaluation. The first scenario includes both Edge nodes connected back-to-back with short fibres. In the second and third scenarios, the proposed technologies are evaluated over the Bristol City Metro Fibre with 8 km and 16 km, respectively, of standard single-mode fibre (SSMF).

The Ethernet performance parameters under consideration include Bit Error Rate (BER) and latency. A traffic analyser (Anritsu MT1100) generates/receives two Ethernet traffic streams to/from the TSON edge node 1 and 2 at 4.4 Gb/s with fixed frames of 1500B length. Figure 2-13 shows the Ethernet BER measurements as a function of received optical power for the different considered scenarios. The BER curves show that the penalty observed for the case of 8 km of SSMF transmission over the Bristol City Metro Fibre Infrastructure compared to the back-to-back (B2B) performance is negligible. A penalty lower than 1dB is observed for 16 km transmission. Table 2-1 displays the upstream/downstream latency for Ethernet scenario. TSON edge node worst-case latency is less than 4%.

The CPRI performance parameters considered include Line Code Violations (LCV) (8B/10B decoding) ratio and latency. The traffic analyser generates/receives CPRI traffic to/from the TSON node 1 and 2 at 4.9152 Gb/s. Figure 2-14 demonstrates the CPRI LCV measurements as a function of received optical power for the different scenarios considered. The LCV curves reveal that the penalty for both 8 km and 16 km compared to the B2B case is lower than <1dB.

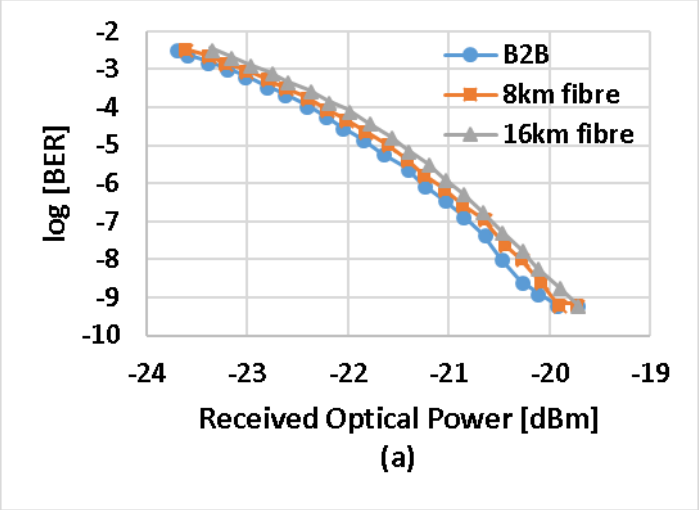


Figure 2-13: Ethernet BER measurements.

Table 2-1: Ethernet upstream/down-stream latency.

Destination	Ethernet Latency [µs]
B2B	1.57
8km	43
16km	84.54

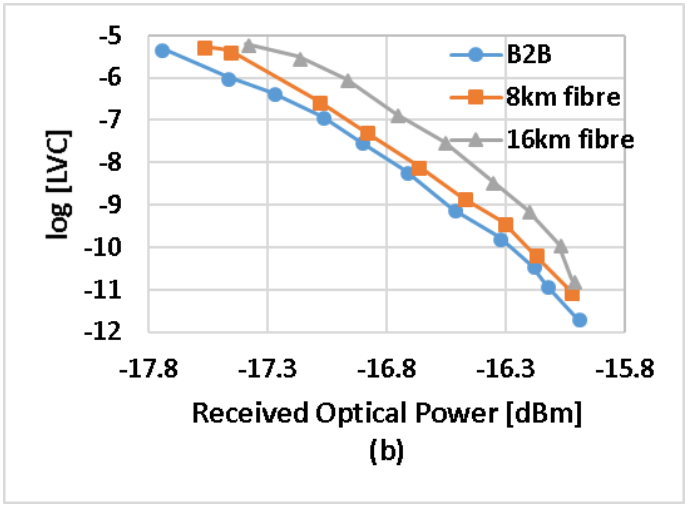


Figure 2-14: CPRI LCV measurements.

The VC709 includes a Silicon Labs Si5324 jitter attenuator on the board. The Si5324 is used for CPRI clock recovery from a user-supplied SFP/SFP+ module and use the jitter-attenuated recovered clock to drive the reference clock inputs of a GTH transceiver. Figure 2-15 illustrates the TSON node 1 CPRI reference clock and TSON 2 recovered clock. As observed in Figure 2-15, the CPRI clock is successfully recovered. Figure 2-16 shows the CPRI recovered clock spectrum.

Table 2-2 shows the upstream/downstream latency for CPRI scenario. TSON edge node worst-case latency is less than 1.5%.

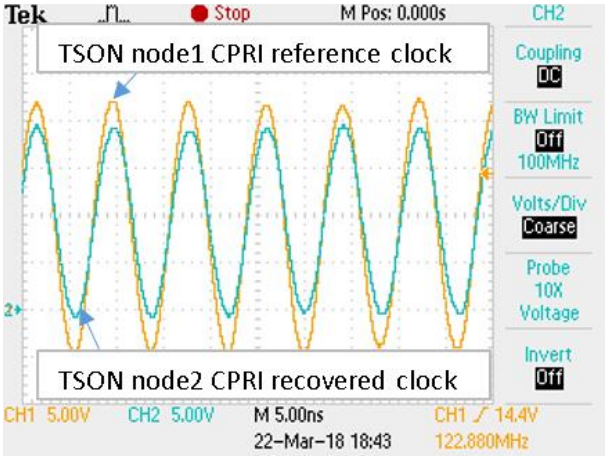


Figure 2-15: TSON node1 CPRI reference clock and TSON node2 CPRI recovered clock.

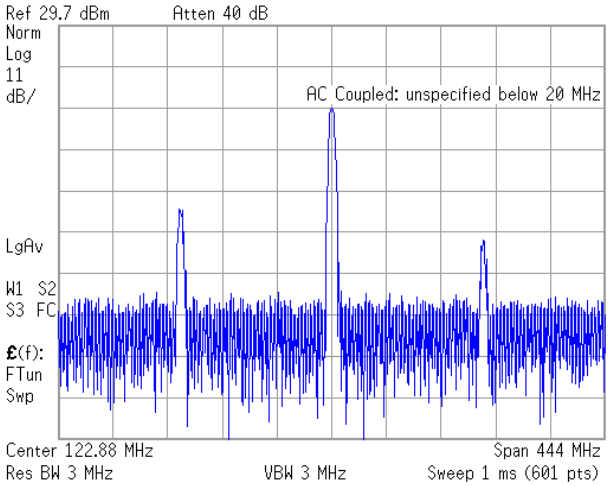


Figure 2-16: CPRI recovered clock spectrum.

Table 2-2: CPRI upstream/downstream latency.

Destination	CPRI Latency[µs]
B2B	0.45
8km	42.04
16km	83.38

2.3 BVT and TSON extensions for 5G-PICTURE

In this section, we present our bandwidth variable transmitter (BVT) based on a real-time modulation-format adaptable transmitter (RMAT) exploiting a high-performance FPGA. Both the FPGA and the transmitter are controlled by a local agent to adapt the optical signal modulation format. In addition, we introduce the TSON extension for 5G- PICTURE.

2.3.1 Real-Time Modulation-Adaptable Transmitter for 5G-PICTURE

The implementation of a fast-tunable bandwidth variable transmitter (BVT) based on a modulation-adaptable transmitter is shown in Figure 2-17. In this scenario, 10 Gb/s input port traffics are aggregated to 100 Gb/s port using an FPGA. The aggregated traffic passes to a modulation-adaptable transmitter and core network responsibly. Both the FPGA and the transmitter are controlled by a local agent (RMAT agent) to adapt the optical signal modulation format. The modulation-adaptable transmitter can be potentially integrated with an SDN controller.

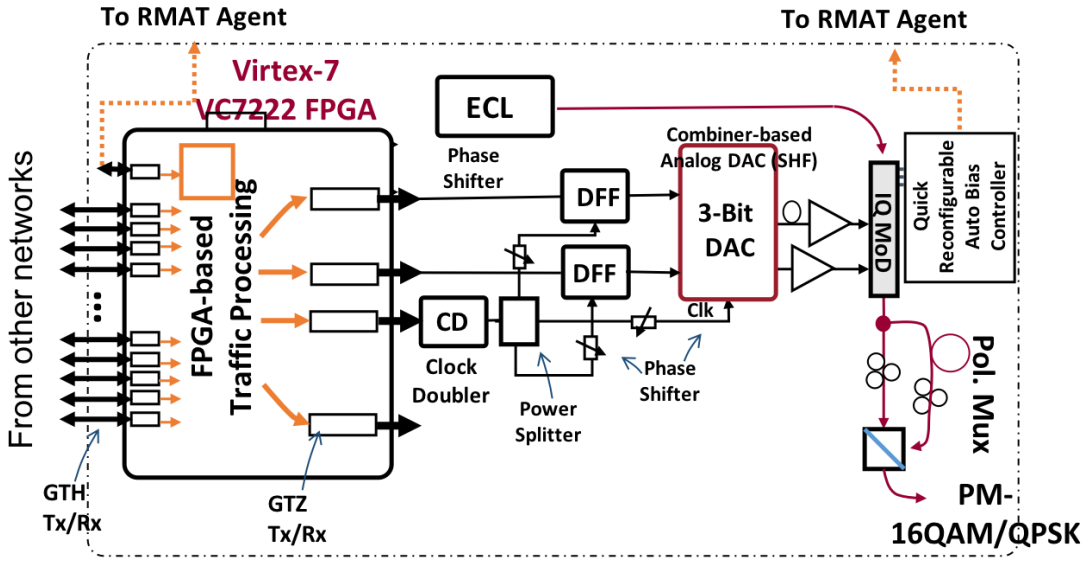


Figure 2-17: Implementation of the modulation-adaptable transmitter.

We implemented the BVT exploiting modulation format adaptability. Three serial GTZ transmitters drive the modulation-adaptable transmitter. Among these, two GTZ transceivers handle the data, while another GTZ provides a half clock, which is doubled by a clock doubler (CD) to synchronise two flip-flops (FF) and a 3-Bit Digital-to-Analogue Converter (DAC). The FFs are used to improve the signal quality. After regeneration, two data streams are sent to a 3-Bit DAC (from SHF) to obtain multiple level signals. The 3-bit DAC is an analogue device based on power combiners, which could provide wide bandwidth and high signal quality with precise phase match. In the current setup, we only use 2 bits for the transmitter. Therefore, the BVT generated 26 Gbaud QPSK/16QAM signals with the required OSNR to achieve BER = 3.8E-3 (7% FEC threshold) about 20.3 dB and 15.36 dB for PM-16QAM and PM-QPSK signals. Further work has been carried out for spectral-efficiency tunable transmitter based on probabilistic shaping, upgrading the transmitter to support PAM BVT for Metro networks, and design and implement a receiver for it.

2.3.2 TSON extensions for 5G-PICTURE

Over the course of 5G-PICTURE, we are taking the advantage of TSON technology and extending it to support following features:

- Transferring the technology to a new FPGA board to support a higher number of I/O ports compared to the previous implementation. According to the previous requirements, we choose NetFPGA SUME and Xilinx vc709 evaluation board for the implementation of TSON technology. AS an example, VC709 contains 4 X SFP+ (4 x 10 Gb/s) interface and we added 8 x Gb/ss ports by connecting an FMC card to this board. The new implementation targets support of several 10 Gb/s, 25 Gb/s, and 100 Gb/s ports.
- The 5G-PICTURE TSON implementation supports convergence from 10 Gb/s interfaces to the 25 Gb/s/100 Gb/s domains.
- The eCPRI protocol will be integrated into TSON technology.

Deliverable D3.3 will report on the TSON extensions addressing these aspects and the evaluation of the associated performance.

2.4 Open Packet Processor (OPP)

CNIT is developing a programmable packet processing pipeline that will be able to execute several stateful network functions (NFs) without the intervention of the control plane. The architecture of the OPP[1] packet

[1] OPP will be presented at the next USENIX Symposium on Networked Systems Design and Implementation with the paper “FlowBlaze: Stateful Packet Processing in Hardware”. In this deliverable we decided to leave the name of OPP for consistency with the other 5G-PICTURE deliverables.

processing pipeline has been thoroughly described in deliverable D3.1. Here we simply recall the main characteristic of the programmable dataplane.

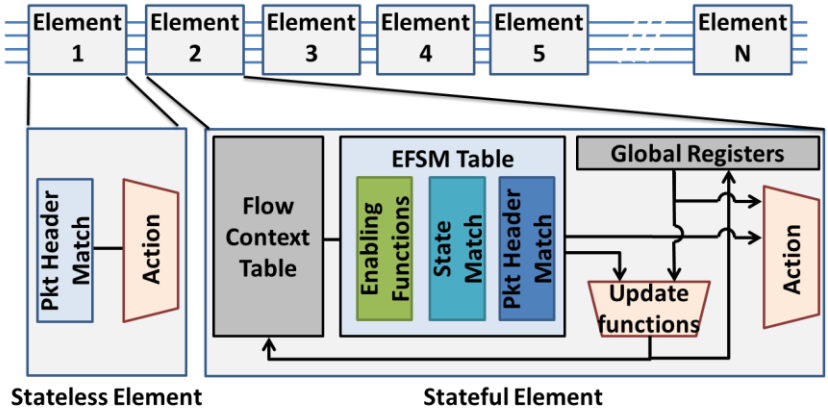


Figure 2-18: OPP machine model.

Figure 2-18 depicts the OPP pipeline. Any element of the pipeline can be configured to act as a stateless block (similar to a standard OpenFlow table) or as a stateful block. The stateful blocks execute network functions that can be expressed as per-flow EFSMs (Extended Finite State Machines). The main blocks of the stateful elements are the *Flow Context Table* where the flow context (a set of registers) is stored, the *enabling functions* block used to check the conditions that triggers the state transitions, the block that actually implements the FSM, and the *Update functions* and *Action* blocks that modify the register values and the packet headers, respectively. In the following, the details about the hardware implementation of the OPP pipeline and the initial performance evaluation carried out on several blocks composing the pipeline.

2.4.1 Hardware Implementation

Our implementation is based on the NetFPGA SUME [19] SmartNIC, an x8 Gen3 PCIe adapter card containing a Xilinx Virtex-7 690T FPGA and four SFP+ transceivers providing four 10G Ethernet links. The system is clocked at 156.25MHz and designed to forward 64B minimum-size packets at line rate. We synthesised OPP using the standard Xilinx design flow. Our prototype fixes the machine model's parameters as in Table 2-3 and uses a non-programmable packet parser, a configurable size flow context table, and a fixed-size EFSM table.

The Flow Context Table is implemented with BRAM blocks. Each entry has 128b for the flow key and 146b for the value (16b of state label plus 4 registers of 32b and 2b acting as internal flags). The EFSM table is implemented by a small TCAM of 32 entries of 160 bits. The scheduler block has a single queue for up to 20 packets and is enough to provide the required throughput and forwarding latency for the tested workloads. We studied the implications of different queue sizes and scheduler configurations in [18]. Table 2-4 lists the logic and

Table 2-3: Parameters of an OPP stateful element in our hardware implementation.

Parameter	Value	Description
K	4x32b	Flow context's registers
M	8	Maximum number of conditions
Z	64b	Size of metadata moved between elements
H	8x32b	Global registers
ALUs	5	The maximum number of ALUs dictates the maximum number of update functions that can be performed for a given transition.

Table 2-4: NetFPGA's resource requirements for OPP with a single stateful element compared to those of a reference, single-stage Ethernet switch.

Resource type	Reference switch	OPP
# Slice LUTs	49436 (11%)	71712 (16%)
# Block RAMs	194 (13%)	393 (26%)

memory resources (in terms of absolute numbers and fraction of available FPGA resources) used by an OPP implementation with a single stateful element and a Flow Context Table with 16k entries. For reference, we also list those required for the NetFPGA SUME single-stage reference switch, i.e., a simple Ethernet switch. The reported resources include the overhead of several blocks, such as the microcontroller for the OPP configuration, the input/output FIFO for the 10G interfaces, etc., which are required to operate the FPGA and do not need to be replicated for each element. We successfully implemented on the NetFPGA SUME up to 6 stateful elements for a total of about 200k flow context entries, using around 57%of LUTs and 85% of BRAM blocks.

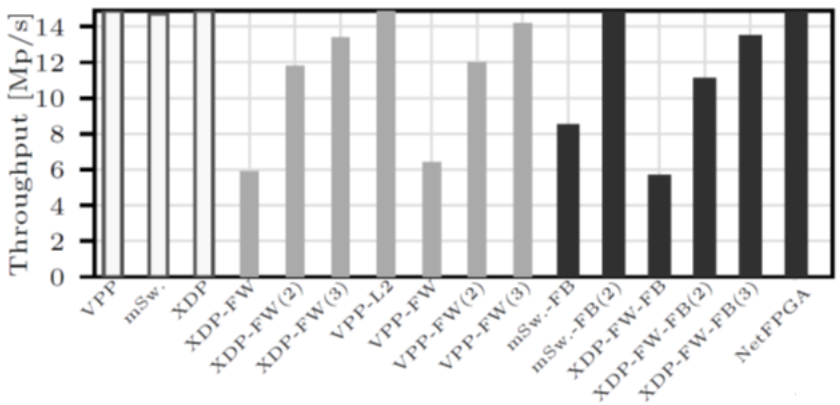
2.4.2 Evaluation Methodology

We experimentally measure the OPP performance with end-to-end tests and microbenchmarks, resorting to simulations to test corner case scenarios or to unveil details that would not be visible with black-box testing. To test the workload-dependent behaviour of the OPP we used a number of traffic traces collected at various operational networks. Here we report the results for publicly available traces selected from carrier networks (CHI15[2], MW15[3]) and from university data-centres(UNI1,UNI2)[4].

2.4.3 Throughput: end-to-end tests

We measure the end-to-end OPP throughput when running different applications using both our hardware and three software implementations, the VPP, mSwitch and XDP implementations. mSwitch and XDP are configured with the Big Flow Detector and UDP Stateful Firewall, respectively, both identifying flows by the 5-tuple. The same application is configured also on the NetFPGA implementation.

Figure 2-19 summarises the measured packet forwarding rates of minimum-sized (64B) packets with various systems. All the bare packet I/O frameworks (white bars) achieve line rate (14.88 Mpps) as they do not touch packet headers. When implementing packet processing logic on top of them, rates decrease (gray bars) and we need more CPU cores to reach the line rate. Implementing network functions with OPP (black bars) does not add much overhead, as we see by comparison between XDP-FW and XDP-FW-FB. Further, OPP scales well to multiple CPU cores (see XDP-FW-FB and mSw.-FB). Here, notice that the software implementations are forwarding only 4 flows, and thus we are showing a best case scenario for the achieved forwarding rate. The NetFPGA implementation can forward packets at 10 Gb/s line rate, and the performance is independent from the number of flows being forwarded. These results show that, even for a relatively simple use case, OPP can free several CPU's cores from network processing tasks.



Rate of bare packet I/O engines(white), stateful packet processing w/o FlowBlaze (lightgray) and that with OPP (dark gray): Numbers in () indicate the number of CPU cores if not 1. OPP does not add overhead (XDP-FW vs XDP-FW-FB) and it scales well (XDP-FW-FB vs XDP-FW-FB(2) and mSw.-FB vs mSw.-FB(2)). The NetFPGA implementation can always forward at line rate while saving up to 3 CPU cores.

Figure 2-19: Packet forwarding rates.

[2] CAIDA. http://www.caida.org/data/passive/passive_2015_dataset.xml

[3] MAWILab traffic trace - http://www.fukuda-lab.org/mawilab/v1.1/2015/07/20/20150720.html

[4] IMC 2010 dataset. http://pages.cs.wisc.edu/~tbenson/IMC10_Data.html

2.4.4 Latency

To measure the end-to-end latency over our NetFPGA implementation, we connect two 10G NIC regular ports to two NetFPGA ports; we then run a pkt-gen program on one regular port to send and receive packets while instrumenting another pkt-gen on the other regular port to echo back received packets. The measured latency includes two passages in the NetFPGA and the latency of the pkt-gen (with the overheads of moving packets through the PCIe bus 4 times), i.e., latency is measured as seen by the user-space pkt-gen application. We measured that OPP NetFPGA implementation adds only 2–9 µs to cable latency, and up to 1 µs to plain FPGA forwarding (no OPP). This is because OPP is optimised to process a packet in just 8 clock cycles. Here, notice that multiple pipelined elements may slightly increase the processing latency of OPP, with each element adding about 50 ns. Also, a full queue in the scheduler block may add up to 384ns of waiting time. In any case, even with these additional delays, OPP packet processing latency is well below the µs.

2.4.5 Packet Manipulator Processor

In this section we present the implementation details and the initial performance evaluation of the VLIW (Very Long Instruction Word) version of the PMP (Packet Manipulator Processor). The details of the V-PMP architecture of V-PMP have been reported in deliverable D3.1. The proposed CPU is based on a VLIW architecture and executes up to 8 instructions per clock cycle, providing a speed-up in the range 3x-6x for the three use cases taken into account.

The platform that has been used to develop and test the design of the V-PMP is the NetFPGA SUME, a PCI express card with a Xilinx Virtex-7 FPGA, and four 10 Gb/s interfaces. The architecture of the FPGA implementation is depicted in Figure 2-20.

The I/O queues and the input arbiter are part of the NetFPGA codebase. In particular, the input arbiter merges the 4 input streams from the 10 Gb/s interfaces into one output stream, in a round-robin fashion. The bus used to interconnect all the blocks of the data-path is the AXI Stream bus, which is typical for Xilinx platforms.

- **Input FIFO**
  This interface stores the packets coming from the input arbiter and allows V-PMP to access them. The peculiarity of this interface is in the way that access can be provided to it: from the input arbiter side, it is a standard FIFO whereas from the PMP side is a read-only RAM with 8 x 32 bits read ports, allowing the 8 lanes of the processor to load 256 bits from the input FIFO to its internal registers in a clock cycle. This interface has been designed to be integrated with the AXI-Stream bus, in fact carries, the packet data and some additional information such as the packet length, the ingress and the egress port. The interface exposes this information as memory mapped registers, allowing PMP to read this information and react accordingly.
- **Metadata RAM**
  This interface allows PMP to access header fields coming from an external parser (the MATs table of the switch pipeline for example) and/or populate empty fields on a packet template, such as an ARP reply packet. It is an 8 read/write port RAM, each port allowing the V-PMP and the external parser to load/store 32 bits per clock cycle. One additional port has been added in order to allow the Direct Memory Access (DMA) block to transfer data from this interface to the output one, moving 256 bits per clock cycle.

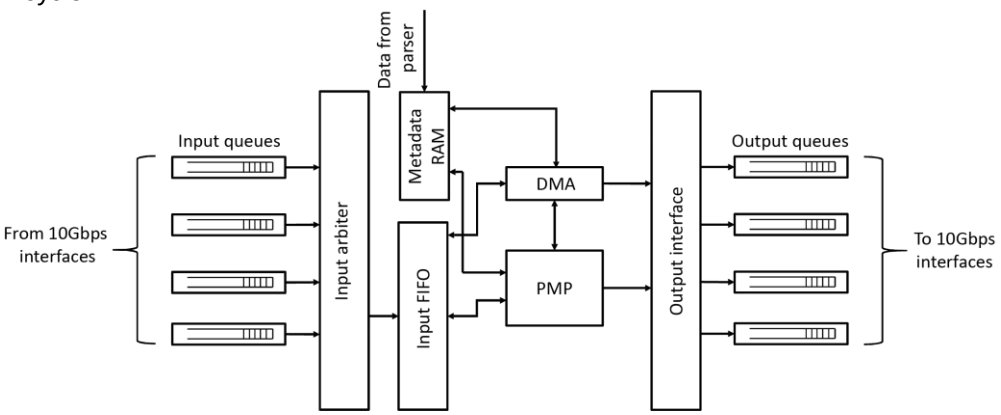


Figure 2-20: PMP ecosystem.

- **Output Interface**
  This interface allows the V-PMP to forge a packet and to send it on the AXI-Stream bus, exposing all the memory-mapped control registers needed for V-PMP to interact with the bus. As the input interface, it exposes 8 x 32 bits ports for the V-PMP to store 256 bits from the register to the output interface per clock cycle. So far, we didn't discuss the problem of memory alignment. In fact, V-PMP can only address 32 bits word, while most of the packets are aligned on a byte basis. To overcome this issue, the output interface addresses realignment, allowing V-PMP to address and write bytes instead of words. This will save many clock cycles that would have been spent on realigning packets. The interface consists of 256 bit-vector, called *interlock*, which can be addressed by 8 pointers coming from the memory units of the V-PMP. Every pointer addresses a byte on the interlock and a maximum of 8 bytes for every single lane can be written in a clock cycle. More than one lane can be writing on a bit, possibly causing data inconsistency on the interlock. This has been solved by means of summing the contribution of all the lanes that hit a particular bit. A more complex approach would have been to put locks on the bits that are currently written.

- **Direct Memory Access**
  In many applications we need to operate only on portions of the packet (i.e. the packet headers), while the rest of the packet must be leaved as-is. This is the case, for example, of the NAPT, where only the IP source field of the packet is changed (and the checksum), while the Ethernet portion and the payload remains unchanged. We developed a *Direct Memory Access* (DMA) block that moves the data from the input FIFO and the metadata RAM to the output interface without the intervention of the V-PMP.

  The DMA has a dedicated input port on the input FIFO and another dedicated input port on the metadata RAM. The V-PMP controls the DMA by means of ad-hoc registers that specify the number of bytes to move and the start address. This block has only one output port which is connected to a dedicated port on the output interface and is able to transfer 256 bits per clock cycle.

2.4.5.1 Synthesis result

The maximum frequency at which the design can be synthesised is 220 MHz, well above the 156 MHz needed for the 10 GbE interfaces to work properly. Porting this design to silicon, with the latest technological node pointed at 22 nm, the system can reach easily the 1GHz frequency. The synthesis results are reported in Table 2-5: As expected the footprint of the VLIW is quite small, allowing the deployment of several instances of the VLIW. For example, for the NetFPGA prototype it is possible to allocate a VLIW instance for each output post, thus sustaining the line rate execution of complex encapsulation actions such as IpinIP encapsulation (see Section 2.4.5.2).

Table 2-5: Synthesis results.

Resource Type	Used/Available	% Used
Look-up tables	9885/433200	2.28%
Block-RAM	4/1470	0.27%

2.4.5.2 Performance Evaluation

To evaluate the performance achievable with the V-PMP we tested three use cases that require the modification of the packet headers. The three use cases taken into account are: (i) the ARP reply, which generate a packet when an ARP request arrives to the V-PMP; (ii) Network Address/port translation (NATP), which modify the IP/TCP headers and (iii) an IPinIP encapsulation, which require field insertion and checksum recalculation. We compare the results obtained with V-PMP with respect to the previous MIPS implementation.

- ARP Reply

Architecture	Clock Cycles	ASIC Throughput [Gb/s]	FPGA Throughput [Mpps]
PMP MIPS	18	28.4	N/A
PMP VLIW	3	170.4	73.3

- NAPT

Architecture	Clock Cycles	ASIC Throughput [Gb/s]	FPGA Throughput [Mpps]
PMP MIPS	44	11.6	N/A
PMP VLIW	10	51.2	22.0

- IP-in-IP

Architecture	Clock Cycles	ASIC Throughput [Gb/s]	FPGA Throughput [Mpps]
PMP MIPS	42	12.2	N/A
PMP VLIW	13	39.4	16.9

The results show that the V-PMP provides a speed-up in the range 3x-6x for the three use cases taken into account.

2.5 FlexRAN+ platform

Software Defined Networking (SDN) [1] is marked through a separation of control and data plane in order to centralise and facilitate network control mechanism. The application of this approach to the Radio Access Network (RAN), termed SD-RAN, brings these benefits to the RAN part of the mobile network infrastructure being arguably the most complex piece in the network. Through SD-RAN, greater flexibility in the control of the RAN can be achieved by means of data plane programmability, as well as enabling coordination between different RAN network nodes more naturally.

FlexRAN [2] is an SD-RAN platform, extending the OpenAirInterface (OAI) platform [3] to support this paradigm. FlexRAN introduces a new custom south-bound Application Programming Interface (API) to separate control from data plane. The data plane can henceforth be controlled through a central controller to which every base station (BS) connects through its agent. Thus, FlexRAN provides the SD-RAN capability to OAI platform in order to support RAN control applications development, enable coordination between various RAN infrastructure entities and provide control of the RAN over time.

To go one step further, we extend our aforementioned FlexRAN implementation to support the heterogeneous and disaggregated RAN deployments as well as the RAN slicing, which are both the major concerns of the 5G-PICTURE project, as the FlexRAN+ platform.

2.5.1 FlexRAN review

As explained in deliverable D3.1, FlexRAN consists of the FlexRAN service and control plane as well as the FlexRAN application plane. The control plane offers a hierarchical design of a real-time controller and a number of RAN runtimes (as the agents), one for each monolithic BS. The runtime provides the execution environment for the RAN instances and also the separation of control plane (at the northbound, towards the controller and control applications) and data plane (as the southbound, toward RAN modules). The application plane allows for development of RAN control applications to support monitoring, control and coordination of the underlying RAN runtime. Figure 2-21 shows the FlexRAN architecture.

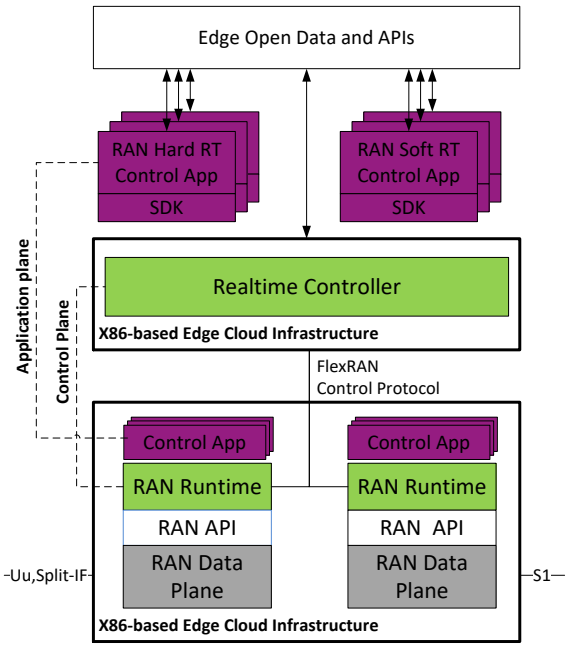


Figure 2-21: OpenAirInterface and FlexRAN platforms.

2.5.2 FlexRAN+ enhancement

So far, in FlexRAN, the SD-RAN principle have been solely applied to the case of the monolithic BSs encompassing the full RAN layer stack from the antenna over the physical layer (PHY) up to the radio resource control (RRC) that are deployed as “black boxes”. Whereas the original C-RAN approach aims to centralise almost all functionality of a BS in a central place, other functional splits like the newly standardized 3GPP F1 split [4] can split the RAN functions between the radio link control (RLC) and packet data convergence protocol (PDCP) sub-layers. This results in multiple RAN entities that work together to constitute a full functionality of BS. Therefore, the traditional approach of handling one full BS per controller connection might not be valid anymore. To this end, it is needed to provide a unified programmable platform that can support a number of deployment flavors (e.g., monolithic/disaggregated) and with the customizable capability for different use cases (e.g., RAN slicing) of the RAN. Also, this unified control platform can provide a common API to simplify the application development on top for the data plane processing over different RAN deployments [5]. We therefore preserve data plane programmability in a disaggregated RAN.

Therefore, we formulate several goals for the FlexRAN+ platform:

1. Abstract a heterogeneous network into *logical* base station (as in the monolithic BS case)
2. Provide unified programmability for different BSs, irrespective of the underlying type of deployment (i.e., monolithic or split-based RAN deployment) through a high-level API.
3. Increase coordination between heterogeneous BSs (possibly with same or different RATs) via connecting them to the same controller such that a flexible coordination of BSs with different levels of centralization becomes feasible.
4. Handle/manage multiple RATs as foreseen in 5G, e.g., to flexibly and dynamically relay user traffic among different RATs.

Moreover, the abstraction of the physical RAN entities into the logical BS representation can be achieved through different types of RAN runtimes that are responsible for different deployment types (according to the applied functional splits between the RAN entities):

1. Monolithic BS (e.g., LTE evolved node B [eNB] or 5G next generation node B [gNB]) runtime with local radio.
2. Runtime for CU and runtime for DU according to the F1 split defined in [4].
3. Runtime for C-RAN: It has special functionality for the PHY split, e.g., 3GPP options 7-1 and 8 defined in [6] for controlling the applied functional split, reconfiguration, etc.
4. Runtime for nFAPI: It has the particular functions for programming this split according to the standard, e.g., instantiating the corresponding PNFs, etc.

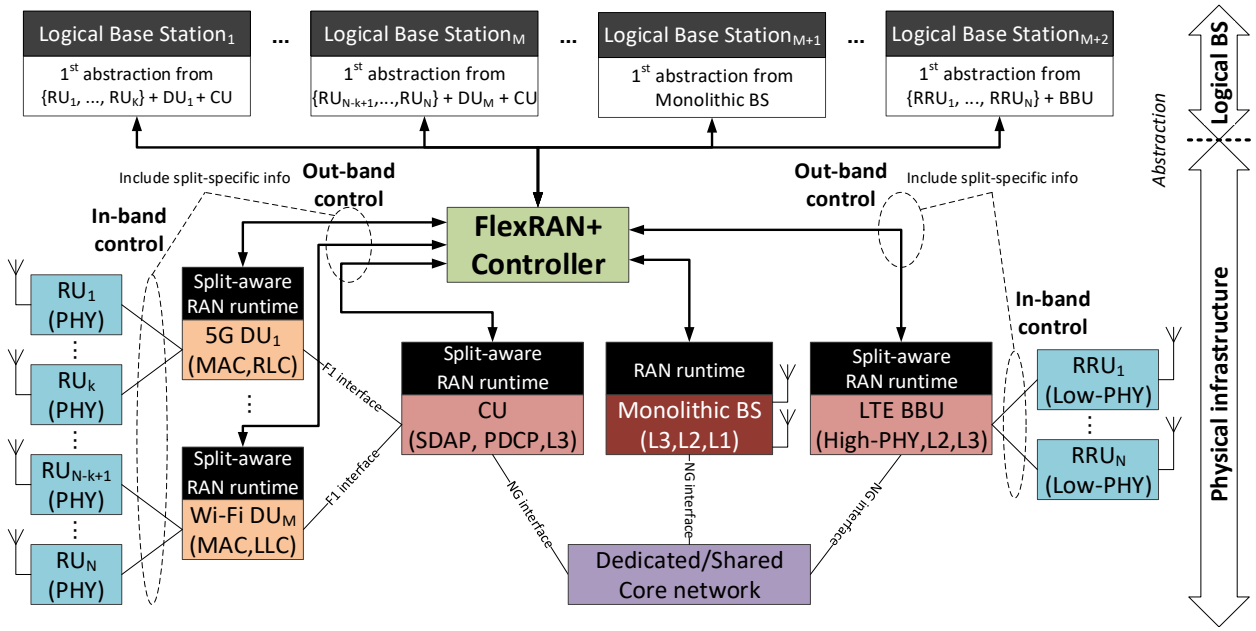


Figure 2-22: Architecture of FlexRAN+ on top of a disaggregated RAN, exposing logical BS at the northbound Interface.

The design of corresponding runtime may rely on introducing additional control modules and omitting some RAN functions in order to achieve the desired functionality, e.g., the PDCP module is skipped at the DU when applying the PDCP-RLC split between CU and DU. Also, in a three-tier RAN deployment, two functional splits can be applied consecutively (e.g., DU with F1 split toward CU and PHY split toward RU), and thus a mix of control modules for different splits are needed.

Moreover, the original FlexRAN protocol between the RAN runtime and the controller is extended to convey the capabilities of each runtime (e.g., PDCP, RRC) as well as the BS identification, as the FlexRAN+ protocol. When the runtimes are connected to the FlexRAN+ controller, the controller can match all runtimes based on the BS identification and the capabilities of all involved runtimes. If the union of the capabilities can be formed completely, a logical BS can be exposed at the northbound interface. This is shown in Figure 2-22, in which multiple deployments of RAN runtimes (three- and two-tier disaggregated and monolithic RAN entities) connect to the central FlexRAN+ controller, which maps to different logical BSs, effectively doing an abstraction from the physical deployment to a legacy view of a monolithic RAN, consisting of a complete BS.

We refer to the term logical BS in order clarify that it is a BS composed of multiple RAN modules and their respective associated runtime, which is similar to a monolithic BS that is composed of control- and data- plane processing among necessary layers as a single logical entity. An example is shown in Figure 2-22: RU_1, providing the PHY layer, is connected to DU_1 with MAC and RLC layers. Through the aforementioned F1 split, it is again connected to CU_1 providing RRC, PDCP and service data adaptation protocol (SDAP) layers. The runtimes of DU_1 and CU_1 are connected to the FlexRAN$^+$ controller. Based on the capabilities of those runtimes, the controller is able to infer that a complete set of capabilities for a BS is available. Since RU_1 is completely controlled by DU_1, no connection between RU_1 and the controller is needed.

Finally, only a complete logical BS, described through complete runtime capabilities, defines a "workable" BS that will be exposed at the north bound for the multi-slice RAN purpose. If only partial capabilities are available, the incomplete logical BS is not exposed. Conversely, if a logical BS has been present but one network function went offline in its life-cycle, the corresponding logical BS disappears as well. Hence, the controller's task with respect to the disaggregated RAN is thus to collect information of different RAN runtimes into the data structure for one logical BS. Based on the format of the simple monitoring currently employed by the FlexRAN+, the controller informs about the participating runtimes as follows:

```
{
  "eNB_config": [         // list of base station configs
    {                     // first base station, assume CU/DU split with local DU functionality
      "eNBId": 234881024, // base station ID
      "agents": [
        {
          "agent_id": 0,
          "capabilities": ["PDCP","RRC","SDAP"]
        },
        {
          "agent_id": 1,
          "capabilities": ["RLC","HIMAC","LOMAC","HIPHY","LOPHY"]
        }
      ],
      "eNB": {
        "cellConfig": {
          // cell-specific configuration parameters, stemming from PHY and RRC
        }
        // ...
      }
    }
  ],
  "mac_stats": [
    "eNBId": 234881024,
    // all stats here, irrespective from which runtime it comes
  ]
}
```

2.5.3 FlexRAN+ evaluation

Based on the provided FlexRAN+ solution, we compare the evolved FlexRAN⁺ and legacy FlexRAN as shown in Table 2-6. We highlight the increased number of supported deployment types while maintaining an identical interface. Where necessary, information about the underlying entities such as DU, CU can be provided.

Table 2-6: Comparison of FlexRAN and FlexRAN+ characteristics.

Property	FlexRAN	FlexRAN⁺
Supported Deployments	Monolithic BS	Monolithic BS, F1 split
North-bound representation	One logical BS	Logical BS, information about RAN runtimes
Programmable through Control Applications	Yes	Yes (identical interface)
Real-time Support	Yes	Yes
FlexRAN protocol	Baseline	Extended to convey capabilities information, Runtimes answer with appropriate information; base stations are identified by BS ID and not a runtime ID.

On a quantitative level, we compare the CPU and memory usage to assess the feasibility of the FlexRAN+ control framework. We distinguish the cases between monolithic BS with either FlexRAN or FlexRAN+ and disaggregated RAN entities with FlexRAN+. Also, two different user association cases are evaluated: (a) idle mode without any associated user, and (b) with one associated user. All deployments were tested on a system based on an Intel i7-8550U with 32GB RAM. The connected commercial-of-the-shelf (COTS) phone targeted a peak data rate of 35 Mb/s in downlink direction. In all cases, every runtime will send the BS statistics in an interval of 10 ms.

In Figure 2-23, we observe that regardless of whether a UE is connected or not, the runtime has a near constant CPU usage overhead of approximately 10%. First, the improved FlexRAN⁺ incurs no additional overhead as compared to the original FlexRAN. However, in the case of a disaggregated deployment, consisting of a DU and CU with F1 functional split, each of the attached runtimes leads to an increase of around 10% CPU usage, such that the total CPU usage is higher as in the monolithic case. An additional UE has no effect on the CPU usage of the FlexRAN+. Note also, that the employed F1 split itself leads to an increased CPU usage.

However, the incurred CPU processing load seems justified considering that the runtimes both send frequent statistics updates which can be lowered (depending on the use case) or be even switched off except occasional one-shot updates. Thus, we conclude that the employed FlexRAN+ controller incurs moderate CPU usage increase which is in line with the previous solution.

Figure 2-24 shows the small overall memory usage when using FlexRAN or FlexRAN+ platforms. This is due to the extra memory overhead of the RAN runtimes (around 3-5%) but can be practically neglected in comparison of the memory usage of the underlying OAI RAN. Also, the number of UE has no impact on the RAN runtimes and only shows minimal effect on OAI. To this end, we can conclude that FlexRAN+ has little impact on the platform performance, but it provides substantial enhancement over the legacy FlexRAN platform.

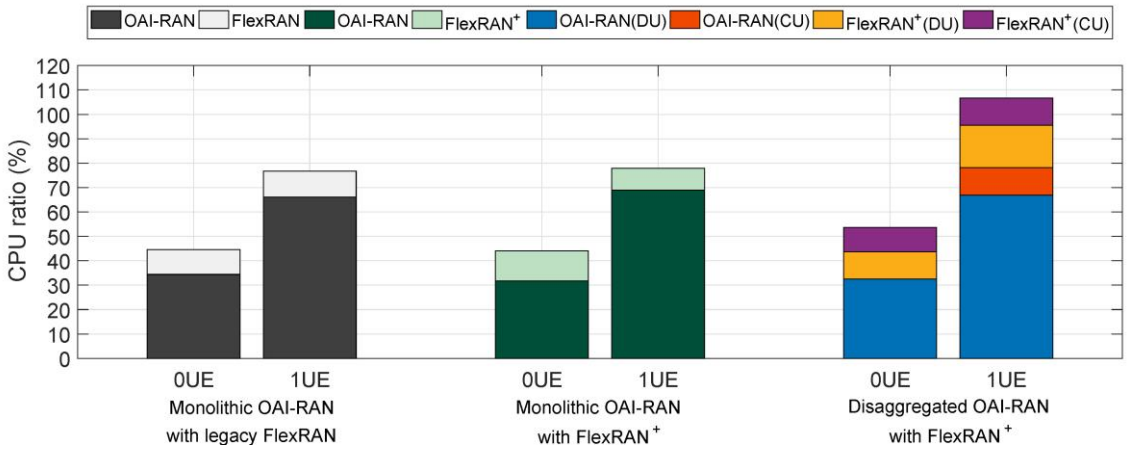


Figure 2-23: Comparison of CPU usage per processing core for original FlexRAN and FlexRAN⁺ for different deployment types.

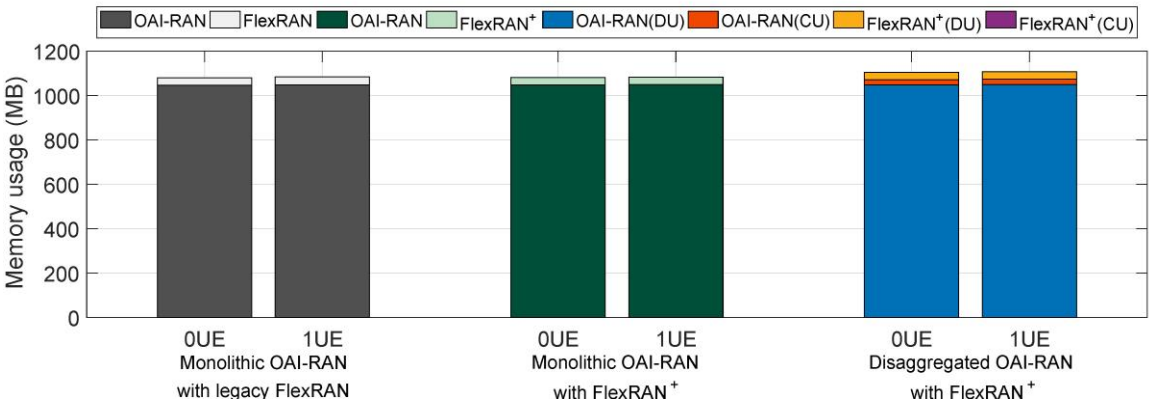


Figure 2-24: Comparison of of memory usage for original FlexRAN and FlexRAN+ for different deployment types.

2.6 PTMP MAC processor

This section describes the Point-to-Multi-Point (PTMP) Medium Access Control (MAC) processor for 60 GHz multi-gigabit single-hop wireless communication. A general medium access scheme and the link establishment using beam steering/beamforming antennas with very narrow (pencil) beam characteristic will be introduced. The single-hop PTMP communication system setup consists of one master device/node and one or more slave device(s)/node(s) as shown in Figure 2-25. In such a scenario beam switching of slave device(s) is not necessary after link setup. Each device is identified by an 8-bit address and can be configured as master or slave. Both master and slave devices use the same hardware and software design which is implemented on IHP's digiBackBoard platform. Figure 2-26 shows the digiBackBoard FPGA platform. This platform is introduced thoroughly in deliverable D3.1. It consists of 2.16 GSPS DAC and ADC and a programmable hardware (i.e. FPGA). Four Gigabit-Ethernet transceivers, USB and SMA connectors provide connectivity to the platform.

The implementation of the baseband and MAC processor design was done completely using a hardware-description language (VHDL). Some design parameters depend on the configuration based on requirements and available resources during design time and not changeable after synthesis. The current design supports one master and up to two slaves. Table 2-7 shows the configurable parameters of current PTMP MAC and BB processor implementations on IHP's digiBackBoard platform. Parameters marked with "r/w" are changeable during runtime. The MAC processor supports mechanisms like aggregation of Ethernet packets and selective repeat request (SRR) in order to reduce PHY and MAC overheads for the highest data throughput. These mechanisms are introduced in the following two paragraphs.

- Aggregation of Ethernet Packets: Incoming Ethernet packets will be extended by a header which stores the packet length. Switch functionality learns the relations of Ethernet source MAC addresses and destination addresses of the MAC device. The following transmit buffer aggregates these packets independently for each destination device (slave) into a bigger one according to the system configuration and requirements.
- Selective Repeat Request. The aggregated packets are marked with a sequence number in order to support selective repeat request. This allows sending frames using delayed acknowledgement to reduce the PHY and MAC overhead. For instance, using immediate acknowledgement limits the data throughput to 270 Mbps at 1.3 Gb/s PHY data rate (without FEC). In opposite delayed acknowledgements increase the data throughput to 1 Gb/s. The acknowledgements are part of the MAC header structure and control sliding windows for TX and RX buffer.

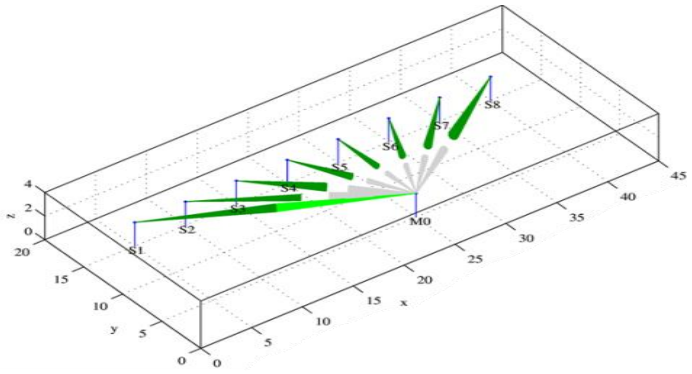


Figure 2-25: Point-to-Multi-Point single-hop scenario.

Figure 2-26: digiBackBoard with the cooling and SMA adapters, Bottom and Top view.

Table 2-7: Configuration of PTMP-MAC.

Type	Value	r/w	Description/remarks
Device ID	0x00 0x01…0xFE 0xFF (reserved)	r/w	Master Slave TX broadcast
Role	0 1	r/w	Master Slave
Superframe duration	0…65535 µs	r/w	Including beacon and beam search slot
Data slot mode	0 1	r/w	Dedicated data slots Unified data slot
Duration beacon slot	0…65536 µs	r/w	Min = 55 µs
Duration beam search slot	0…65536 µs	r/w	Min = 55 µs
Duration per data slot	0…65536 µs	r/w	(Superframe – beacon slot – beam search)
Retransmissions	0…15	r/w	Retransmissions per frame
Aggregation max bytes	1..buffer_max_bytes	r/w	Maximum Bytes per frame
Aggregation timeout	1..2047	r/w	Aggregation timeout per frame
Buffer max bytes	16384	r	Max bytes per frame buffer
Buffer count	8	r	Frame buffers per slave
Slave count	2	r	Maximum slaves
Slot count	8	r	Configurable slots (incl. beacon and beam search)
Modulation scheme	0 1	r/w	BPSK (1.3 Gb/s) QPSK (2.6 Gb/s) 16-QAM (5.2 Gb/s) (depending on available resources)
FEC scheme	0 1	r/w	Viterbi (R=1/2) Viterbi (R=1/2) + RS (R=239/255)

The following subsections will give specific details of the implemented Point-to-Multi-Point MAC processor.

2.6.1 Point-to-Multi-Point TDMA Medium Access Scheme

Synchronisation is needed for contention-free medium access between the devices. Therefore, the medium access is organised in a superframe which is divided into beacon-, beam search-, and data slot. Additionally, antenna beams of master and one or more slave devices have to be aligned before communication can be started. This is done at the beginning of each time slot of the superframe. Finding such a configuration (alignment) of suitable beams between two devices is processed in a separate link establishment procedure using the beam search slot. This procedure is described in subsection 2.6.2.

The beacon slot is used for synchronisation, configuration and link monitoring. The beam search slot is used for finding, refinement, and tracking of the beam alignment between master and slave device(s). The duration of the beacon and beam search slots is 55 µs each and it is based on the smallest MAC-protocol-based round-trip-time (RTT) plus guard/overhead time. The data slot can be divided into several slots depending on the design configuration and requirements. The next paragraphs give more details to the slot types.

- Beacon Slot

  A beacon packet is sent to all available slave devices in round robin fashion for synchronisation and providing system parameters. Each slave device answers with the copy of the parameters sent by master. A successful answer is used for link monitoring between master and slave.

  The beacon is sent periodically starting a superframe containing beacon-, beam search-, and data slot(s). The period/duration of the superframe can be selected depending on the requirements of the system, e.g. latency and amount of slave devices. The MAC postpones a data transmission into the next superframe, when the current superframe will end during transmission.

- Beam Search Slot

This slot is optional; however, it is needed for the link establishment of devices that have to be integrated in the system in order to examine the best beam configuration for communication. It is also used for software-based beam refinement and continuous beam tracking.

- Data Slot(s)

  A communication is started by sending a packet from the master to the dedicated slave device. The MAC header defines the type of answer and the granted frame duration based on payload length for data transmission. The slave answers with acknowledgements of received packets and data payload. The MAC postpones a data transmission to the next slot or superframe, if the current slot will expire during the transmission.

  In single-hop communication scenario, a unified (single) data slot is used. All slave devices point to the master. The master can select the slave devices in a random access scheme. The duration of the unified data slot is the remaining duration of the superframe. Figure 2-27 shows such a superframe structure.

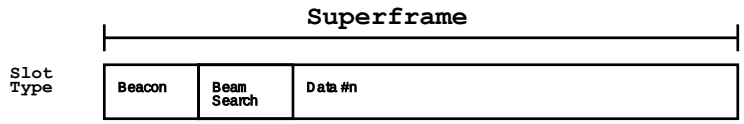


Figure 2-27: Superframe structure with unified data slot for all devices.

In multi-hop scenarios dedicated data slots are necessary. Each slot points to a different device with different beam configurations. The durations of the dedicated data slots for each master/slave device are configurable based on the remaining time of the superframe. Figure 2-28 shows such a superframe structure.

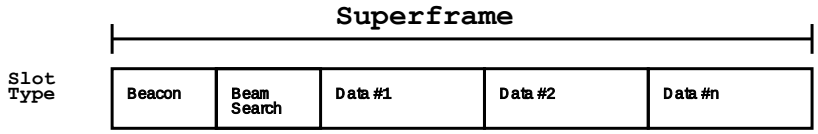


Figure 2-28: Superframe structure with dedicated data slots per device.

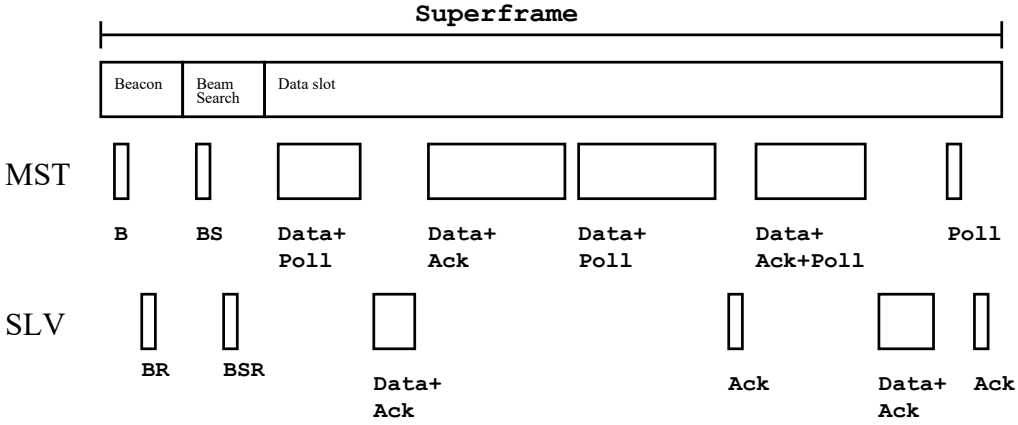


Figure 2-29: Example of TDMA medium access during superframe and one unified data slot.

Communication always starts by the master device. The slave device gives an answer packet with or without data payload respectively, i.e. an acknowledgement packet when the destination device address is matched and an answer is requested. The beacon and the beam search packet always expect an acknowledgement. A data packet (with/without payload) can request an immediate or no acknowledgement. The delayed acknowledgement is requested with the last packet of a packet block. Figure 2-29 shows an example for the medium access using a unified data slot. In the beacon slot the master (MST) sends a beacon packet (B) to the slave (SLV). The addressed slave answers with an acknowledgement packet (BR). In the beacon search slot, a tracking packet (BS) is sent from the master to the slave and answered by the slave with an acknowledgment packet (BSR). Afterwards, the data slot starts with a single data packet from the master to the slave with a poll

request. The slave answers with a data packet and an acknowledgement included. After receiving the answer from the slave, the master starts a data packet block. The first data packet includes an acknowledgement but no poll request. The second packet of the block includes also the poll request. Then the slave answers only with an acknowledgement due to an empty buffer. At the end of the superframe/slot only a poll request and an acknowledgement is possible but it is used for update of the buffer states. After this, the next superframe starts with the beacon slot first.

2.6.2 Link establishment

This section describes the link establishment procedure between two devices using an analogue frontend (AFE) with N (overlapped) beam positions. This procedure has to be done after the system starts each time a device is added to the system or moved out. The superframe timing/structure of the MAC protocol controls the access of the new devices. For initial timing, slaves assume a default superframe duration when communication is not possible and synchronisation is not finished yet.

The procedure runs an exhaustive search of all combinations of beam positions between master and slave. The slave changes its beam position after the multiple of default superframe duration and beam positions. The master changes the beam positions after default superframe duration. The control is done from the firmware side. After the first working beam configuration pair is found, synchronisation via beacon packet is possible, and then a further software-based tracking procedure can maximise the quality of the link.

The following explanations give an overview of the link establishment for current system configuration. Assuming a superframe duration of 4 ms (SF_{dur}=4 ms) and N = 64 beam positions (BP_{cnt}=64), maximum duration of the link establishment procedure equals $BP_{cnt} \times BP_{cnt} \times SF_{dur}$. Table 2-8 shows an example flow of the procedure.

Table 2-8: Example of beam search algorithm for the link establishment procedure

Step	Master	Slave
1	Start superframe	Start virtual unsynchronised superframe timing via firmware
2	Setup beam search: set beam position n=0	Setup beam search: beam position n=0
3	Configure beam position n	Configure beam position n
4	Send beam search packet in beam search slot Wait for beam search answer Goto step 6 when answer successful received Increase beam position n=n+1 Wait for end of superframe Goto step 2 when (n==BP_{cnt}) Goto step 3	Goto step 6 when received beam search packet or beacon packet from master Goto step 5 when superframe duration (BP_{cnt} * SF_{dur}) expired Increase beam position n=n+1 Goto step 2 when (n==BP_{cnt}) Goto step 3
5		Increase beam position n=n+1 Goto step 2 when (n==BP_{cnt}) Goto step 3
6	Wait for end of superframe	Waiting for beacon packet or timeout (4 * SF_{dur}) Goto step 7 when beacon packet successful received Goto step 2 when timeout has occurred
7	Do tracking of best beam position	Waiting for action from master

In deliverable D3.3 we plan to perform experimental evaluation of the P2MP MAC processor using multiple slave devices. In addition, evaluation of beam search and beam tracking algorithms in different scenarios will be carried out.

2.7 NETCONF server and Yang models for Time Sensitive Networks (TSN)

2.7.1 YANG model

During the process of development of TSN products different requirements may often contradict each other. Hence, a general model is needed that can be augmented with extra features on demand. YANG presents this flexibility. However, it is important that the design of the model is kept as close as possible to a standard widely

adopted hardware that can be used as reference base. Another requirement is that the model can be easily integrated in a Network Manager, e.g. OpenDaylight (ODL). A common approach for designers of prototype networking hardware in recent years is to create extensions to the OpenFlow protocol that enable the new feature[5]. However, there is no YANG/NETCONF involved in this approach and one has to document the extension and implement it without the advantages provided by the machine readable YANG modules and the flexibility of the NETCONF protocol. The idea of using YANG but retaining the advantages of OpenFlow for models of TSN devices motivates for a YANG model based on the OpenFlow flow forwarding configuration and monitoring semantics paired with a flexible scheduler management module. Thus, creating an implementation that works for OpenFlow devices with TSN extension easing integration with ODL. Our proposal for such modules are published as IETF drafts ietf-network-bridge.yang ietf-network-bridge-flows.yang ietf-network-bridge-scheduler.yang[6]. We have demonstrated partial implementation for our existing devices during the ECOC 2018 demo session and we have presented the model as part of the OMNET++ 2018 summit where automated simulation of instance data representing network topologies based on this model was one of the projects of the hackathon event. The conclusion so far has been that the flexibility of the scheduler part of the model allows support for all standard TSN solutions while the wide adoption of OpenFlow provides enables implementation of the model for many preexisting devices and allows fast development, testing and deployment times.

2.7.2 NETCONF server

The netconfd server part of the yuma123 (maintained by TransPacket) toolchain is one of the few alternatives for implementation of device side NETCONF server that has YANG based automation features. Among its properties is the use of C language, open source BSD license, and detailed documentation[7]. Furthermore, it is a part of a mainstream operating system distribution Debian/Ubuntu[8]. The recently released 2.11 version includes and supports the ietf-network-bridge.yang ietf-network-bridge-flows.yang ietf-network-bridge-scheduler.yang draft modules, and the server has been tested for interoperability with ODL. The project maintains repository with server and client side tools for development. TransPacket products are based on the solution.

2.7.3 Joint NETCONF/YANG service for FUSION and passive WDM

For 5G-PICTURE, the combination of TransPacket TSN and ADVA passive WDM technologies is considered by those two partners as a stable reusable option for creating the optical transport infrastructures for demonstrations in WP6, review meetings, and external conferences. To simplify the SDN orchestration for this platform combination, TransPacket and ADVA are evaluating solutions for a common NETCONF/YANG-based control plane.

The joint FUSION/pWDM management service would be ideally provided by the passive WDM tail-end device, since it is realized as a pluggable module of ADVA's FSP 3000 AgileConnect[9] platform, according to the proposal in deliverable D3.1 [1] section 5.1, and is therefore already integrated with that platform's management module, which is running a full GNU/Linux operating system. This would also reduce the need for external controller hardware for the purely FPGA-based TransPacket FUSION nodes.

2.8 NETCONF/YANG service development framework for ADVA's hardware platforms

In deliverable D3.1 [1] sections 2.10 and 5.1, ADVA proposed SDN-integration of all their 5G-PICTURE hardware technologies. This is achieved by implementing NETCONF-enabled control planes on those platforms, based on device-specific YANG models, which expose their controlling and monitoring features.

As further described in section 3.5, a prototypical NETCONF/YANG service implementation for ADVA's passive WDM transport platform already exists. The control plane of ADVA's programmable edge device platform

[5] https://pure.tue.nl/ws/files/52754319/20170125_Miao.pdf

[6] https://datatracker.ietf.org/doc/draft-vassilev-netmod-network-bridge/

[7] http://www.yuma123.org/wiki/index.php/Yuma_Developer_Manual

[8] https://tracker.debian.org/pkg/yuma123

[9] https://www.advaoptical.com/en/products/scalable-optical-transport/fsp-3000-agileconnect

will be developed in accordance to the future evaluation and API definition of the platform-specific P4 programmability, which is introduced in sections 3.4.2 and 4.8.

These NETCONF/YANG services follow a different implementation approach compared to e.g. the FUSION NETCONF services of TransPacket described before in 2.7.2, but aim at the same interoperability with SDN orchestrators, especially ODL and ONOS, and at compatibility with TransPacket's yuma123-based implementations, especially for enabling possible joint FUSION/pWDM SDN service options as mentioned in 2.7.3. For getting a better understanding of the yuma123 approach and the TransPacket FUSION control plane, ADVA joined IETF 101 Hackathon[10] to work together with TransPacket on their official hackathon project[11] for improving their framework.

The basic SW technologies for ADVA's NETCONF/YANG services are free open source projects developed and maintained by CESNET[12] (Czech Educational and Research Network), available from their GitHub account[13], and also targeted at GNU/Linux operating systems. The decision for preferring those technologies was made in accordance with ADVA product developers, based on their requirements, to have a common SDN agent software platform for future ADVA product development and research activities.

For simplifying and accelerating the development of SDN interoperability through NETCONF and other protocols like RESTCONF of the actual device management applications, and for enabling a common programmatic integration style in different software development environments, ADVA has developed additional higher level APIs (Application Programming Interfaces) for C++ and Python on top of the basic CESNET C language APIs. Further APIs for Java and Go[14] are also under development. All those APIs are named Yangaroo and will also be made freely available by ADVA under the free open source Apache License 2.0[15]. Further, a YANG-based code generator tool named Alpakka, which can be used in conjunction with Yangaroo, was developed and already publicly released[16] on the official ADVA GitHub account. Code samples from ADVA's passive WDM platform can be found in 5G-XHaul deliverable D3.3 [33] section 2.5.

Such an ADVA NETCONF/YANG service implementation consists of three components as shown in Figure 2-30.They are running as separate software processes, interacting via IPC (Inter Process Communication):

(1) The YANG module repository and configuration data store, which is managed by CESNET's sysrepo[17] server component.

(2) The actual device management application, which synchronises the hardware state with the YANG configuration state, and translates any incoming RPC method calls from external NETCONF clients to actual hardware commands. This component exclusively uses the ADVA Yangaroo APIs, which internally communicate with the YANG configuration data store through CESNET's sysrepo client library, and additionally provide any YANG module information from the local repository through CESNET's libyang[18] parser component.

(3) CESNET's Netopeer2[19] NETCONF protocol handler, which internally communicates with the sysrepo server component independently from the actual device management application, and exposes the actual NETCONF over SSH 2.0 management service to external NETCONF clients, taking over authentication credentials from local GNU/Linux system users of the managed device.

[10] https://trac.ietf.org/trac/ietf/meeting/wiki/101hackathon

[11] http://www.yuma123.org/wiki/index.php/IETF_101_Hackathon

[12] http://www.cesnet.cz

[13] https://github.com/CESNET

[14] https://golang.org

[15] https://www.apache.org/licenses/LICENSE-2.0

[16] https://github.com/advaoptical/alpakka

[17] http://www.sysrepo.org

[18] https://github.com/CESNET/libyang

[19] https://github.com/CESNET/Netopeer2

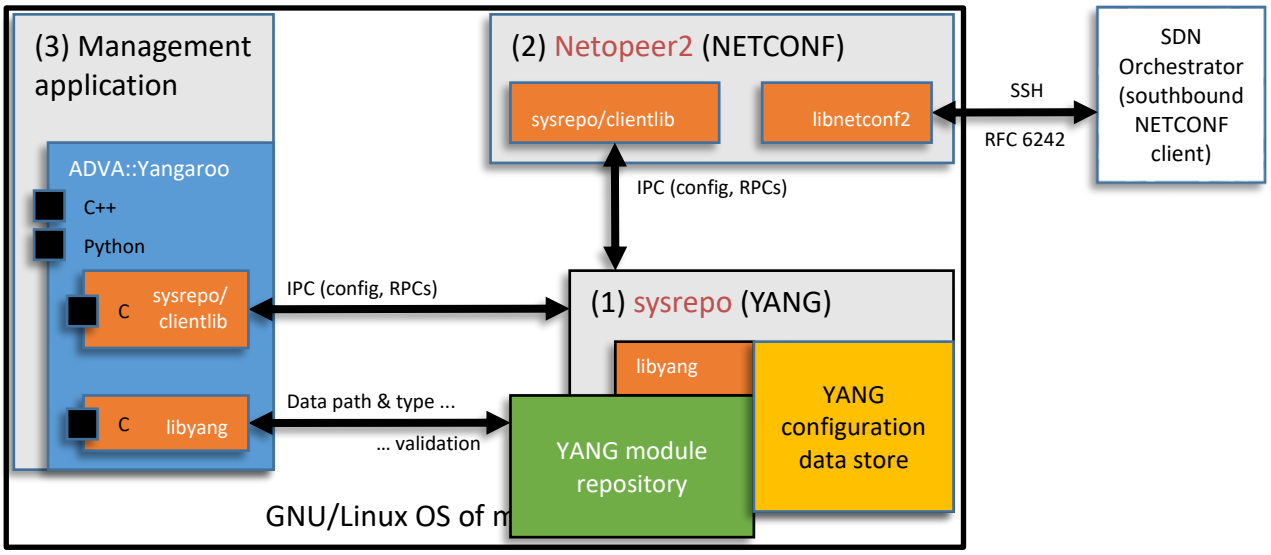


Figure 2-30: ADVA NETCONF/YANG service implementation components.

3 Hardware abstractions: performance assessment and functional evaluation

In deliverable D3.1 [1] several HW abstraction methodologies to enhance usability of the programmable network platform and to abstract the implementation details of each platform has been defined. In particular, this section will present: i) the implementation of the API/interfaces for the Typhoon, GateWorks Ventana and OpenAirInterface platforms; ii) the programming languages for data plane programmability; iii) the NETCONF/YANG service for the passive WDM platform.

In this deliverable, the detailed implementation of the hardware abstractions is presented, and the preliminary performance assessment and functional evaluation of the developed HW abstractions is provided. A thorough evaluation of the hardware abstractions together with the integration of the different abstractions will be carried out during the rest of the project and will be documented in the deliverable D3.3.

3.1 APIs for the Typhoon platform

3.1.1 Control and Programmability of Synchronisation Functions

In 5G-PICTURE deliverable D4.1 [2], synchronisation functions were presented for the BWT Typhoon IEEE 802.11ad mesh nodes. In summary, the Typhoon device supports transmission and reception of IEEE 1588 Precision Time Protocol (PTP) [12] messages with hardware timestamping of PTP event messages. In the present deliverable, the primary goal is to describe how these functions can be controlled in the network. More specifically, the goal is to cover how the synchronisation function fits within the context of NETCONF and OpenFlow, while at the same time relying on the IEEE 1588 management provisions that are available in most PTP application implementations.

The architecture that is considered in this contribution is depicted below. The clock synchronisation function is performed in user space of a host running a PTP application. The application, in turn, requires specialised driver and network device hardware, such as the one provided by the BWT Typhoon's BH2 device, which was initially presented in 5G-PICTURE deliverable D4.1 [2]. The PTP application is responsible for the communication with the device driver, with whom it agrees to gain control of a hardware clock (time counter) and negotiates the ability to receive departure and arrival timestamps of pre-defined packets (PTP event messages). The driver, in turn, communicates to the network adapter and ensures that clock and timestamp interactions become possible.

While the PTP application alone configures the synchronisation capabilities of the driver and network device hardware, the PTP application itself can be managed by external entities. A convenient option is to implement such management capabilities using PTP management messages (packets) that are defined in the IEEE 1588 standard. These allow "get", "set" and "command" transactions to be sent to the PTP application either by a remote host or locally with packet-based inter-process communication. The goal in this contribution is to highlight the level of programmability that this infrastructure can provide and how it can be leveraged while enabling NETCONF and OpenFlow support. Furthermore, this text highlights the potential benefits of using this architecture, including performance.

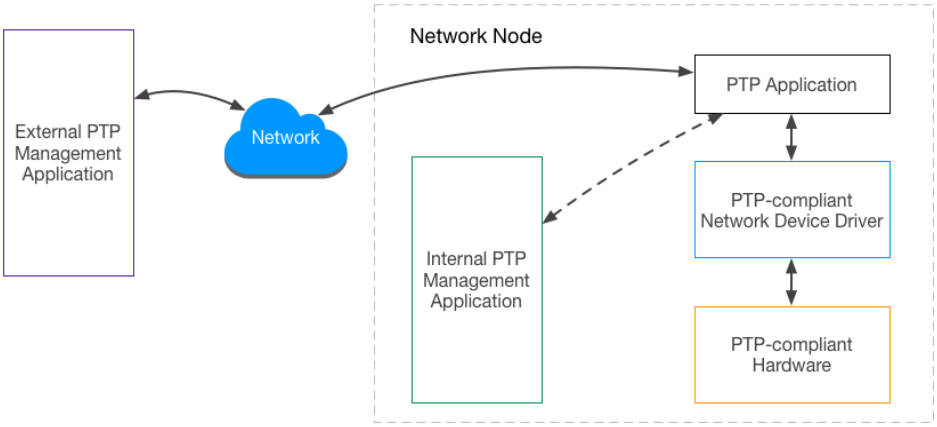


Figure 3-1: PTP-based synchronisation control architecture.

3.1.2 The IEEE 1588 Management Interface

The IEEE 1588 standard defines a management protocol and a PTP device whose sole role in the network is to manage other PTP clocks[20], entitled as management node (or node manager). This node works by exchanging messages (packets) with the PTP nodes to be managed, that is, PTP messages whose "messageType" header indicates management type (see Figure 3-2). These messages allow the node manager to read and write attributes in the target PTP clock (or clocks), as well as the generation of events or notifications. In the sequel, we describe the main aspects of such management transactions and focus on how to leverage them in southbound control interfaces. In particular, we discuss management message addressing, their capabilities and the associated device attributes that can be collected.

First, the IEEE 1588 management messages can be flexibly targeted. They can be sent to all PTP clocks in the network at once, or to a specific PTP clock, or to a specific PTP port[21] in a PTP clock. They can also be addressed to all PTP ports of all PTP clocks in the network. This is enabled by the "targetPortIdentity" field of the management message, which identifies a clock (PTP-capable network node) and the specific port (network interface) in that device. When all bits of this identity field are asserted, it means that the message is supposed to be sent to all clocks and all ports.

The originator of a PTP management message is typically the manager node and the responders are the PTP ordinary, BCs or TCs in the network [12]. A responder sends the management response message uniquely to the originator, particularly by setting the targetPortIdentity field of the response equal to the sourcePortIdentity of the original request.

In addition to the given addressing level provided by the "targetPortIdentity" field, a management message also contains a PTP domain number in the "domainNumber" field of the header. For PTP clocks and ports that participate in multiple PTP domains (e.g. following different grandmasters), this allows targeting the data set that is held in the PTP application for a specific domain. These data sets are discussed momentarily.

In terms of capabilities, management messages can *get* or *set* configurations in a PTP clock, as well as convey *commands* (also thought as events). *Get* or *set* actions are responded by the receiving node with a *response* management message. Meanwhile, a command is responded with an *acknowledge* management message. These messages carry their content within type-length-value (TLV) field (see Figure 3-3), the so-called "*managementTLV*", presented below. The "*managementId*" field of the TLV, in particular, is what defines the specific attribute or command that is being sent or requested to the PTP clock.

Bits								Octets	Offset
7	6	5	4	3	2	1	0		
header								34	0
targetPortIdentity								10	34
startingBoundaryHops								1	44
boundaryHops								1	45
reserved				actionField				1	46
reserved								1	47
managementTLV								M	48

Figure 3-2: PTP management message format.

Bits								Octets	TLV Offset
7	6	5	4	3	2	1	0		
tlvType								2	0
lengthField								2	2
managementId								2	4
dataField								N	6

Figure 3-3: Management TLV.

[20] A PTP clock is defined as a network node participating in PTP.

[21] A PTP port can be interpreted as a (virtual or physical) network interface in a given device that participates in PTP.

A PTP device is characterised by *static*, *dynamic* and *configurable* attributes. Only the *configurable* attributes can be targeted by management *set* messages, while all of them can be read via *get* messages. Furthermore, an entire group of attributes can be read at once, by requesting a specific data set in its entirety. For configuration, however, specific attributes are targeted individually. Table 3-1 identifies the PTP data sets and the operations (managementId's) that read them entirely:

Table 3-1: PTP data sets and read operations.

Data set	ManagementId (Hex)	Actions
defaultDS	0x2000	GET
currentDS	0x2001	GET
parentDS	0x2002	GET
timePropertiesDS	0x2003	GET
portDS	0x2004	GET
transparentClockDefaultDS	0x4000	GET
transparentClockPortDS	0x4001	GET

For the individual data set attributes, there are managementId values that allow their direct configuration. The following identifies some noteworthy commands for the purposes of network programmability of synchronisation functions.

Table 3-2: commands for network programmability and synchronisation.

ManagamentId name	Action	Scope	Description
NULL_MANAGEMENT	GET, SET, COMMAND	port	Null message used for testing management interfaces (such as whether one exists).
CLOCK_DESCRIPTION	GET	port	Reads: 1. Type of PTP clock (ordinary, boundary, transparent or management). 2. Physical layer protocol adopted to transport PTP in the target port. 3. Physical address of the port (e.g. MAC address). 4. Protocol address (e.g. IP address) 5. Manufacturer OUI 6. Product description and revision. 7. PTP profile implemented by the port.
SAVE_IN_NON_VOLATILE_STORAGE/ RESET_NON_VOLATILE_STORAGE	COMMAND	clock	Saves/resets current dynamic and configurable data set members into non-volatile storage.
FAULT_LOG	GET	clock	Returns on or more "fault record", indicating implementation specific fault description and severity code.
PRIORITY1	GET/SET	clock	Influences the BMC algorithm, to control the master-slave hierarchy in the network. From DefaultDS [configurable member]
PRIORITY2	GET/SET	clock	Influences the BMC algorithm, to control the master-slave hierarchy in the network. From DefaultDS [configurable member]
DOMAIN	GET/SET	clock	Used to distinguish synchronisation domains running on the same hardware. From DefaultDS [configurable member]

SLAVE_ONLY	GET/SET	clock	Reads whether the clock can solely behave as slave or set it to behave so. DefaultDS [configurable member]
LOG_ANNOUNCE_INTERVAL	GET/SET	port	Defines the rate of Announce message transmission. From PortDS [configurable member]
ANNOUNCE_RECEIPT_TIMEOUT	GET/SET	port	Defines how long the PTP clock can keep its current grandmaster reference without needing to receive another Announce message. From PortDS [configurable member]
LOG_SYNC_INTERVAL	GET/SET	port	Defines the rate for Sync message transmission in Master mode. From PortDS [configurable member]
DELAY_MECHANISM	GET/SET	port	Defines whether the adopted delay mechanism is end-to-end or peer-to-peer. From PortDS [configurable member] or transparentClockPortDS [configurable member]
LOG_MIN_PDELAY_REQ_INTERVAL	GET/SET	port	Defines the rate of transmission of peer delay request-response messages if using peer-to-peer delay mechanism. From PortDS [configurable member] or transparentClockPortDS [configurable member]
VERSION_NUMBER	GET/SET	port	Defines the implemented PTP version (1 or 2). From PortDS [configurable member]
ENABLE_PORT/DISABLE_PORT	COMMAND	port	Enable/disable a PTP port
CLOCK_ACCURACY	GET/SET	clock	Defines the accuracy of a given clock and, hence, influences whether the given clock gets higher or lower priority in the BMC algorithm. The clock accuracy is the same as the attribute kept within the *default* dataset, particularly within the *clockQuality* field. It allows specification of the range to which the local time is accurate, from within 25 ns up to 10 s. This is useful for PTP clocks that are fed with external time sources, for example a GPS receiver, and whose local time is slave to the external source. When the conditions of the external source change, this attribute should reflect them. From DefaultDS [Dynamic Member]

An important aspect of PTP network design is planning the grandmaster architecture including fall-back plans. This relates to the best master clock (BMC) algorithm to automatically define the best clock to play the role of the grandmaster. Due to the BMC algorithm, the node that plays the role of the grandmaster is automatically defined, specifically by each node discovering its own role (and the grandmaster discovering that it should be it). However, the network manager can exert some control on the results of this automatic definition.

Note that all configurable members of PTP datasets are configurable using management messages. However, not all dynamic members are accessible. Table 3-3 summarises the dynamic attributes for which a dedicated management message is not defined. The entire content of parentDS is read-only, since this information depends on the node's master and grandmaster[22], which gets defined via the BMC. The same rationale applies for the members of the currentDS data set, which are also dependent on the current master that the PTP clock is slave to (in case it is). Meanwhile, the clockClass and offsetScaledLogVariance members of the defaultDS

[22] Grandmaster is the PTP master clock at the top of the hierarchy. All other PTP masters downstream of the grandmaster (I.e. boundary clocks) are solely masters.

are not configurable because these relate to the clock hardware and therefore should be adjusted by the clock alone. Note, that in contrast a property such as the timeSource member of the timeProppertiesDS, despite being related to hardware, is configurable via the TIMESCALE_PROPERTIES managementId. However, the latter property is information-only, so its configurability seeks convenience while knowing it won't affect the reliability of the BMCA (this property is not used in the BMCA).

Table 3-3: dynamic attributes for which a dedicated management message is not defined.

Data set	Dynamic members with no dedicated management message	Comments
defaultDS	clockQuality.clockClass and clockQuality.offsetScaledLogVariance	From the clockQuality structure, only the clockClass.clockAccuracy value can be configured (via CLOCK_ACCURACY managementId).
currentDS	stepsRemoved, offsetFromMaster, meanPathDelay	
parentDS	parentPortIdentity, parentStats, observedParentOffsetScaledLogVariance, observedParentClockPhaseChangeRate grandmasterIdentity grandmasterClockQuality grandmasterPriority1 grandmasterPriority2	
timePropertiesDS		Set via the UTC_PROPERTIES, TRACEABILITY_PROPERTIES and TIMESCALE_PROPERTIES managementIds
portDS		

3.1.2.1 NETCONF

The synchronisation function provided by the PTP application in a network node can be made configurable by combining NETCONF and IEEE 1588 management. The NETCONF server can rely on the IEEE 1588 YANG model that is currently under discussion through the Internet Draft [13]. When requests involving this model are received, these should trigger actions via the configuration API, which in turn can translate the requests into PTP management messages. This architecture is depicted in Figure 3-4.

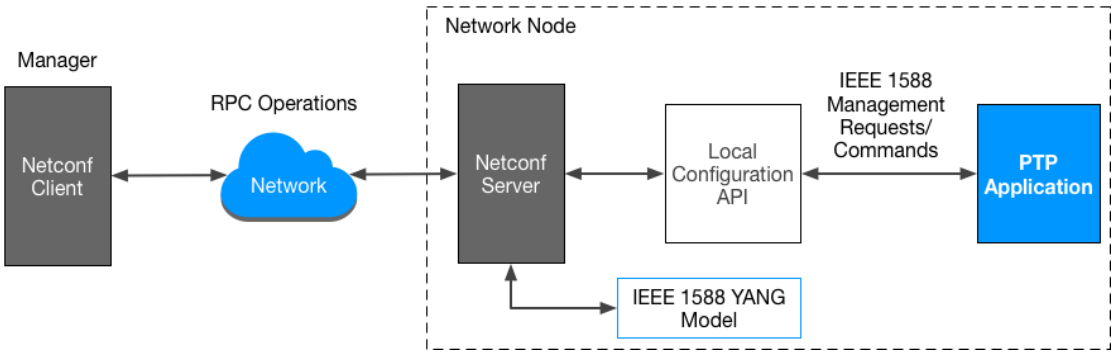


Figure 3-4: NETCONF architecture.

An important point of the YANG model in [13] is that it supports multiple PTP instances, as these are called. This allows multiple PTP applications to be executed in the same host, which allows synchronisation with respect to distinct PTP domains. The coordination between these domains and the shared hardware is out of scope, but the point here is that the layer that is responsible for translation from NETCONF to IEEE 1588 management should be able to distinguish the appropriate PTP application to speak to, namely be able to

recognise PTP instances. One way to achieve this is by having the translator (local configuration API) broadcast messages requesting either the ”domainNumber” member of the defaultDS or the ”primaryDomain” member of the transparentClockDefaultDS.

To prevent indefinite life of management messages in the network, these are also limited in terms of hops they undergo. An ordinary clock (a PTP master or slave) necessarily consumes a management message in the sense that it either responds to the sender or does nothing. Meanwhile, BCs and TCs let the management continue along the network by forwarding. So, to prevent loops, every time a boundary clock (BC) forwards a management message, it decrements the boundaryHops field of the message and, once this field becomes zero, the management message can no longer be forwarded. Besides, the response of a management message inherits the boundary hops count so as to prevent loops due to responses.

3.1.2.2 OpenFlow and Open vSwitch

The next topic is restricted to the use case of a PTP TC) [12] application running on the host. The goal of such an application is to forward PTP messages while tracking their residence time at the node such that the destination of the message can know an accurate of approximation of the message’s one-way delay. This implies that the PTP application involves forwarding functionality, which may conflict with the forwarding function that is provided by another application, such as Open vSwitch (OVS). However, the driver for the modem is able to intercept PTP packets and pass these to the PTP application. This application can then generate a new PTP message with an updated residence time field.

In deliverable D3.3, BWT will report on work to allow the Typhoon module to be controlled by the 5G OS using Netconf/YANG. The focus will be on implementing parts of the IEEE 1588 YANG model to allow configuration of PTP functionality. Other features may also be exposed. The work will exploit the open source Netopeer software and will add a TransAPI function (i.e. how NETCONF interfaces with the underlying system).

3.2 APIs for the GateWorks Ventana platform

5G-PICTURE’s deliverable D3.1 [1] provided both a description of the HW platform used to implement 5G-PICTURE’s Sub-6 GHz access/transport nodes and a high-level view of its configurability, enabled through different APIs. This section provides more details on the hardware abstraction of the 5G-PICTURE’s Sub-6 GHz node defined by means of YANG modelling.

The objective is to enable the different Sub-6 GHz network slicing functions envisioned in 5G-PICTURE’s WP4 [2] and, at the same time, aiming to remain flexible and easily extendable.

The architecture shown in Figure 3-5 allows transport slicing in the Sub-6 GHz access/transport portion of the network, where multiple tenants deploy Small Cells with support for wireless backhauling from the RAN to the tenant’s servers. Recall from deliverable D3.1 [1] that the hardware devices supporting Sub-6 GHz nodes, i.e. Gateworks Ventana (GWV) are equipped with multiple access/backhaul NICs (IEEE 802.11n/ac interfaces). Each NIC can support multiple wireless virtual interfaces (i.e. vifs) that belong to multiple tenants. The NETCONF and REST interfaces developed in WP3 are used to design a management system to provision and manage vifs for tenants on demand.

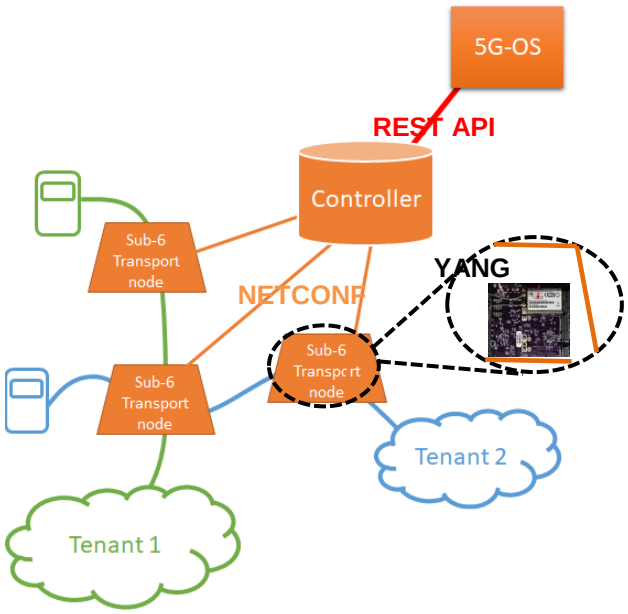


Figure 3-5: example of multi-tenant Sub-6 GHz access/transport network.

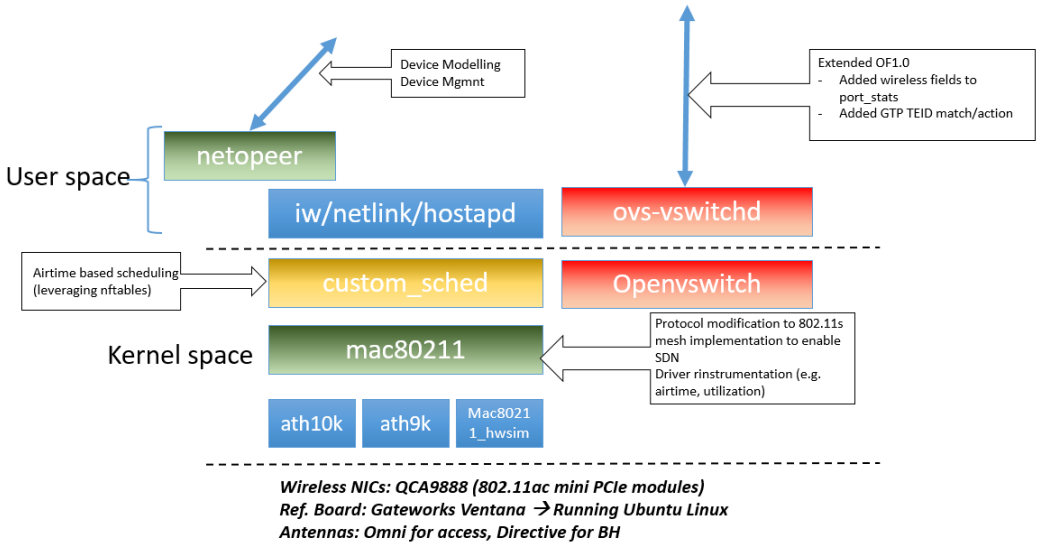


Figure 3-6: Software architecture of the 5G-PICTURE Sub-6 GHz node.

As depicted in Figure 3-5, configurability of Sub-6 GHz transport nodes is achieved through the implementation of different interfaces: i) a northbound interface (solid red line) used to exchange high-level configuration with a controller, based on REST principles; ii) a set of YANG models representing configurable entities in the transport nodes, made available to the controller via NETCONF; and iii) a Transactional API (TransAPI) to implement the system calls needed for those operations on the YANG models to take effect.

3.2.1 YANG and NETCONF

Internally, as shown in Figure 3-6, Sub-6 GHz nodes offer a NETCONF interface through CESNET-Netopeer, the code of which was modified to allow compatibility with GWV's ARMv7 architecture. The API, on the other hand, has been entirely developed by I2CAT from scratch. The TransAPI, a set of resources (e.g. callback functions, scripts, local databases, etc.) used to make effective the commands received through the NETCONF interface, interact with different operating system layers: Linux's user space *iw* tool or netlink sockets to configure NICS' physical parameters; *hostapd* daemon to manage Wi-Fi Access Point (AP) capabilities; scheduling functions at kernel level, etc. Data plane configurability is achieved through Open vSwitch and OpenFlow (OF) protocols but its operation is out of the scope of this document.

The NETCONF interface allows operations on the YANG model used as an abstraction of the Sub-6 GHz node, hereinafter called the ***i2cat-box***, depicted in Figure 3-7. Each Sub-6 GHz node is represented by one i2cat-box. The abstract i2cat-box is composed of different elements:

- A unique identifier.
- Location information (lat. and long. plus a description).
- A set of general configuration parameters, including the possibility to add extensions by means of loadable modules (future work).
- A set of active tenants; that is, a list of tenants who have active network slices defined on this i2cat-box.
- A set of Wi-Fi interfaces, corresponding to the physical IEEE 802.11-based NICs.

The Wi-Fi devices can be used to instantiate virtual interfaces, enabling the operation of the node as a Wi-Fi AP, a wireless backhaul node (a.k.a mesh point), or wireless traffic monitor for different tenants simultaneously. Each virtual interface can be managed individually, with its own network (IP, MAC address, mask) and security configuration. However, some configuration parameters, such as transmitted power or frequency channel and bandwidth, are common to all those vifs, since they directly affect the operation of the physical device.

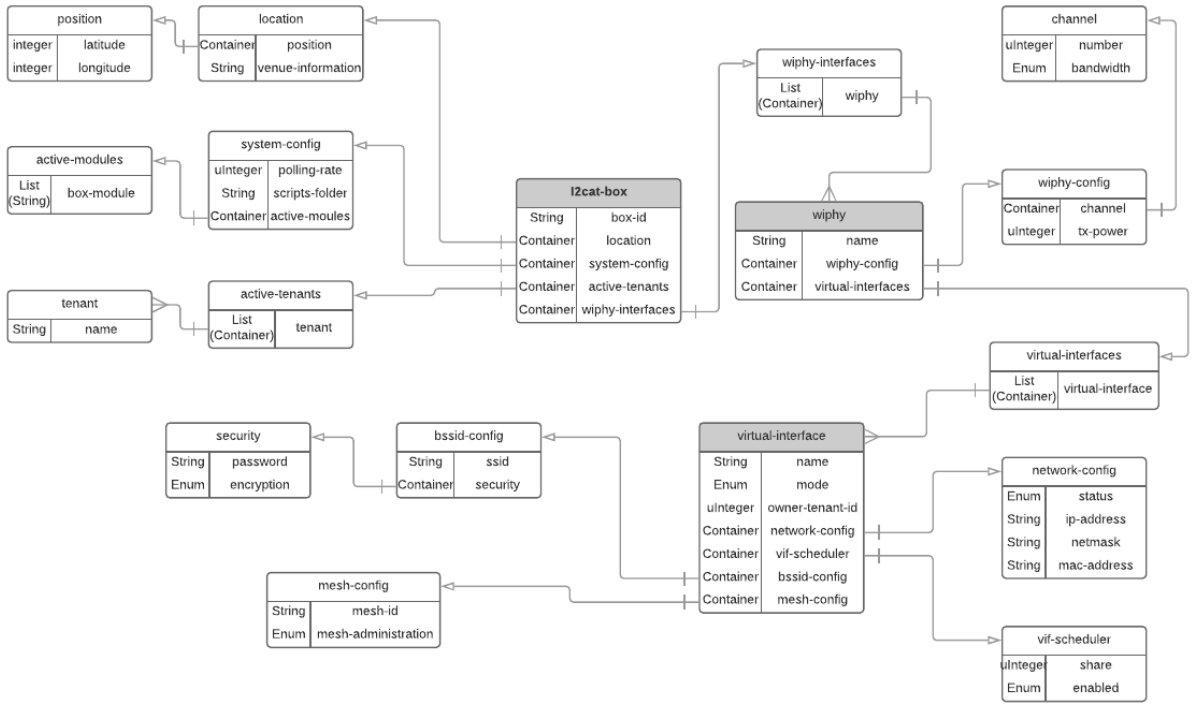


Figure 3-7: YANG model of the I2cat-box.

3.2.2 Northbound REST API

Different tenants or, in general, the 5G OS devised in 5G-PICTURE (see deliverable D5.1 [3]), can access the functionalities enabled by the YANG modelling of the Sub-6 GHz nodes through the controller, which offers a REST API on the northbound interface. The REST operations implemented by the controller are listed in Table 3-4.

Table 3-4: operations on the REST API-based northbound interface of the Sub-6 GHz portion.

Mess.#	Request	Scope	Description
1	GET	/netconfDevices	Get a list of registered devices (i.e. Sub-6 GHz transport/access nodes)
2	POST	/netconfDevices/{id}	Register a new device on the network

3	DELETE	/netconfDevices/{id}	Delete a given device (identified by {id}) from the list of registered devices
4	GET	/netconfDevices/{id}/config	Get a given device's current configuration
5	GET	/netconfDevices/{id}/hostname	Get the device's current hostname
6	PUT	/netconfDevices/{id}/hostname	Set a device's hostname
7	GET	/netconfDevices/{id}/physicalInterface/{phyname}	Get the configuration of a physical interface (identified by {phyname}) in a given device
8	PUT	/netconfDevices/{id}/physicalInterface/{phyname}	Set the configuration of a physical interface (identified by {phyname}) in a given device
9	GET	/netconfDevices/{id}/physicalInterface/{phyname}/virtualInterface/{vifname}	Get the configuration of the selected virtual interface (in a given physical interface of a given device)
10	DELETE	/netconfDevices/{id}/physicalInterface/{phyname}/virtualInterface/{vifname}	Remove a virtual interface (in a given physical interface of a given device)
11	PUT	/netconfDevices/{id}/physicalInterface/{phyname}/virtualInterface/{vifname}/config	Configure network settings of a given virtual interface
12	PUT	/netconfDevices/{id}/physicalInterface/{phyname}/ap/{vifname}	Instantiate a new virtual interface operating as a Wi-Fi AP
13	PUT	/netconfDevices/{id}/physicalInterface/{phyname}/mp/{vifname}	Instantiate a new virtual interface operating as a wireless mesh point (backhauling)
14	GET	/netconfDevices/{id}/tenant/{tenantname}	Returns information of a given tenant
15	PUT	/netconfDevices/{id}/tenant/{tenantname}	Creates a new tenant on a given device
16	DELETE	/netconfDevices/{id}/tenant/{tenantname}	Removes a given tenant from a device

3.2.3 Functional evaluation and performance

This section describes an example of application, where an authorized tenant (or a function of the 5G OS) instantiates a new virtual AP as part of a new slice. Figure 3-8 shows the dialogue between the tenant and the controller (cf. Figure 3-5) over the northbound interface, according to the messages defined in Table 3-4. First message (***#2***) is used to register a new device on the network. This operation is only needed once, for example, when a new node has been deployed. The body of the POST request contains basic information for a generic device in json format, e.g.:

```
{       "internalName": "Ozzy",
        "device": {
                "hostname": "10.0.10.52",
                "userName": "root",
                "password": "1234",
                "port": 830
        }
}
```

As a result, the controller checks its NETCONF connectivity with the newly registered device (NETCONF <*hello*> message exchange), and the device is added to the controller's database. Once the registration takes effect, the controller replies with HTTP 201 (successful creation of a new resource), including the (numerical) ID of the registered device for future reference.

Next, message ***#8*** contains basic configuration parameters of a physical interface, for example:

```
{       "channel-number": 36,
        "channel-bandwidth": 40,
        "tx-power": 26
}
```

The controller will forward the configuration requests received through the REST API to the server running on the managed transport node via NETCONF <*edit-config*> messages. Successful configuration on the device

is acknowledged by means of an HTTP 200 message from the controller to the 5G OS (or tenant). Now, in order to enable the management of a virtual interface by a given tenant, message ***#15*** requests the registration of a new active tenant (e.g. "`operator_1`") on the device (cf. Figure 3-7), upon successful registration, an HTTP 200 response is generated by the controller. Finally, the new tenant can request the instantiation of a new virtual interface (VIF) operating as an access point by means of message ***#12***, which includes basic configuration parameters for an IEEE 802.11 AP such as the service set identifier (SSID) and security aspects; for example:

```
{       "vif-ssid": "MovifonOTE",
        "vif-security": "NONE",
        "owner-tenant-id": "operator_1"
}
```

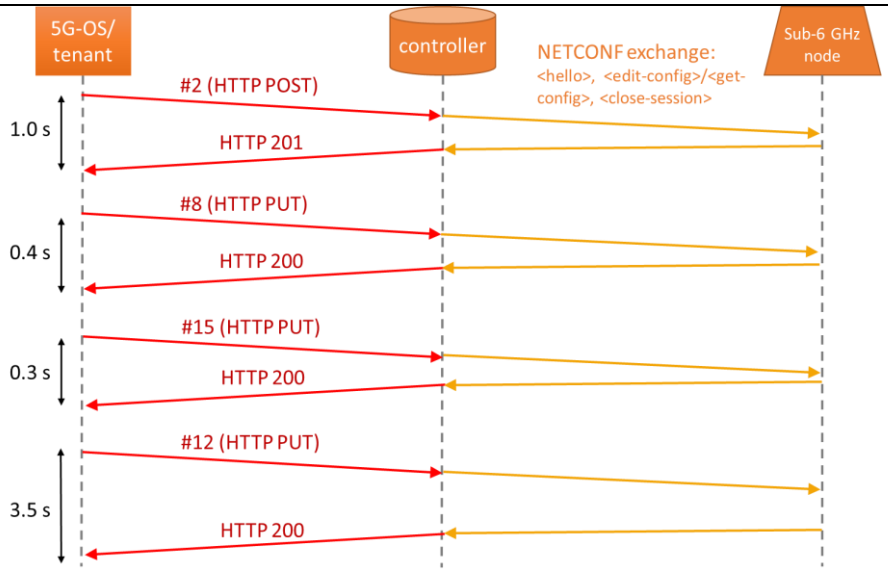


Figure 3-8*:* **Message exchange through northbound API.**

In order to verify the success of the operation, the tenant could send a GET *config* (i.e. message ***#4***) to retrieve the running configuration, and a Wi-Fi AP scan within the coverage area of the node will show the presence of the newly created (virtual) AP. Note that, at this point, the data plane of the new AP still lacks connectivity with the rest of the tenant's slice; network-level configuration is needed (message ***#11***).

An important key performance indicator is the time required to modify parameters of the physical device (message ***#8***) since, radio resource management strategies implemented in the controller (or 5G OS) may need recurrent updates of the frequency channel, bandwidth or the transmitted power. Another frequent operation is the creation (and removal) of a virtual interface, upon request by potential tenants. For those reasons, we measured the time taken to perform what we think are the most frequent and relevant operations requested trough the northbound REST API (which imply the use of a NETCONF interface to manage the YANG model in the device). Figure 3-9 shows the results of 1000 iterations of different functions: a) the blue solid line shows the cumulative distribution function (CDF) of the latency experienced after a PUT request to a physical interface (i.e. message ***#8***) to modify parameters of the device's PHY, with values typically below 400 ms (95% of the time); b) green dotted line represents the CDF of the time required to create a new virtual interface in AP mode (message ***#12***), a costly operation for the managed device, with values between 3.5 and 3.6 s; and c) dashed red line shows the latency of a delete operation (message ***#10***), intended to remove an existing virtual interface.

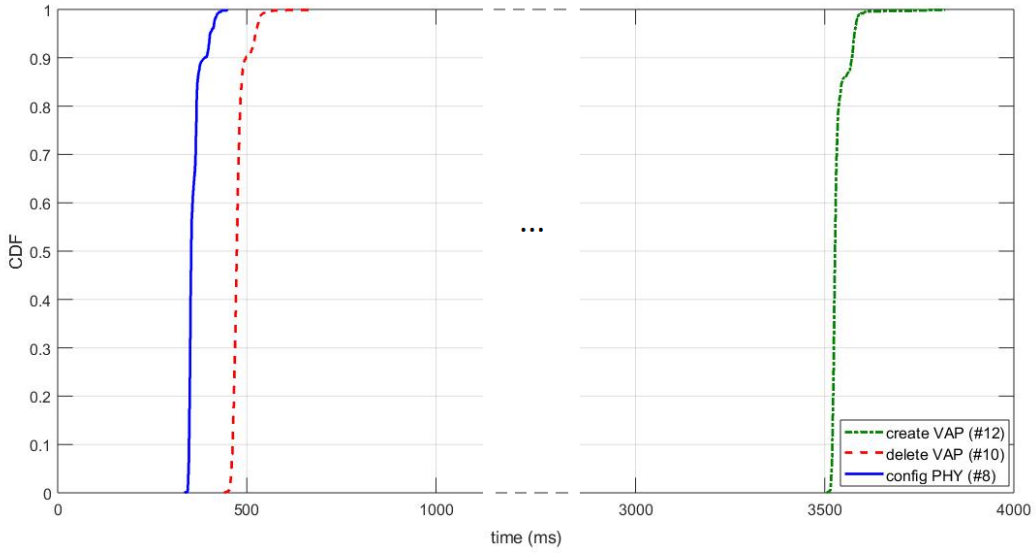


Figure 3-9*:* Latency analysis (CDF of 1,000 runs) of three operations over Sub-6 GHz northbound API.

3.3 OAI Interfaces for Physical Network Functions

In this subsection we present the current implementation of the OAI Interfaces for Physical Network Functions. We focus on the initial setup of the VNFs (e.g. configuring the interconnection between the RAN VNFs and the Core Network, configuring the PLMNIDs, starting/stopping the OAI executables). The agents will be further extended during the project in order to introduce more programmability to the infrastructure, like adding new subscribers to the core network, configuring RAN parameters (e.g. operating bandwidth, frequency band, configuring the latency on the fronthaul link for emulating longer distances, configuring different functional splits etc.) and able to be used through tools like Cloud-Init when launching the VNFs. While the initial functionality will be demonstrated during the Athens Review meeting, the extensions that will be developed will be described in Deliverable D3.3. UTH will also extend the agent API for bootstrapping the OAI related VNFs built in WP4 in order to introduce new features.

OAI is providing a state-of-the-art solution for 5G prototyping based on Open Source code. To this aim, and as OAI is the target platform for application of the project's developments, an API shall be present for the configuration of the disaggregated base station instance as either a PNF or a VNF. This API will be used for the initial bootstrap of the VNFs/PNFs, through a REST based interface that can be invoked by tools like Cloud-Init[23] or JuJu[24]. Currently, there are several efforts that introduce network programmability to OAI. One notable addition is the implementation of the Small Cell Forum's network functional API or nFAPI interface[25]. nFAPI defines a network protocol that is used to connect a Physical Network Function (PNF) running LTE Layer 1 to a Virtual Network Function (VNF) running LTE layer 2 and above. The development of the nFAPI interface provides an open interface between LTE layer 1 and layer 2 that allows for interoperability between the PNF and VNF and also to facilitate the sharing of PNF's between different VNF's. Similarly, other splits of the platform are currently available (e.g. 3GPP option 7-1 and 7-2) or being developed (3GPP option 2).

In this context, the creation of a programmable API for the configuration of the OAI instances is of great importance. The existing FlexRAN API is solely targeting in changing parameters regarding the RAN configuration, on the fly, and post deployment. Nevertheless, the creation of a generic tool that handles and configures the different OAI instances to intercommunicate with each other is critical during the initial bootstrapping of the VNFs. To this aim, we introduce the *bscontrol* network service, which through the latest extensions can handle OAI-based base stations and core networks.

[23] https://cloud-init.io/

[24] https://jujucharms.com/

[25] https://github.com/cisco/open-nFAPI

The *bscontrol* service has been developed by UTH over the years with the purpose to handle different base stations with a single API. Through this API, the tool is able to configure different types of base stations, depending on their configuration mechanism (e.g. SNMP/TR-196). In 5G-PICTURE, the tool has been further extended to support the configuration of OAI base stations and core networks during the VNF instantiation. The tool has a REST API and can configure the different components in a unified fashion. Therefore, during the instantiation of an OAI VNF, the tool is able to automatically retrieve addressing information about the VNF, and appropriately configure the VNF instance so that it can communicate with the rest of the components for the following cases:

1. A BS communicating with the core network, e.g. eNB to the Mobility Management Entity (MME).
2. Disaggregated BS components communicating with each other, e.g. Central Unit (CU) to Distributed Unit (DU).
3. Disaggregated core network components communicating with each other, e.g. MME to Home Subscriber Server (HSS).

It is worth to mention that the tool can handle other parameters, which by default are user-defined in OAI, such as determining the level of split that will take place in the base station. The tool can be instantiated along with OAI through platforms like Cloud-Init or JuJu, thus allowing its further integration with other platforms. The Figure 3-10 is showing a complete example for a base station instantiation through *bscontrol*. The corresponding 5G OS that is being developed in WP5 will detail the resources as either Network Service Descriptions (NSDs) or VNF Descriptions (VNFDs), based on the VNFs that are developed in WP4. The tool can be orchestrated through Cloud-Init, and post its installation automatically bootstraps the operation of OAI. After its installation, the tool exposes a REST API, so that further configuration of the base station (e.g. on the wireless parameters) can take place.

In deliverable D3.3, the agent API will be extended for bootstrapping the OAI related VNFs built in WP4 in order to introduce new features. The current implementation focuses only on the initial setup of the VNFs (e.g. configuring the interconnection between the RAN VNFs and the Core Network, configuring the PLMNIDs, starting/stopping the OAI executables). The agents will be further extended in order to introduce more programmability to the infrastructure, like adding new subscribers to the core network, configuring RAN parameters (e.g. operating bandwidth, frequency band, configuring the latency on the FH link for emulating longer distances, configuring different functional splits etc.) and able to be used through tools like Cloud-Init when launching the VNFs. The initial functionality will be demonstrated during the Athens Review meeting.

3.4 Programming languages for data plane programmability

In deliverable D3.1 [1] we identified two possible programming languages for the development of portable network functionalities. The first one is P4[26], a programming language recently proposed to allow programming of packet forwarding planes. In contrast to a general purpose language such as C or Python, P4 is a domain-specific language with a number of constructs optimized around network data forwarding. The second programming language identified is the Open Computing Language (OpenCL[27]), a unified programming framework for developing parallelizable numeric computing algorithms for different processing architectures like CPUs, GPUs, and FPGAs with a single code base, using a specific subset of either C or C++ programming language syntax together with native vectorial data types. The two languages focus on different aspects of the network programmability, since P4 is oriented toward the processing of packets and the programmability of SDN networks, while OpenCL is suitable to describe digital signal processing algorithms that could be useful to support different RAN functional splits. While we identified several use cases and programmable platforms that could leverage the use of the P4 programming, we don't yet found a concrete network function to use in the 5G-PICTURE network architecture that could actually benefit of a OpenCL description. This is due to two main reasons: (i) the overhead of developing function in OpenCL is not negligible, since require the use of specific development frameworks such as the Xilinx SDAccel kit and (ii) porting to other hardware platform based for example on DSP is not straightforward. Therefore, the 5G-PICTURE consortium focuses the work described in this WP only on the P4 related activities.

[26] www.p4.org

[27] www.opencl.org

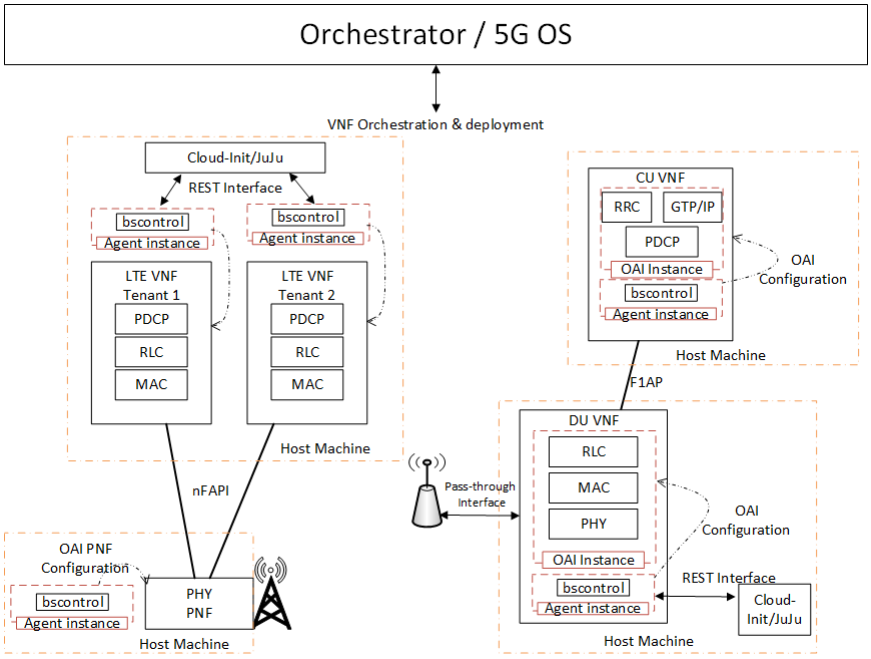


Figure 3-10: Bootstrapping of different OAI VNFs through the bscontrol tool.

3.4.1 P4 compiler

3.4.1.1 Development of P4 compiler for Spectrum device

Mellanox will use their Mellanox Spectrum™ Ethernet Switch as target platform for the development of the P4 compiler for programmable dataplanes. In particular, Mellanox is designing and developing a compiler for the P4_16 version.

3.4.1.2 Mellanox P4 Compiler main components

The main components of the P4 compiler that were implemented as part of WP3 are:

1) Mellanox P4 compiler hybrid target architecture- application sandbox:

A hybrid target architecture model has been created in P4 and a specific compiler backend was developed based on this model.

The ***hybrid*** target pipeline architecture enable users to use all the legacy HW pipeline and control protocol functional E.g routing, bridging, while providing a way extend it using p4-16

This concept well implemented on top of two network OS:

- Onyx[28] – Mellanox commercial network OS.
- SONiC[29]- an open Source network OS.

[28] www.mellanox.com

[29] https://azure.github.io/SONiC/

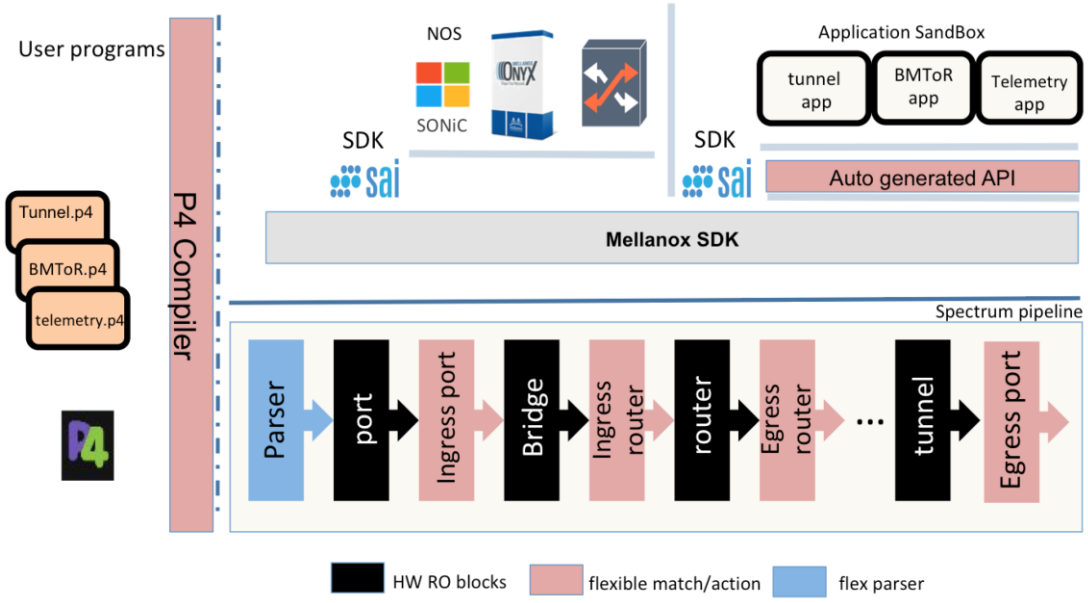


Figure 3-11: Mellanox hybrid target pipeline.

2) P4 APIs

The Mellanox P4 compiler currently generates three different layers of APIs. Each layer produces a set of shared libraries and/or C header files that applications can dynamically link against.

a. Mellanox SDK API

The base flexible ("fx") layer provides an auto-generated set of APIs based on the defined P4 program. The implementation is dependent on the Mellanox Switch SDK library to control and program the Spectrum ASIC. A user familiar with the SDK or in need of specific hardware resource control may build an application based on this API layer, with much of the mundane SDK programming details handled automatically.

b. Flex SAI API

The flex SAI layer auto-generates a set of SAI compatible APIs that extend match action table functionality programmatically based on the P4 program. The Mellanox P4C compiler generates C header files plus a library of functions that convert SAI objects to SDK objects.

c. P4Runtime API

The P4Runtime API layer auto-generates a set of P4Runtime APIs and a Spectrum target specific shared library. The P4Runtime APIs are based on p4lang/PI objects, while the auto-generated target specific shared libraries allow applications to be built using a stable ABI that is independent of the P4Runtime info itself. For example, a P4Runtime agent that is loaded on the Spectrum switch control plane, based on protobuf and gRPC, is an application built on top of these two layers. Details such as pipeline configuration and match action specifics are embodied in the base "fx" layer.

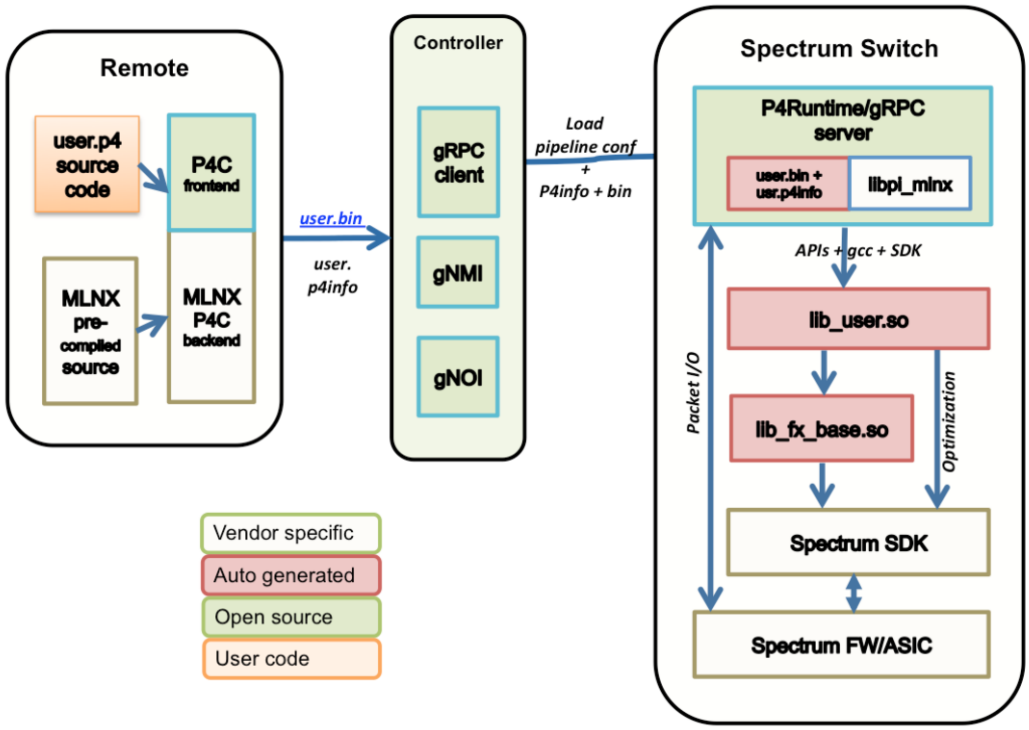


Figure 3-12: Mellanox autogenerated APIs.

3.4.1.3 Mellanox P4 Compiler

Architecturally, the Spectrum P4 compiler is based on a hybrid model architecture with a discrete pipeline. Well known, fixed functional blocks (parsing, ingress L2 bridging, ingress/egress L3 routing, egress port) are performed at discrete points in the pipeline, and are not programmable (although may be configurable). The programmable blocks are invoked in between each of these fixed function blocks. The Mellanox P4 Compiler generates output that realizes this architectural model. The clear advantages are that the programmer need not redefine and re-develop well know concepts such as IP packet parsing and routing tables.

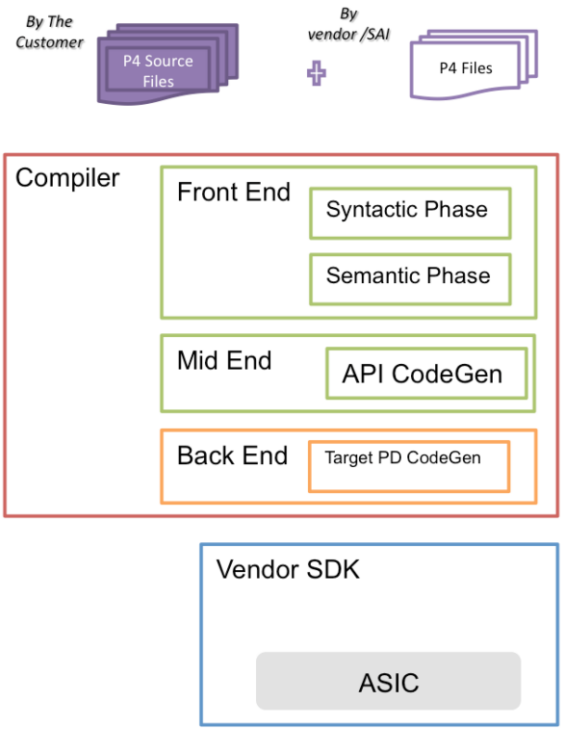


Figure 3-13: Mellanox P4 compilation flow.

From the implementation standpoint, the Mellanox P4 Compiler consists of a front-end, mid-end and back-end stages. The front-end processes the syntax of the P4-16 language, and outputs a series of symbol tables. The mid-end verifies the semantics of the P4-16 program as well as performing some transformations and reductions to make it easier for the backend. This layer also generated the platform independent (PI) P4Runtime info output. Both front-end and mid-end code is taken from the open source p4lang project, without modification. The back-end compiler is a target specific (Spectrum) code generation stage that generates the actual shared libraries and SDK code from the symbol tables generated from the first two stages (in an annotated BMV2 JSON format). A subsequent back-end pass converts this to the actual target binaries. As the Mellanox P4 Compiler is typically deployed off box, these compiler artefacts are then sent either directly to the switch agent or to a network controller in charge of configuration and deployment.

3.4.1.4 Mellanox P4 API

As described in section 3.4.1.2 part 2), the Mellanox P4 compiler generates 3 separate types of APIs. Each API targets a different type of development environment. The Mellanox SDK API supports the SDK developer by generating easy to use APIs and hides a large part of the complexity and boilerplate coding, allowing the programmer to focus more on the business logic at hand. The Flex SAI API allows the SAI developer to extend the standard SAI APIs. P4Runtime API is used to support SDN use cases, where the pipeline is described in the P4-16 language and table entries are provisioned dynamically from an SDN controller using P4Runtime info generated by the compiler.

3.4.2 ADVA P4 Xilinx FGPA development workflow

As first step towards enabling high-level programmability of the white-box edge device prototype developed in 5G-PICTURE, ADVA has established a P4 programming workflow for the IAF 5G Development and Prototyping Platform F-PU-5G[30], which was first presented as most promising ADVA NFV hosting platform for 5G-PICTURE in deliverable D3.1 [1] section 2.10.

[30] http://www.iaf-bs.de/en/products/fpga-motherboards/F-PU-5G

This P4 workflow is based on the Xilinx SDNet[31] development environment as the centrepiece of P4-based FPGA design, which involves two compilers:

- **P4-SDNet Translator** [35] -- Currently only supported on 64-bit Linux operating systems. It is used in conjunction with the SDNet tools to compile $P4_{16}$ code.
- **SDNet Compiler** [36] – It compiles P4 SDNet programs that target the XilinxSwitch architecture into a Verilog module that has standard AXI (Advanced eXtensible Interface) [37] Stream packet interface and an AXI-Lite control interface.

The generated Verilog modules are then integrated in the existing FPGA design using Xilinx Vivado Design Suite[32]. Once the project is configured and all source files have been added, Vivado synthesis and implement processes is run in order to generate the target FPGA binary.

A C-API is generated directly by SDNet Development Environment and can be used to manipulate the registers and tables instantiated in the P4 program. The C-API is integrated in the software of MicroBlaze Controller using Xilinx SDK development environment and run as a driver for programming the design's P4 match-action tables.

P4 Runtime[33], a new open source project, first described by Google at 2017 P4 Workshop[34] as P4 Program-Dependent Controller Interface for SDN Applications[35], is a protocol to control at runtime a P4-defined pipeline, for example, to install entries in a table in the P4 program. P4 Runtime will run on an ONOS Controller that is connected to the FPGA through a 1Gbit Ethernet interface.

A visual overview of the complete P4 workflow is shown in Figure 3-14. A specific use case with performance evaluation is presented in section 4.8.

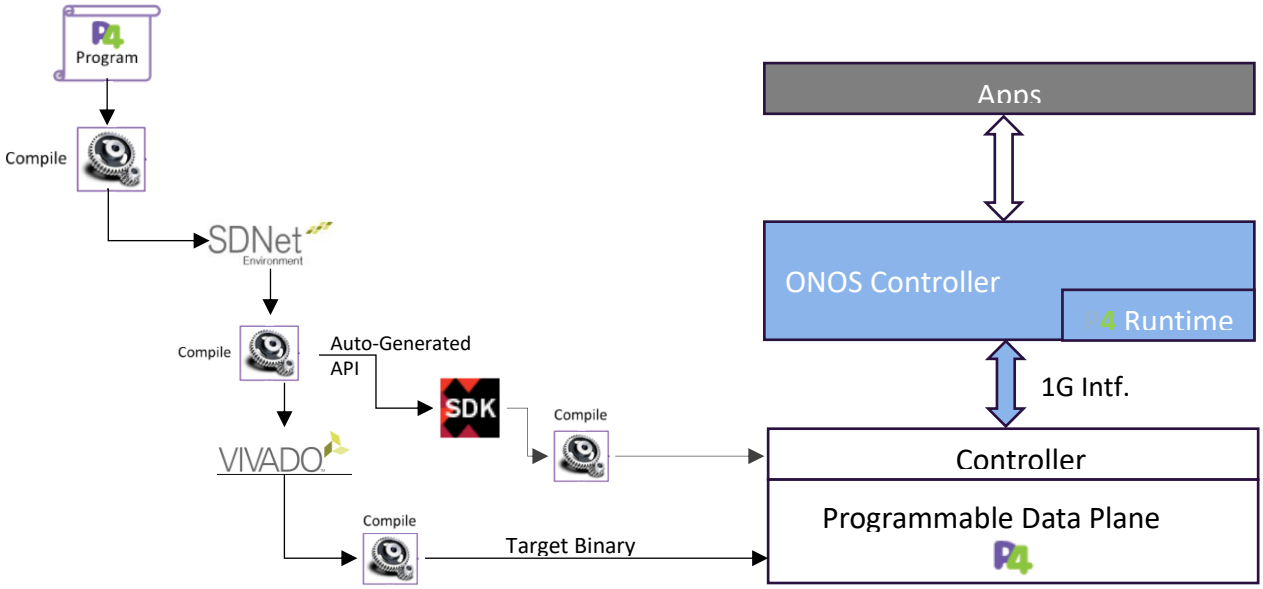


Figure 3-14: ADVA P4 Xilinx FPGA development workflow for F-PU-5G.

[31] https://www.xilinx.com/products/design-tools/software-zone/sdnet.html

[32] https://www.xilinx.com/products/design-tools/vivado.html

[33] https://p4.org/p4-runtime

[34] https://p4.org/events/2017-05-09-p4-workshop

[35] https://p4.org/assets/p4-ws-2017-p4-program-dependent-api-for-sdn-applications.pdf

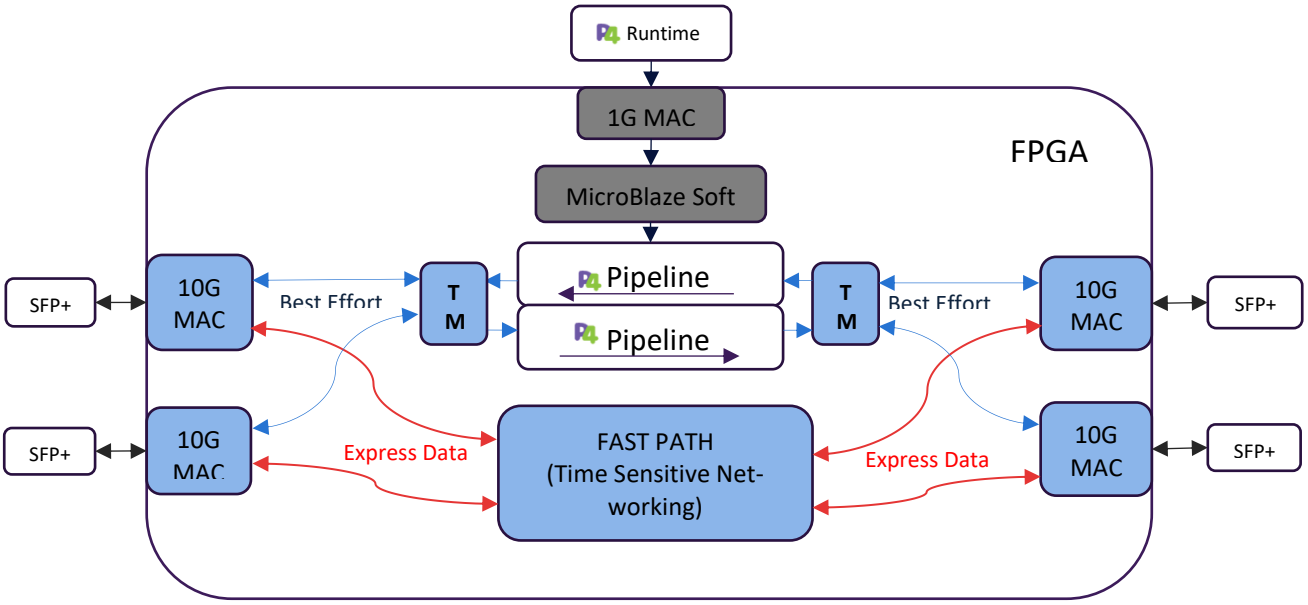


Figure 3-15: ADVA P4 Pipeline programming on F-PU-5G platform.

One of the advantages of using an FPGA for implementing P4 is that in an FPGA Device offers the flexibility to program and integrate low latency HDL components for critical paths, e.g. in a time sensitive networking design.

The design shown in Figure 3-15 below describes a P4 assisted FPGA design, that implements the IEEE 802.1CM[36] standard (Time-Sensitive Networking for FH). Therefore, in order to achieve the low and deterministic latency criteria of the express data traffic, a non-P4-programmable fast path HDL (Hardware Description Language) component will be required.

In a multi-interface design, for a complete end-to-end solution the P4 pipelines are not enough. Switch configuration, including port management and traffic management for tasks like scheduling, queueing, congestion control and replication, should also be taken into consideration. Traffic management components are currently not programmable in P4 for two reasons:

- A traffic manager has tight timing requirements not easy to meet by a software abstraction.
- There are variety of scheduling algorithms out there, but there is no consensus on the right abstraction.

3.4.3 XTRA language

XTRA lang[37], is a *domain specific language* for programming portable transport layer protocols and application, using an Extended Finite State Machines (XFSMs) abstraction.

XFSMs seem a *natural* way to describe *stateful processes*, like transport layer protocols, thanks to the ability of managing asynchronous events, e.g. timers and packets arrival. Moreover, it is the right abstraction for HW implementation, because they can meet the crucial requirement of having a time-bounded state transition as, in essence, they require "just" the time of performing a *match/action* table lookup.

Implementing a protocol using an XFSM means writing a set of *if-then* rules, as in Figure 3-16. In the *if* part, there are three type of information: the state label (a string, like "*wait*"), the event which triggers the rule (e.g. *TimerExp(A)*), and the further conditions which must be verified in order to make the transition (*A > 7 && Reg1 == 2*). The effects of this rule are described in the then part, where there is a list of actions to execute, like sending a packet or setting a timer, the next state on which the XFSM has to go (e.g. "*active*"), and a set of updates on the internal registers of the XFSM itself (*Reg2 = hash(C)*).

[36] https://1.ieee802.org/tsn/802-1cm

[37] http://netprog.uniroma2.it/xtra/

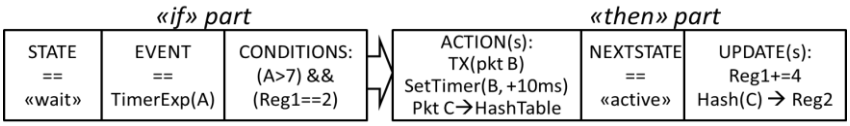


Figure 3-16: Example of XFSM rule entry.

The set of rules are organized in a table but, unfortunately, these tables are not easy to read and write. Therefore, it comes the necessity of having a language to easily describe them.

XTRA lang, is the language developed that allows XFSMs table specification. The main benefit of using XTRA lang is having platform agnostic language, which gives to the programmer the ability of having a "*code-once-port-everywhere*" code. This code can be executed without modifications on every platform supporting XTRA, even on HW. Another great characteristic of XTRA lang is its clear notion of state. In fact, the behaviour of a protocol/application (i.e. how its state evolves in time), can be decoupled from the events which cause this evolution, and the actions that are triggered by this state transition.

To develop a protocol/application, the programmer has to write only the code in XTRA lang. This code is compiled using *xtrac* (the XTRA lang compiler, written in Java, using the ANTLR library), in a JSON file representing the table entry, with all conditions and registers used. Then, this JSON file can be loaded into the different platforms supporting XTRA, without any need to change it (see Figure 3-17). For every platform, HW or SW, there is a different loader that configures the execution engine to execute the algorithm specified in the file.
A program written in XTRA lang begins with the declaration of the registers the programmer will use, using the *Register* keyword (see Figure 3-18), and the macro-action definition. A macro-action is a combination of primitives and/or other macro-actions that can be called inside the program, like a procedure call, reducing the redundancy of the code (see Figure 3-19).

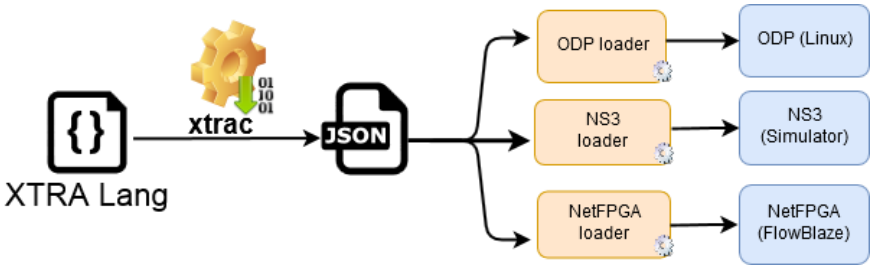


Figure 3-17: XTRA lang workflow.

```
Register cwnd, lastAckedSeq, availWin, nextTxSeq, rtt;
```

Figure 3-18: Register declaration in XTRA lang.

```
State slowStart { // actions with event and conditions
   on (timeout) {
      if (timeout.data1 <= 0, availWin >= 1448) {
         sendAndUpdate();
         updateAvailWin();
         setTimer(nextTxSeq, 0, 1);
      }
   }
}

State syn { // actions without conditions
   on (timeout) {
      sendPacket(1, 1);
      setNextState(ack);
   }
}
```

Figure 3-19: An example of macro-action declaration and call.

To check the value of a register, XTRA lang provides the *debug* directive, which permits to print the value of the register on a desired interval of time.

```
#debug cwnd 500000
```

Figure 3-20: The debug directive prints the value of the register "cwnd" every 500000 µs.

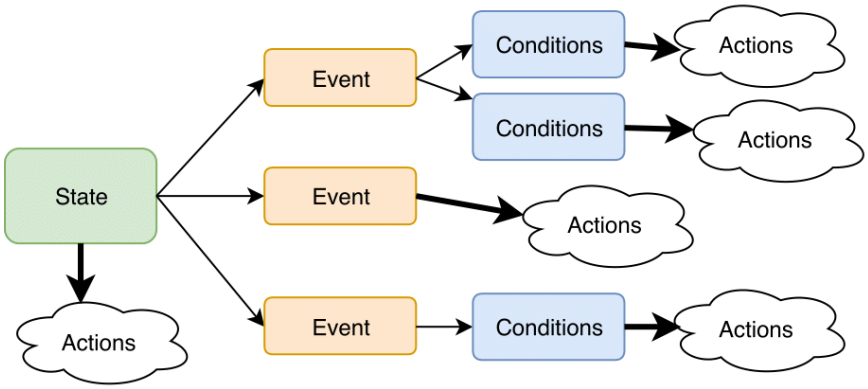


Figure 3-21: State definition in a typical XTRA lang program.

```
State slowStart { // actions with event and conditions
   on (timeout) {
      if (timeout.data1 <= 0, availWin >= 1448) {
         sendAndUpdate();
         updateAvailWin();
         setTimer(nextTxSeq, 0, 1);
      }
   }
}
```

```
State syn { // actions without conditions
   on (timeout) {
      sendPacket(1, 1);
      setNextState(ack);
   }
}
```

Figure 3-22: An example of state declaration (slowStart), with an event (timeout) and two conditions in the *if* clause.

```
State initial init {
   nextTxSeq  =  1;
   lastAckedSeq  =  1;
}
```

Figure 3-23: An example of state marked as initial.

After register declarations, macro actions definition and the debug directives, an XTRA lang program is a list of all the states of the XFSM, with all the events and conditions considered for every state. The structure of a state definition in XTRA lang, is shown in Figure 3-21, and follows a *State-Event-Conditions* structure.

It is possible to specify some actions to be executed every time the XFSM enters into a chosen state (the state is declared with the *State* keyword). Inside the state it is also possible to specify the behaviour of the state machine when a specific event is happening (e.g. a packet arrival or a timer expiration), associating some actions to that event to be executed every time happens this specific event (the event is specified with the *on* clause). Inside an event it is also possible to specify some conditions. When these conditions are satisfied they cause the execution of the actions in that condition block. The conditions are a set of Boolean expressions connected with AND, and are expressed with the *if* clause.

One state must be marked as initial, and it will be the state on which the XFSM will start. This is usually used for registers initialisation and for starting the protocol/application.

The possible events are the arrival of a packet (*pktRcvd*), the expiration of a timer (*timeout*) or a call from the OS (*connect, appData*). The events can carry some data, like for instance *timeout.data0*, *timeout.data1* (custom data associated with the timer when is set) or *pktRcvd.port* (the port on which the packet has been received).

It is also possible to access and manipulate the fields of the packets on the whole protocol stack like, for instance, the sender's IP address (*ipv4.src*) or the TCP source port (*tcp.sport*).

XTRA lang permits also to perform updates on the declared registers. It is possible to use in the operations, registers, packet fields, data carried by the events and constants. For the updates the basic arithmetical operations are available (+, -, *, /), as well as the maximum and the minimum between two operands. The supported primitives are:

- *sendPacket(pktRef, iface)* – send a packet refered by *pktRef* on the *iface* interface.
- *setTimer(data0, data1, timeout)* – set a timer with a given timeout, with associated some custom data.
- *storePacket(pktRef)* – store a packet in a local key-value store.
- *setField(headerField, pktRef, value)* – set a field in the packet *pktRef.*
- *deletePacket(pktRef)* – delete the specified packet from the key-value store.
- *setNextState(state)* – set the next state on which the XFSM will go after the transition.
- *deleteInstance()* - delete the XFSM instance.

The translation between the XTRA lang source and the JSON representing the XFSM is done using *xtrac*, the XTRA lang compiler, used as shown in Figure 3-24.

An XTRA lang JSON file consists of an object containing three fields: *"table_rows"*, which contains a JSON array of table entries; *"initial_state"*, which contains the label of the state marked as initial; and *"conditions"*, which contains all the conditions used in the program, with an unique id, and they are referred into the table entries.

As an example of the translation in JSON, a little snippet is shown in Figure 3-25, corresponding to an XTRA lang program. This is translated with a single table entry, as in Figure 3-26. In the field *"condition_results"* there is an array of the conditions that must be verified to execute the actions. For every condition there is the id (which references to the conditions listed at the beginning of the JSON) and their result. All the conditions not listed are intended as *don't care*. In the field *"state"* there is the label of the state of this entry, and in *"event"* the event which will trigger this table entry. Then in actions there is an array of the actions, which are executed by this entry. Every action has its *"opcode"* field and can have some operands (e.g. *"op0"*, *"op1"*) that can be registers, constants, packet or event fields. For distinguishing the different type of operands there is a type field (e.g. *"op0Type"* for the operand *"op0"*).

```
$ java -jar xtrac.jar -i input.xtra -o output.json
```

Figure 3-24: The xtrac usage.

```
State syn_rcvd {
   on (pktRcvd) {
      if (tcp.flags == 16, pktRcvd.port == 1) {
         sendPacket(0, 2);
         setNextState(syn_sent_to_server);
      }
   }
}
```

Figure 3-25: A simple state with one event and two conditions. This will be translated in a single table entry.

```
{
   "condition_results": [
      {
         "id": "2",
         "result": "XTRA_TRUE"
      },
      {   "id": "1",
          "result": "XTRA_TRUE"
      }
   ],
   "state": "1",
   "event": "XTRA_PKT_RCVD",
   "actions": [
      {
         "opcode": "XTRA_SENDPKT",
         "op0": "0",
         "op0Type": "XTRA_CONSTANT",
         "op1": "2",
         "op1Type": "XTRA_CONSTANT"
      },
      {
        "opcode": "XTRA_STATE_TRANSITION",
         "op0": "2"
      }
   ]
}
```


Figure 3-26: The JSON translation of the table entry in Figure 3-25.

The two conditions of the table entry in Figure 3-25 are translated as shown in Figure 3-26. The condition with id 1 corresponds to *pktRcvd.port == 1*; and the one with id 2 to *tcp.flags == 16*. The *"opcode"* field, is one of the six comparisons allowed by XTRA lang (==, !=, <, <=, >, >=), and in *"op0"* and *"op1"*, there are the two operands used in the condition, which can be any of the type allowed by XTRA lang.

3.5 NETCONF/YANG service for ADVA's passive WDM platform

In 5G-XHaul deliverable D3.3 [33] section 2.5, ADVA already presented a working prototypical NETCONF/YANG service implementation and specific YANG module for the passive WDM platform developed in that project (termed WDM-PON therein), including implementation code samples, which are already based on ADVA's NETCONF/YANG service development framework described in section 2.7.3.

In 5G-PICTURE, this passive WDM platform will be reused, details are given in Deliverable D3.1 [1]. For this purpose, the existing NETCONF/YANG service of the head-end control plane is currently further improved as part of ADVA's 5G-PICTURE development activities. A new YANG module is defined to reach better integration with SDN orchestrators ODL and ONOS, and to reach the interoperability with TransPacket's FUSION platform and its specific YANG modules as mentioned in section 2.3.2.

This new YANG model will again be mainly based on the IETF YANG Data Model for Network Interface Management[38], extending the `/ietf-interfaces:interfaces/interface` list with network interface statistics and configuration options for the optical head-end and tail-end ports and any network interfaces of TransPacket FUSION nodes connected to the system.

[38] https://tools.ietf.org/html/rfc7223

4 Hardware technologies: implementation results and performance evaluation

This section describes optical and radio technologies and components, which provide the integrated physical network infrastructure used in the 5G-PICTURE project. While section 5 of deliverable D3.1 [1] presented the initial design of the hardware technologies, this deliverable discusses the achieved implementation results and shows the testbed implementation of the developed technologies and provides a preliminary performance evaluation. In particular, this section describes: i) the passive optical technologies based on the WDM-PON and active technologies based on the elastic frame-based optical networks, ii) the development of time sensitive Ethernet technologies, iii) the Flex-E technology together with its testbed implementation, iv) the description of the X-Ethernet platform and its testbed; v) the channel modelling and comparison between Sub-6 GHz and mmWave frequencies, vi) the Active Antenna Distributed Unit (AADU) architecture with functional split options, vii) use of MIMO technologies at mmWave wavelength and viii) experimental P4 use case for the performance evaluation of a fully programmable white-box edge device based on the F-PU-5G FPGA board.

4.1 Passive Optical technologies

In deliverable 3.1 [1] section 5.1, ADVA proposed the reuse of new passive WDM technologies under development as part of the 5G-XHaul project. The final outcome of this development activity is a fully functional G.metro[39] (ITU-T G.698.4) compliant passive WDM transport platform, which was eventually successfully demonstrated at the end of that project. The final demonstration setup and results are displayed and described in 5G-XHaul deliverable D5.3 [34] in section 3.1.

The HW level of that system is therefore ready to be integrated in 5G-PICTURE demonstration network infrastructures in Work Package 6 (WP6). Its existing prototypical NETCONF/YANG-based SDN control plane is currently improved and adapted for the needs of 5G-PICTURE, as explained in section 3.5.

4.2 Time sensitive Ethernet

Within the area of TSN for use in 5G networks, aggregation of traffic with low latency in FH is being investigated along with synchronisation capability using PTP. Furthermore, for evaluating a converged infrastructure, techniques for combining latency sensitive FH traffic with BH traffic is being investigated. As part of this work, a field-trial evaluation has been conducted.

In the context of 5G-PICTURE, a field trial evaluation of low-latency and timing-accurate 100G Ethernet Aggregator for Converged Mobile X-haul has been demonstrated in the European Conference on Optical Communication (ECOC) 2018, which took part in Rome (Italy) in September 2018.

The overall setup of the field testbed is depicted in Figure 4-1, where five different FH and BH services were aggregated and converged onto a single metro 100 Gb/s fibre link by leveraging a flexible FPGA prototyping platform based on Xilinx Ultrascale VCU 110 evaluation boards, which included the Integrated Hybrid Optical Network (IHON) and Ethernet MACs IP Cores. Two identical 100G aggregator nodes were deployed at the Central Office (CO) and remote node (RN), respectively. The metro link used in the demonstration test is shown in Figure 4-2. In the test scenario, the link was looped at the terminal BS location and back to the CO, for achieving an optical path of 6 km. Such a length is typical for backhauling mobile connections and for C-RAN FH, covering the range of a city C-RAN. This introduced 30 µs of propagation delay in the link, representing a realistic scenario for the equipment under test.

The converged services include: (i) a CPRI-to-Ethernet mapper developed to transport CPRI streams in Ethernet frames (3GPP Option 8). The size of the Ethernet frames can be dynamically configured, while each payload of the Ethernet frame consists of an internal header information field and 2^n CPRI basic-frames; (ii) a 10GbE BH traffic less time-critical than the FH traffic; (iii) a 60 GHz digital unit and RU implementing a lower layer split (LLS, 3GPP Option 7) and operating at >2 Gbit/s throughput; (iv) a full LTE virtual RAN (vRAN) digital unit/RU setup with MAC-PHY LLS (3GPP Option 6) running at 0.4 Gbit/s; (v) a 4G-LTE femto-cell access at the RN connected to the *Telekom Slovenije* core network via the 100G metro fibre link.

[39] https://www.itu.int/rec/T-REC-G.698.4-201803-I

To enable the synchronisation between the CO and RN, a grand master clock compliant with the ITU-T G.8275.1 profile received the reference from GNSS satellites, and the IEEE 1588v2 PTP traffic was then broadcasted to all the remote 10GbE client ports. At the RN, the PTP slave signal was input at a 1GbE port. The CPRI-to-Ethernet demapper used the PTP stream for generating a 10 MHz synchronisation signal.

To minimise the Packet Delay Variation (PDV), the most critical parameter for PTP delivery, we investigated an implementation of IHON called FUSION. The FUSION IP core maintains ultra-low PDV for high-priority Guaranteed Service Transport (GST) traffics by preserving the gaps between the GST frames and filling lower-priority statistically multiplexed traffic (SMT) into vacant GST inter-frame gaps. This eliminates the need for any central scheduling, compared to time-aware shaping. In the following section, we will analyse the performance of the PTP stream transported in the GST class and its impact on the SMT.

A VIAVI MTS-5800 tester with a separate GNSS input analysed the PTP stream received at the second 10GbE egress port at the RN. PDVs of the sync and delay request messages are shown in Figure 4-3. In a 19.5 hours long test (with client services loaded), the average PDVs of both sync and delay request messages are around 100 ns. Characterisations in the lab have shown that this PDV includes a 50 ns and 27 ns component attributed to the 10GbE MAC/PHY and the 100GbE MAC/PHY, respectively. Hence, the maximum additional PDV introduced by the IP core is 19 ns.

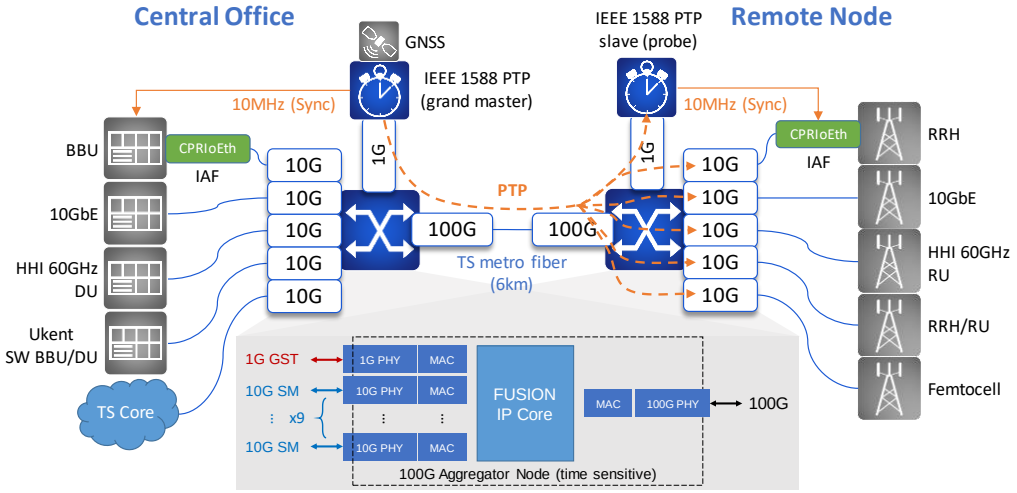


Figure 4-1: Field-trial setup demonstrating multiple radio and split approaches over a converged time-sensitive Ethernet link (inset: Breakdown of the prototyped 100G time-sensitive aggregator).

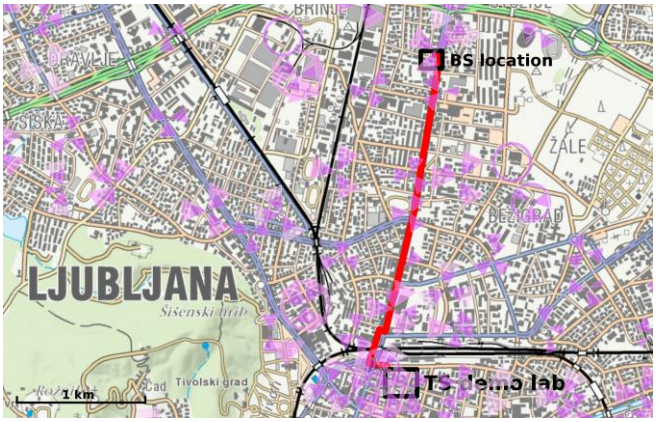


Figure 4-2: Metro fibre link (in red) used in the field trial.

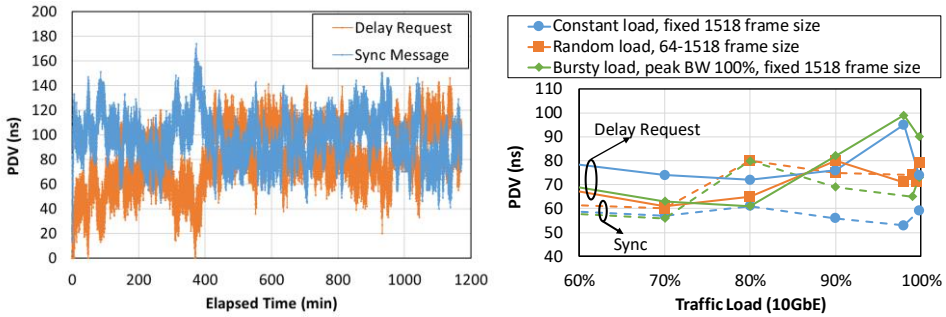


Figure 4-3: Left: PDV analyses at the remote 1G and 10G ports. Right: PDV versus 10GbE load with different traffic patterns.

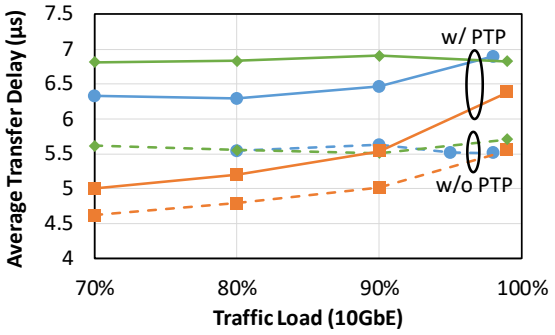


Figure 4-4: Average round-trip transfer delay of SMTs.

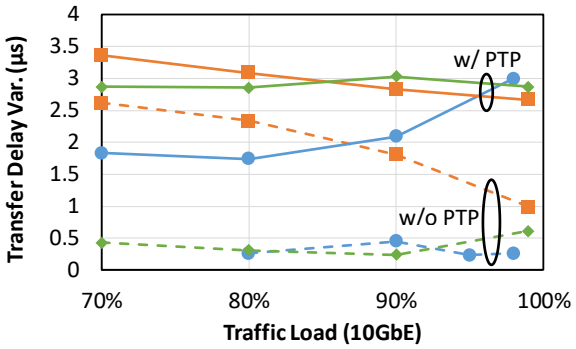


Figure 4-5: Transfer delay variation of SMTs.

Different SMTs were loaded together with the PTP stream on the 100GbE metro link. Three different SMT patterns were investigated: (i) constant load with a fixed frame size of 1518 Bytes; (ii) random load with frame sizes between 64 and 1518 Bytes; (iii) burst length composed of 10 burst frames with fixed 1518 Bytes generating bursts of 100% peak load. In all the cases, the average load was varied identically from 70% to the maximum without any packet loss. The measured PDVs of PTP messages with respect to the growing traffic load are depicted in the right picture of Figure 4-3. It can be seen that the GST service kept the PDV of PTP packets below 100 ns for all loads of SMT.

Similarly, we also evaluated the performance of SMT with and without the PTP stream. The average round-trip delay with the 30 µs propagation delay of the 6 km dark fibre subtracted for different traffic patterns is shown in Figure 4-4. The impact of the PTP stream increases the SMT delay of burst load traffic by 1.4 µs, while for the constant SMT load the delay impairment is less than 1 µs. Therefore, given a MTU of 1518 Bytes, the maximum latency produced by a single 100G aggregator is 1.72 µs. Figure 4-5 also shows the transfer delay variation of different traffic patterns. The peak value of about 2.7 µs is in line with the theoretical estimation. Identical performances were observed on all disaggregated client ports.

4.3 Flex-E Technology

Flex-E technology is introduced in [7] as a thin layer, known as Flex-Shim, being able to support data rates out of the conventional range offered by current Ethernet standards. The main idea behind Flex-E is to decouple the actual PHY layer speed from the MAC layer speed of a client, i.e. being able to support multiple MAC clients over multiple PHY layers. Flex-E is based on a TDM mechanism that is able to drive the asynchronous Ethernet flows over a synchronous schedule over multiple PHY layers.

Flex-E can run on top of an Optical Transport Network (OTN)-WDM and is able to provide Ethernet services, where multiplexing of users is done in time. Time multiplexing between client groups is performed in a layer between the MAC and the PCS.

Flex-E introduces a Flex-E Shim layer that is responsible for the mapping of Flex-E clients (i.e. Ethernet flows) to groups of PHYs. Flex-E Shim layer is introduced between the Ethernet MAC and the PCS of the PHY layer, as depicted in Figure 4-6. In particular, each aforementioned layer supports:

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

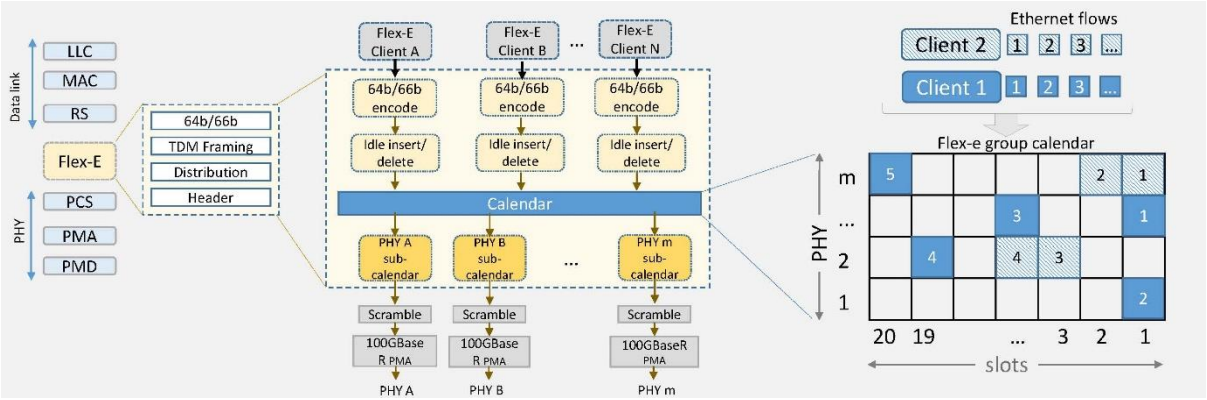


Figure 4-6: Flex-E layer between Ethernet MAC and PCS. Additional FlexE Shim distribute/aggregate sub-layer in PCS/PMD.

Each Flex-E client has its own separate MAC, RS above Flex-E shim which operate at the Flex-E client rate. The layers below the PCS are used intact as specified for Ethernet. As a first step in every Flex-E client flow, a 64b/66b encoding is performed to facilitate synchronisation procedures and allow a clock recovery and alignment of the data stream at the receiver. Then a procedure of idle insert/delete is performed. This step is necessary for all Flex-E clients in order to be rate-adapted, matching the clock of the Flex-E group.

The rate adaptation is accomplished by idle insertion/deletion process, according to IEEE 802.3. This rate is slightly less than the rate of the Flex-E client in order to allow alignment markers on the PHYs of the Flex-E group and insertion of the Flex-E overhead in the steam. Then all the 66b blocks from each Flex-E client are distributed sequentially into the Flex-E group calendar where the multiplexing is performed.

A detailed technical analysis of Flex-E technology can be found in [7] where Flex-E was introduced by OIF. An introduction on Flex-E and possible Flex-E use cases are described by Google in [8]. In [9] the authors present an integration approach of control and management of Flex Ethernet over OTN. Regarding Flex-E implementation landscape Huawei incorporates Flex-E in PTN990 router series.

Ixia presented a demo at OFC 2016 (Altera/Intel booth 2667 March 22-24), with FlexE 2x 100GbE and Ciena provides the Flex-E Liquid Spectrum solution. Regarding the Flex-E standardization activities OIF is providing the implementation agreements for the Flex-E data plane and IETF for the Flex-E control plane. On January 2015, the project was setup in OIF; while IA1.0 was published on March 2016. In [10] IETF presents the primitives for a GMPLS solution on the control plane while potential impact is expected based on the work in

the Common Control and Measurement Plane (CCAMP) Working Group. The latter deals with understanding the Flex-E technology and the relevant use cases, framework and extensions to RSVP-TE to establish signaled Flex-E channels, PCE operations, but also how to support for Deterministic Ethernet (DetNet) and routing algorithms (OSFP, IS-IS). Flex-E can be viewed as a generalization of the Multi-Link Gearbox (OIF MLG 1.0 and OIF-MLG-02.0). The MLG interface multiplexes ten 10GBase-R PCS channels into a single 100 Gb/s link, compatible with the IEEE 802.3ba.

4.3.1 Flex-E Testbed Implementation Description of HW –SW used

The logical testbed design is depicted in Figure 4-7. Two Flex-E enabled routers are connected to support the traffic originating from one server side to another. A hardware traffic generator is used to generate backup traffic, trying to dominate the link. Flex-E is used to perform channelisation, allocating a specific channel and protecting the end-to-end flow.

The actual implemented Flex-E testbed is depicted in Figure 4-8. The switch fabric is used to interconnect the server systems with the routers and also interconnect the server and the routers to the management networks for admin purposes. In the following we provide details for the server systems and the routers which are the main components used to carry the Flex-E demonstrator.

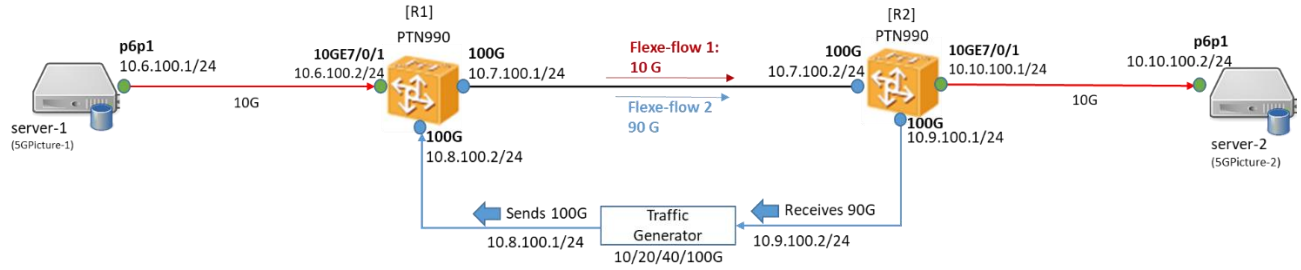


Figure 4-7: Flex-E testbed logical view.

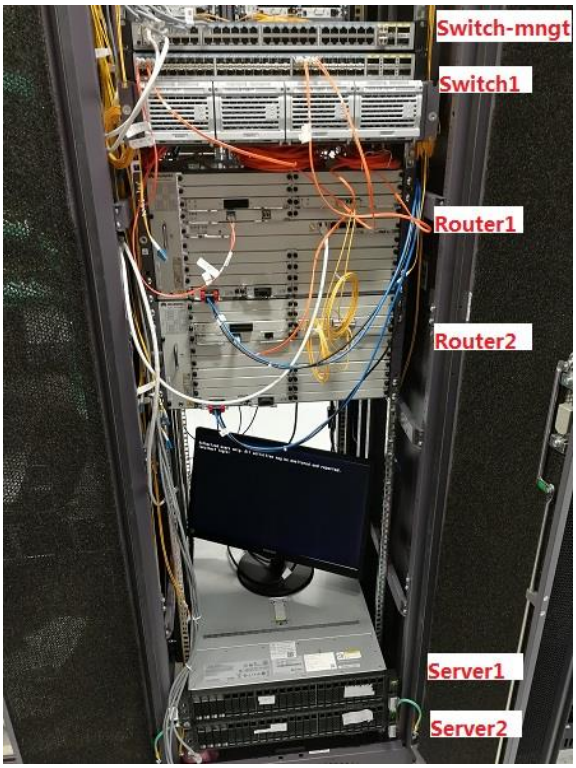


Figure 4-8: Flex-E testbed.

4.3.2 Server Systems

We are using Huawei RH2288H V3 server systems, whose specifications are presented in Table 4-1.

Table 4-1: Server HW specification.

Form factor	2U rack server
Number of processors	Processor model Intel® Xeon® E5-2600 v3 series processor
Number of memory slots	24 slots for DDR4 RDIMMs or LRDIMMs
Maximum local storage	Supports the following hard disk configurations: • Eight front 2.5-inch SSDs or SAS or SATA HDDs • Ten front 3.5-inch SATA HDDs • Twelve front 3.5-inch SAS or SATA HDDs and two rear hard disk modules. Each module provides two 2.5-inch SSDs or SAS or SATA HDDs or 3.5-inch SAS or SATA HDDs. • Twenty-five front 2.5-inch SSDs or SAS or SATA HDDs and two or three rear 2.5-inch SSDs or SAS or SATA HDDs Supports built-in Flash: • Two Mini SSDs (SATA DOM) • Two SD card
RAID	Supports RAID 0, 1, 10, 5, 50, 6, and 60 Uses a supercapacitor to protect RAID cache data from power failures Supports RAID state migration, configuration memory, self-diagnosis, and web-based remote configuration Network ports
Supports the following network configurations:	• Two GE electrical ports, supporting Network Controller Sideband Interface (NC-SI), WOL, and PXE • Four GE electrical ports, supporting NC-SI, WOL, and PXE • Two 10GE optical ports, supporting NC-SI and PXE • Two 10GE electrical ports, supporting NC-SI, WOL, and PXE
PCIe expansion	Supports a maximum of 9 PCIe slots
Fan	Hot-swappable fan modules in N+1 redundancy
Power supply	2 hot-swappable power supplies in 1+1 redundancy
Management	The on-board iBMC module supports Intelligent Platform Management Interface (IPMI), SOL, KVM over IP, and virtual media, and provides a 1 Gbit/s RJ45 management network port supporting NC-SI.
Operating temperature	5°C to 45°C (41°F to 113°F)

Regarding SW, we are using BIOS Firmware Version:(U47)3.22 and Operating System Ubuntu Linux Kernel 4.4.0-128. The server series is a commercial product and more information on the server can be found in [11].

4.3.3 Flex-E Routers

The Flex-E solution will be demonstrated using PTN990 series of Huawei Optical Routers. PTN devices are primarily used on bearer networks that carry various services, such as mobile communication, enterprise users services. In more detail, the OptiX PTN 990 is primarily used at the access or aggregation layers of a metropolitan transport network. It transports packet services on the network and converges them to an IP/MPLS backbone network (see Figure 4-9).

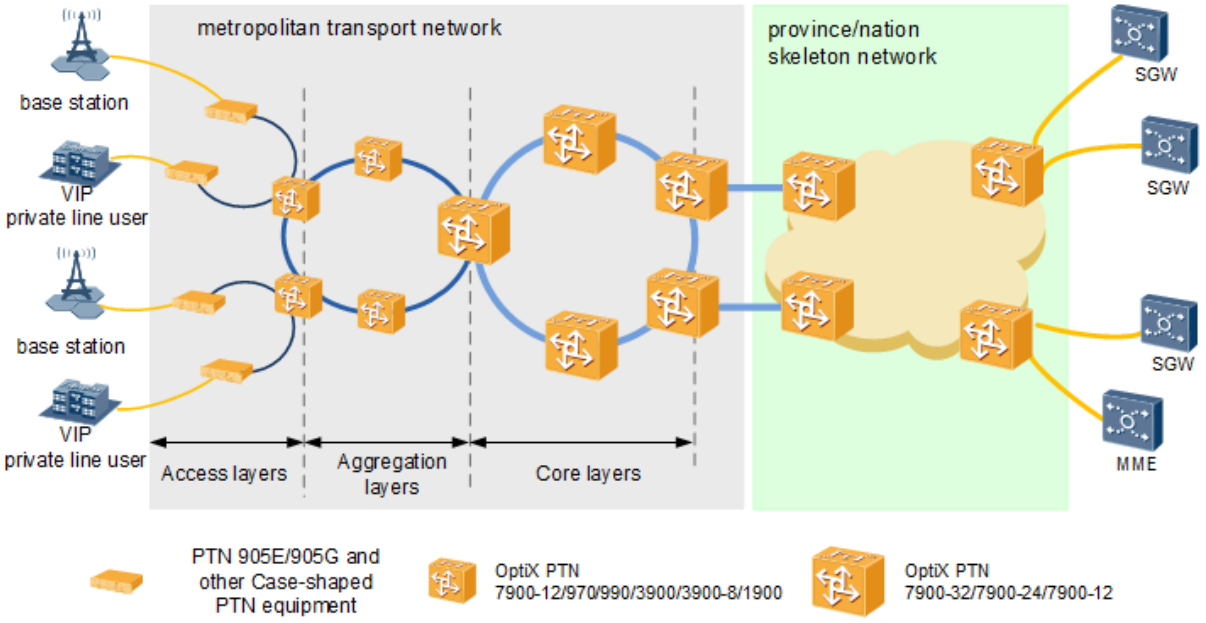


Figure 4-9: OptiX PTN 990 networking on bearer networks and related functions.

4.3.4 PTN990 Router features relevant to 5G-PICTURE

The OptiX PTN 990 provides high processing capability for various types of services and various functions to ensure transport quality and efficiency of various types of services. These span from CES, L2VPN services, SDH support, etc. A number also of HW interface cards are supported More details on the product can be found in [12]. In the following we only provide some details to the HW aspects that are relevant to the demonstrator.

The OptiX PTN 990 provides large-capacity packet switching and processing and multi-granularity service access. It fully meets the processing and access capability requirements on metro transport networks' access and aggregation layers (see Table 4-2).

4.3.4.1 10G Interfaces

For the 10G interfaces we are using J1EX2S boards. The TPJ1EX2S mainly consists of the interface conversion module, control driving module, clock module, and power supply module. Figure 4-10 shows the block diagram for the functions of the TPJ1EX2S.

Table 4-2: Switching capability of the OptiX PTN 990.

Switching Board	Switching Capability	Line Rate I/O Capability
TPJ2CXP	320 Gb/s	320 Gb/s

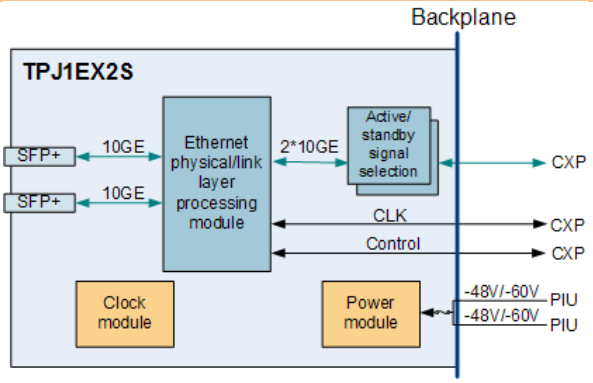


Figure 4-10: Block diagram for the functions of the TPJ1EX2S.

4.3.4.2 100G interfaces

For the 100G interfaces we are using TPJ1EH1 boards. A TPJ1EH1 is a 1 x 100GE Ethernet interface board. This section describes the front panel, functions, and principles of a TPJ1EH1. A TPJ1EH1 consists of the interface conversion module, control driver module, clock module, and power supply module. Figure 4-11 shows the block diagram for functions of a TPJ1EH1.

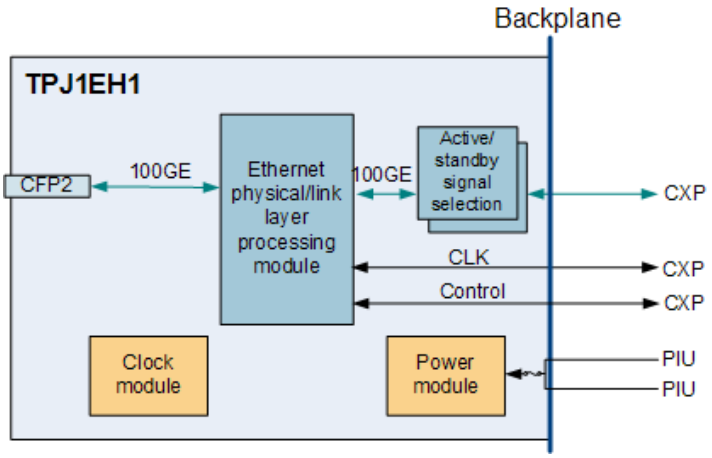


Figure 4-11: Block diagram for functions of a TPJ1EH1.

4.3.4.3 Receive Direction

After a TPJ1EH1 receives 100GE services through CFP2 interfaces, the Ethernet physical/link layer processing module completes service data convergence and scheduling, processes synchronous Ethernet clock signals and 1588v2 packets, and transmits the processed data to the active and standby system control boards.

4.3.4.4 Transmit Direction

A TPJ1EH1 selects service data from the active system control board, the Ethernet physical/link layer processing module processes clock signals and IEEE 1588v2 packets, identifies the destination port of the service data, schedules and distributes the service data, and sends it through a CFP2 interface on the panel.

4.3.4.5 Ethernet Physical/Link Layer Processing Module

The Ethernet physical/link-layer processing module performs the following functions:

- Schedules and distributes service data.
- Supports synchronous Ethernet and 1588v2.
- Exchanges service data with the active and standby system control boards using the dual-fed and selective receiving function.
- Works with the system control boards to manage and control each module on the board.

4.3.4.6 Software

By means of software the VRP 8.130 is used V100R008C10, with a special patch that enables the Flex-E functionality. Note that Flex-E only runs under diagnose mode and still is not available for commercial use.

4.4 X-Ethernet Platform Description

Traditional Ethernet is fixed rate, like 100M, 1GE, 10GE, 40GE and 100GE. Currently, the industry is working on the Flexible-Ethernet, described in the previous section, which has three major functions of bonding, channelization and sub-rating.

Flexible-Ethernet is a transformation of Ethernet where only PCS is involved, while the other layers are still following the standard of IEEE802.3 [13]. The 64/66 bits block is processed with a TDM reconstruction and considered as the payload of Flexible-Ethernet frame. The Flexible-Ethernet overhead is introduced per 20,460 data blocks. Nevertheless, Flexible-Ethernet is just an interface technology, other key component of network technology, such as switch technology, is not included in a foreseeable future. Based on Flexible-Ethernet, in the following we describe the platform exploited for a new network technology Huawei proprietary, named X-Ethernet, where X stands for extended distance, expanded granularity and extremely low latency.

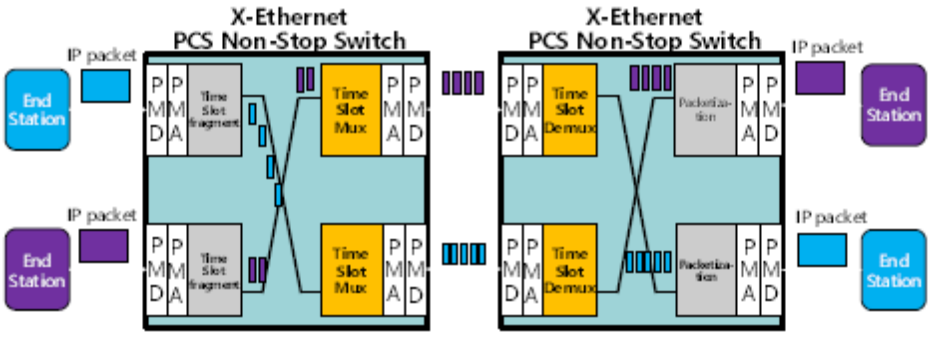


Figure 4-12: X-Ethernet PCS NSS switching mechanism.

X-Ethernet introduces Ethernet switching based on the interface offered by Flexible-Ethernet. The switch device will redirect FlexE Clients (64B/66B block streams) from its inbound port to its outbound port without waiting for the arrival of the whole Ethernet frame for FCS checksum and forwarding decision with table lookup. Therefore, all the time consuming procedures, such as encapsulation/decapsulation, queuing and table lookup, can be removed. We give another name to the PCS switching that is PCS Non-Stop Switch (NSS). What is more, the remaining procedure processing time is predictable, which results in deterministic device latency. Idle insertion or deletion according to IEEE 802.3 may be performed to rate-adapt FlexE Client to the Flex Group. A schematic view of PCS NSS is depicted in Figure 4-12.

At the transmitter side, the IP packet stream is injected into the X-Ethernet inside device. After the standard physical layer processing procedure, the IP stream is transferred to FlexE Clients, which can be regarded as time slot fragments since the Flexible-Ethernet reshapes the 64B/66B bit blocks in a TDM manner. Then, a time slot mux is applied to redirect the time slot fragments to the correct outbound port according to the configuration. In addition, the configuration of time slot mux can be done out-of-band or in-band depending on how we define the Flexible-Ethernet frame. Details of the X-Ethernet technology can be found in [14].

4.4.1 Description of HW – SW used

To demonstrate the capability of X-Ethernet as the unified data plane technology, a prototype is developed based on FPGA. As shown in Figure 4-13, the X-Ethernet prototype is a pizza box like device with the size of 46x44.5x6.4 cm. It has six 100G CFP2 optical module slots, two 10G SFP+ optical module slots, one Ethernet interface slot and one RS232 interface. The RS232 is for device management and control.

The FPGA board (see Figure 4-14 for the overall design) is mainly composing one Virtex UltraScale chip, one ZYNQ chip, two DDR3 SDRAM chips, one DDR4 SODIMM, two Quad-SPI flashes and one Micro SD. The system clock is generated by 25 MHz TCXO, and there are two Si5345A chips responsible for clock management. A temperature sensor is used to prevent the device to become over heated. The ZYNQ can be configured in three approaches, Quad-SPI flash, JTAG and SD memory card. The UltraScale can be configured in two approaches, JTAG and 8 bit Slave SelectMAP.

The X-Ethernet prototype software mainly comprises application software on Windows side and application software on Petalinux side. The Windows side application software has three major functions:

- Firstly, it sets up the X-Ethernet prototype configuration information via RS232 port.
- Secondly, it acquires X-Ethernet prototype status via RS232 port.
- Thirdly, it downloads FPGA ROM to X-Ethernet prototype via RS232 port.

Figure 4-13: X-Ethernet prototype exterior.

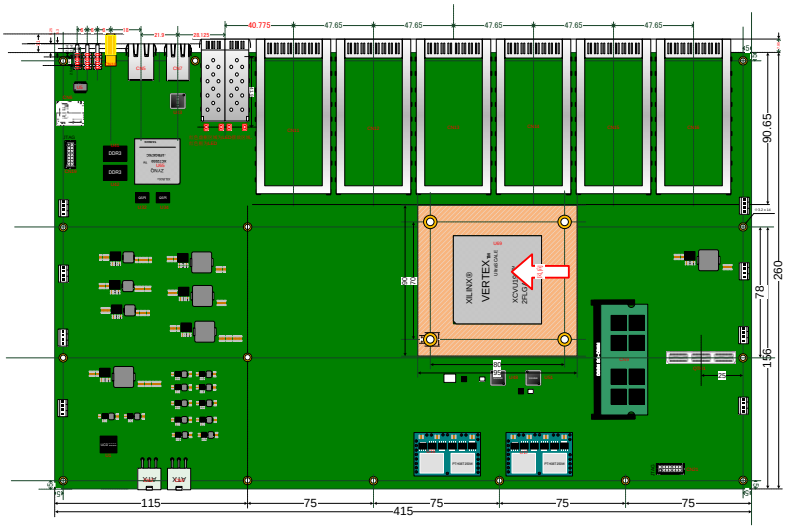


Figure 4-14: X-Ethernet FPGA board design.

The Petalinux side application software also has three major functions. Firstly, it receives the windows side application software command via RS232 port. Secondly, it sends prototype status to Windows side application software via RS232 port. Thirdly, it receives and saves FPGA configuration after it receives configuration commands from Windows side application software.

The windows side application software GUI is shown in Figure 4-15. It has the function of network connection, PLL clock configuration, 100G interface transmission rate monitoring, prototype temperature monitoring, Debug status monitoring, and FPGA Config.

Petalinux side application software requires install TeraTerm on PC first. Then we login the Petalinux through TeraTerm, the interface after it entering system is shown in Figure 4-16.

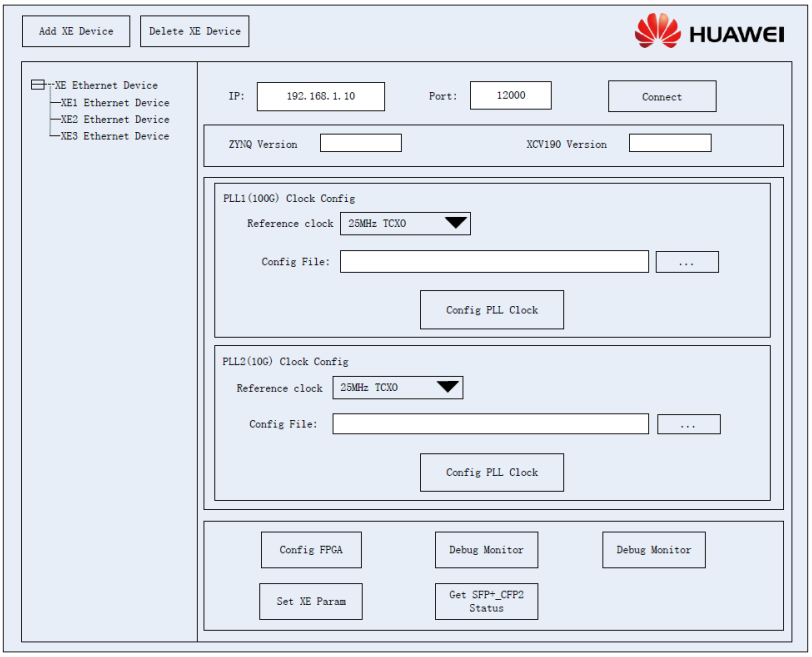


Figure 4-15: Windows side application software GUI.

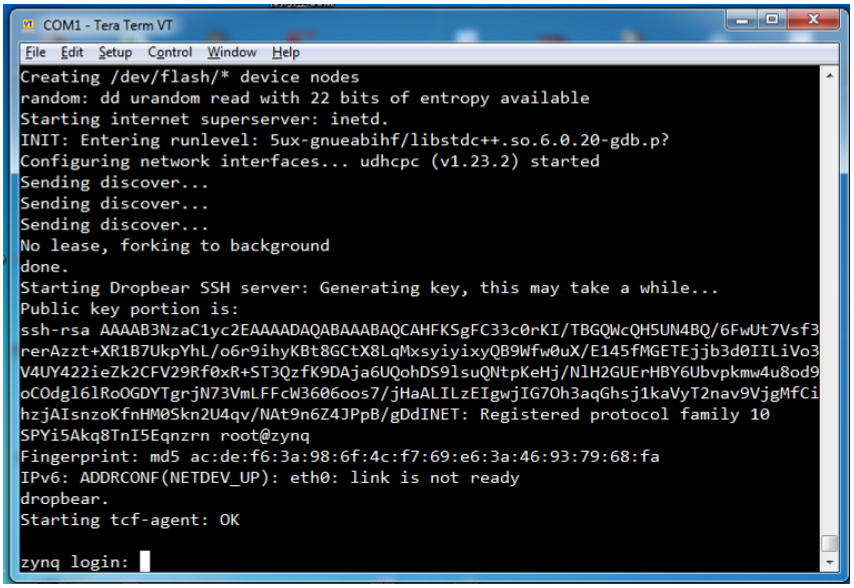


Figure 4-16: Console interface.

4.5 RF processing/modelling

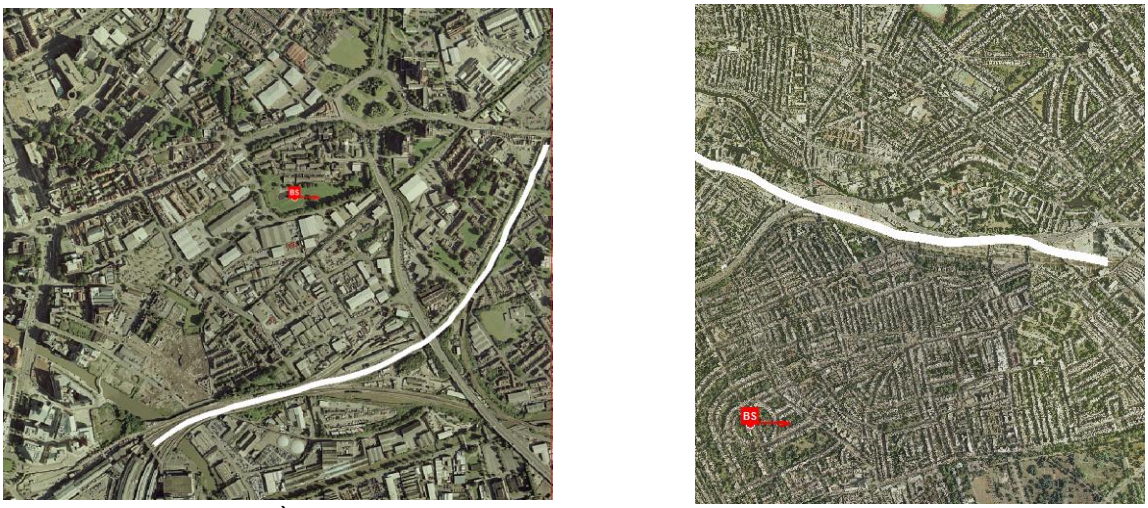
One of the vertical use cases that are being investigated in the context of 5G-PICTURE is the rail use case, i.e. providing 5G communication to trains. In previous deliverables (D2.2, D3.1) we have presented the required tools to evaluate the performance of two technological solutions in a rail environment, along with the implementation plan and initial performance results. In the following sections we present and discuss further evaluation performance of these two solutions: a) Sub-6 GHz, and b) mmWave Access Points (APs).

Our initial results were based on evaluating the throughput performance in two rail scenarios (Bristol Temple Meads station and London Paddington station), considering a 1.5 km track and point-to-point links. In this deliverable, our performance evaluation is based on an actual train consideration (similar to the one that will be used for the final rail demo) moving along the abovementioned tracks. Thus, for our simulations we consider a 240 metres long train, with 8 vehicles (30 m long each).

The two rail scenarios investigated in D3.1 and D3.2 for mmWave and sub-6GHz SISO and MIMO, will be evaluated (performance-wise) during the rest of the project, considering a Massive MIMO Base station and several antennas on the train. Depending on resources, there might also be a performance evaluation on Massive MIMO employment in a stadium scenario.

4.5.1 Sub-6 GHz LTE Massive MIMO coverage cell

Two rail environments, were investigated: a) Bristol Temple Meads, and b) London Paddington, at 2.6 GHz (see the scenarios depicted in Figure 4-17). Initially, for deliverable D2.2, we considered a Single-Input-Single-Output (SISO) system.



a) b)

Figure 4-17: LTE: Bristol Temple Meads scenario (a), London Paddington scenario (b).

In this deliverable we investigate a 2x2 Multiple-Input-Multiple-Output (MIMO) system at 2.6 GHz. We consider both Space Time Block Codes (STBC) and Minimum Mean Square Error (MMSE) in our simulations.

The resulting throughput and SNR for point-to-point links along the rail track is depicted in Figure 4-18 for Temple Meads and Figure 4-19 for the London Paddington station. As expected, with MMSE equalisation, almost double the throughput can be achieved compared to STBC. Moreover, although the achieved throughput in Temple Meads is very good, for the London Paddington route, low throughput performance can be observed, which should be due to the environment and the placement of the BS.

In D2.3, we will present results by considering an actual train (similar to the one that will be used at the final rail demo of the project), deploying one antenna per vehicle in the SISO case, and two antennas per vehicle in the MIMO case. The BS is equipped with one antenna in the former, and two antennas in the latter case.

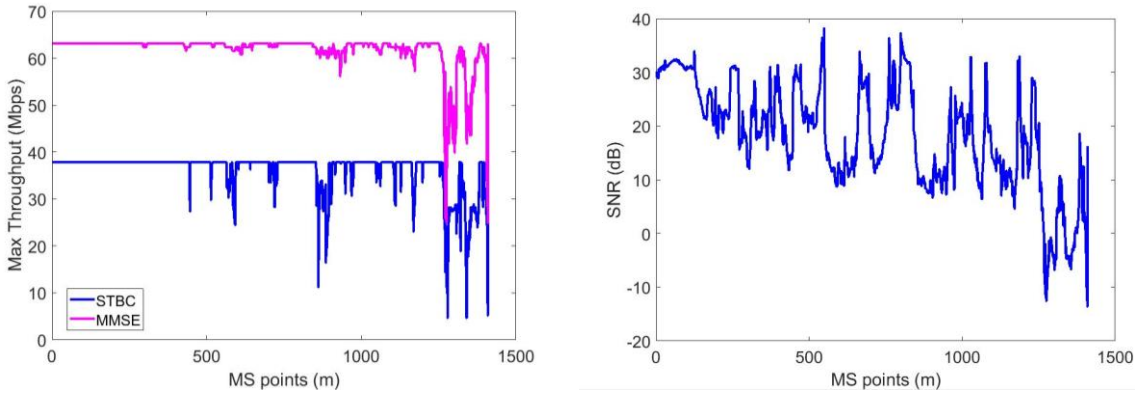


Figure 4-18: LTE: P2P links, 2.6 GHz – Temple Meads: Max. Throughput (left), SNR (right).

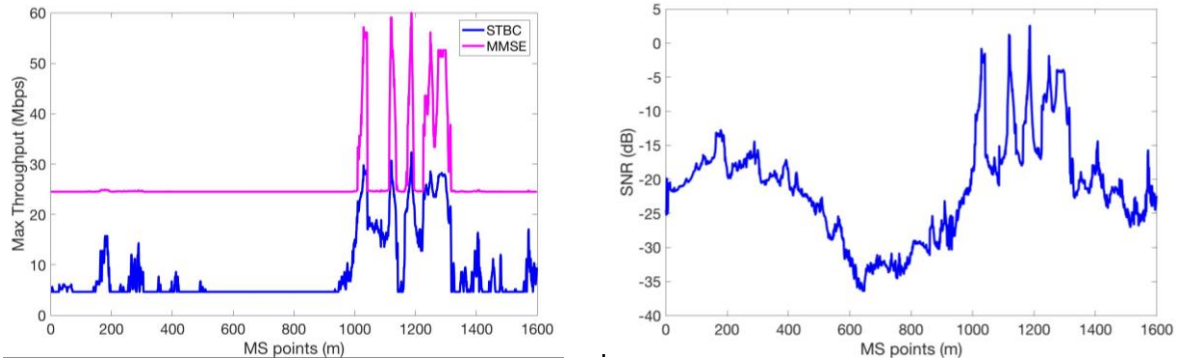


Figure 4-19: LTE: P2P links, 2.6 GHz – London Paddington: Max. Throughput (left), SNR (right).

4.5.2 mmWave Access Points (APs) along the trackside

Two rail environments/scenarios, depicted in Figure 4-20, are investigated: a) Bristol Temple Meads (1410 m long), and b) London Paddington (1599 metres long). In both scenarios, 8 APs (200 m distance between them) are placed along the track. The main analysis and simulation parameters are listed in Table 4-3; some of the parameters are varied in our results to check performance impact.

Our initial results for point-to-point links were based on Tx beamwidth of 10° and Rx beamwidth of 120°. In order to adjust our simulation, in the best possible way, to the actual final rail demo, we change beamwidths, to 7.5°, which follow the configuration of the BWT Typhoon modems. In Figure 4-21, Figure 4-22, Figure 4-23, Figure 4-24 the point-to-point links performance (Temple Meads-APs placed 400 metres apart), at 60 and 26 GHz, is depicted, considering beamforming (BF), maximum power ray selection, and no BF. Furthermore, we also compare the results to an AP beamwidth of 10°. As shown there is not significant difference in the results between 7.5° and 10°, and also between the two center frequencies.

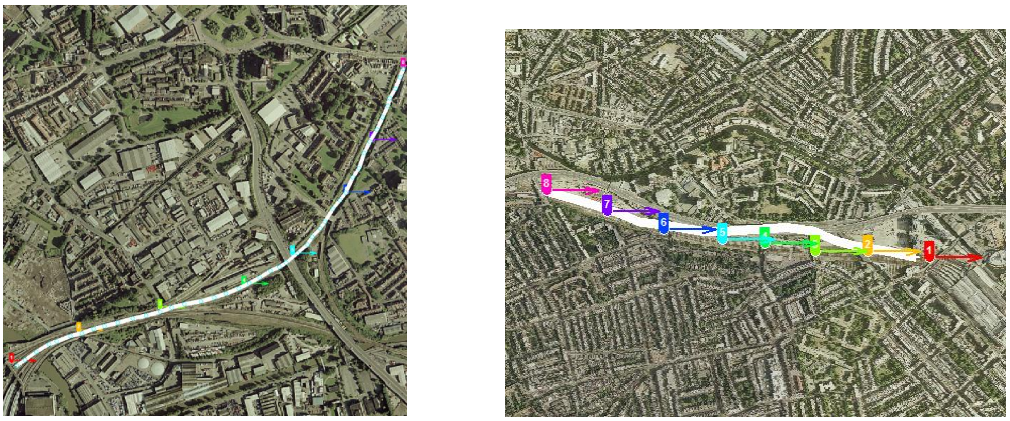


Figure 4-20: mmWave: Bristol Temple Meads scenario (left), London Paddington scenario (right).

Table 4-3: mmWave: Modelling and analysis parameters

Parameter	Value
Carrier Frequency	26 GHz & 60 GHz
Tx power (ray-tracing)	0 dBm
Tx power (simulator)	22 dBm
Tx antenna gain	0 dBi
Rx antenna gain	30 dBi
Bandwidth	2.16 GHz
Implementation loss	5 dB
Noise floor	10 dBm

AP (Tx) height	3 m
MS (Rx) height	2.5 m
Ray tracing resolution	1 m
Route length	1.41 km & 1.59 km
Number of APs	8
Length of train	240 m
Width of train	4 m
Tx beamwidth (degrees)	7.5°
Rx beamwidth (degrees)	7.5°

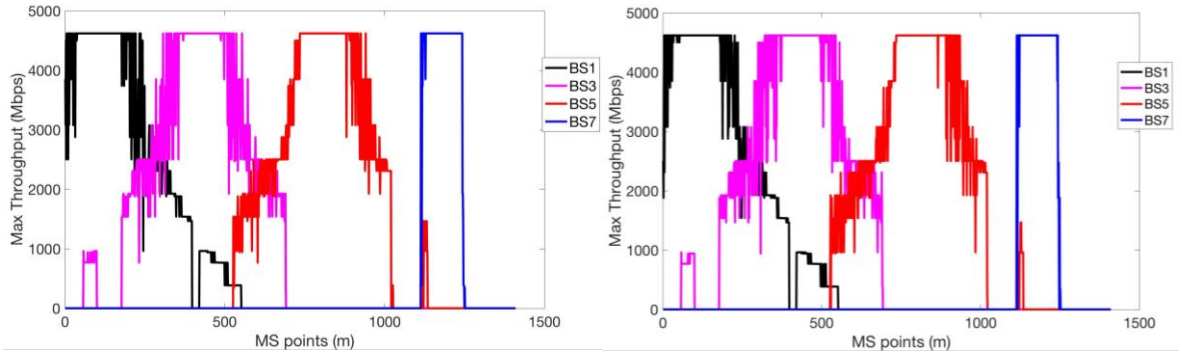


Figure 4-21: Temple Meads - Throughput at 60 GHz without BF: (left) 7.5° bwidth (right) 10° bwidth.

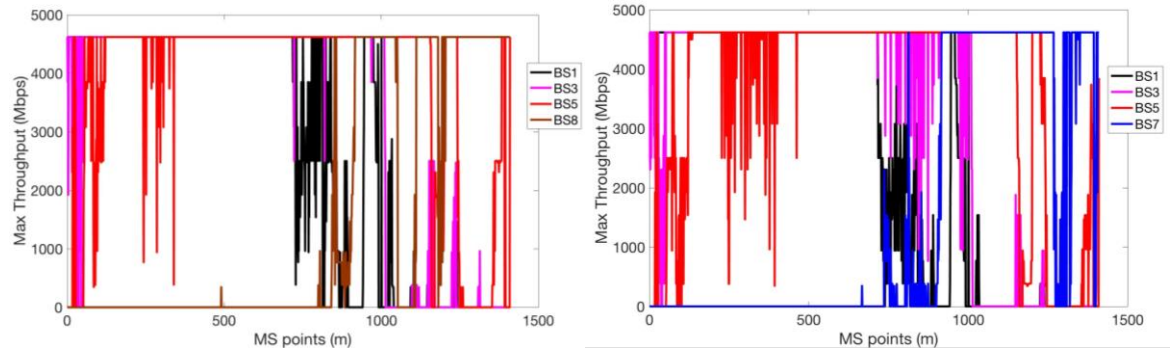


Figure 4-22: Temple Meads - Throughput at 60GHz with BF: (left) 7.5° bwidth (right) 10° bwidth.

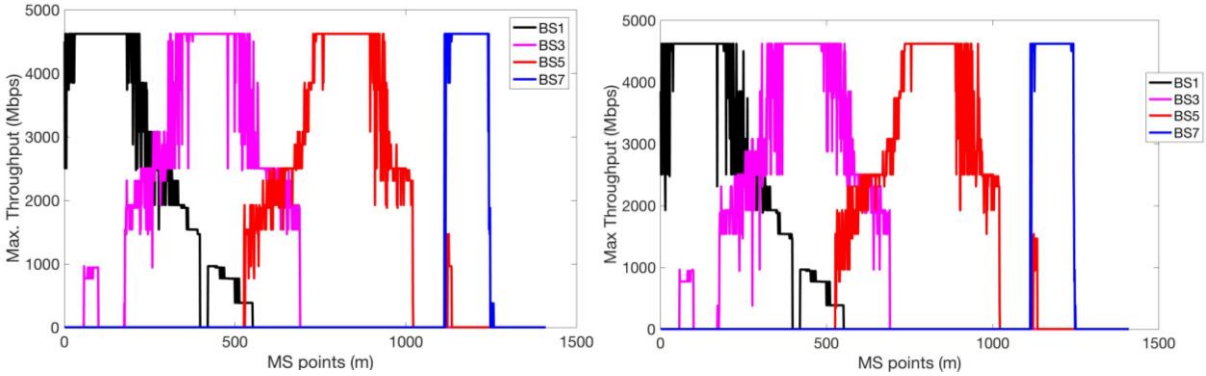


Figure 4-23: Temple Meads - Throughput at 26GHz without BF: (left) 7.5° bwidth (right) 10° bwidth.

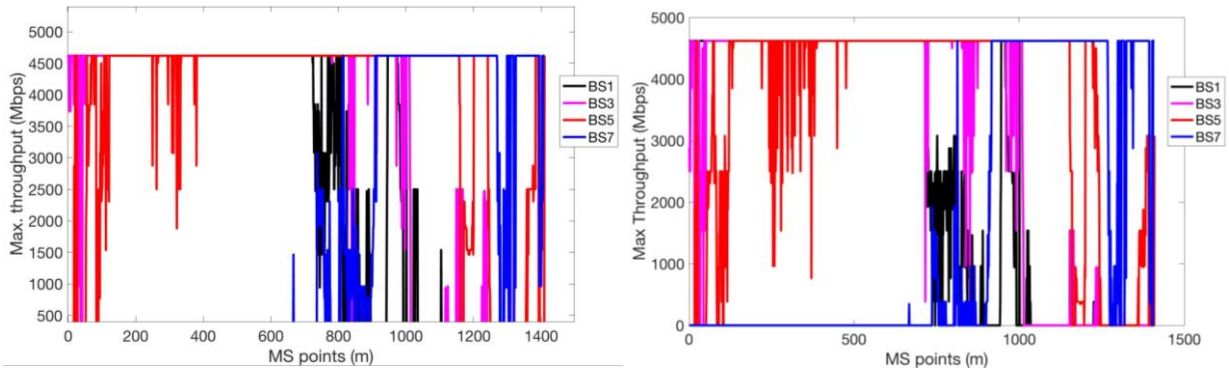


Figure 4-24: Temple Meads - Throughput at 26GHz with BF: (left) 7.5° bwidth (right) 10° bwidth.

For the two scenarios, at mmWaves we consider the train depicted in Figure 4-25, with one mmWave AP placed at the front (A) and one at the rear of the train (B). We start our simulation when the rear of the train is at point 1 (beginning of track) and the front of the train is at point 241, and the simulation is completed when the front of the train is at the last point of the track and the rear of the train at point 1170 and simulation is completed when the front of the train is at the last point of the track.

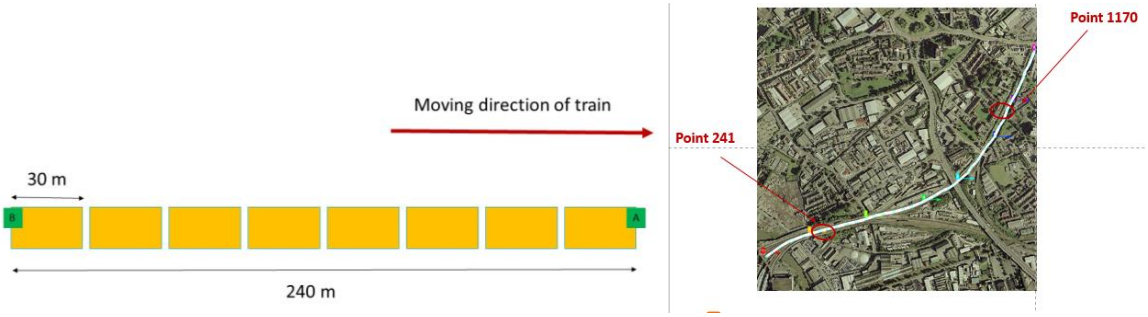


Figure 4-25. (left) Train consideration in mmWave, (right) example of simulation in Temple Meads.

Considering the train moving along the track, we investigated the throughput performance, for both Temple Meads and Paddington, with and without beamforming (max ray selection), with 7.5° beamwidth at both ends, and for APs distance 400/600/800 m, at 26/60 GHz. Results will be thoroughly presented and discussed in deliverable D2.3.

4.6 BB Processing in Active Antenna Distributed Unit (AADU)

4.6.1 Architecture Summary

In the previous Deliverable D3.1 [1], Sec. 5.5.4, architecture options were discussed to implement an active antenna distributed unit (AADU) including partial PHY layer processing. For an AADU, Split 7.2 is the most promising, as it reduces the required FH capacity significantly. Figure 4-26 shows the corresponding processing, illustrating the processing steps to be performed locally in the AADU. In the following we will investigate first the computational complexity and the configurability of such an AADU.

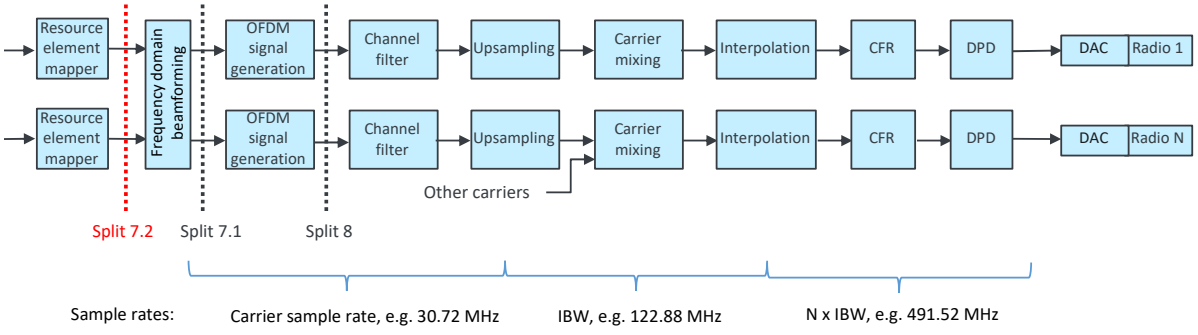


Figure 4-26: AADU processing chain and functional split.

4.6.2 Functional Split: Computational Complexity

An open question for future RAN architecture is the tradeoff between centralised and distributed/disaggregated processing: while including more processing in the AADU, the required FH capacity decreased, which is desirable to lower network deployment cost. At the same time however, more processing power is required remotely. One of the benefits of a centralised RAN processing is that it is assumed, that general purpose processors (GPPs) can be utilized, benefiting form economy of scale and pooling gains. For an AADU, an interesting question is, whether it, too, can utilize GPPs for lower PHY processing, to benefit similarly. Alternatively, the AADU has to utilise FPGAs or ASICs, which are in general less flexible, at the benefit of a higher efficiency.

First, we need to revisit the lower PHY processing chain as presented in deliverable D3.1. It included only basic 3GPP processing, while not considering more practical aspects. Figure 4-26 now shows the full processing chain. The necessary additions are mainly due to two practical considerations: first, AADUs have to support several carriers at the same time. For this, digital channel filtering, up-conversion and carrier mixing is required to create a composite signal of several carriers. Second, in order to provide sufficient transmit power, AADUs have to use non-linear power amplifiers. In order to avoid distortion by their non-linearity, digital predistortion (DPD) needs to be applied, which in turn requires a crest-factor reduction (CFR) first. These processing steps increase the required computational complexity significantly, as analysed in the following.

The computational complexity of a certain function in general depends very much on the specific implementation. However, the order of magnitude can be estimated for comparison based on the basic operations to be performed. As an example, consider a beamformer, which converts $N_L = 18$ data streams (layers/ beams) into $N_A = 64$ antenna streams. For a 20 MHz LTE carrier with $N_{SC} = 1200$ subcarrier and $N_{symb} = 14$ symbol per $T_{subframe} = 1$ ms, the output has to be produced at a rate of

$$R_{BF,L} = N_A \cdot N_{SC} \cdot N_{symb} = 1075.5 \text{ MSamples/s}$$

provide some headroom, let's assume that a slightly higher rate is used, e.g. $R'_{BF,L} = 1250$ Msamples/s. One antenna sample requires $N_{CMAC,L} = 18$ complex multiplications and additions. Multiplications and additions can usually be performed within one clock cycle. One complex multiplication requires $N_{OP,CMAC} = 3$ non-complex operations (flop).

Finally, the total rate of operations is:

$$R_{BF,tot} = R'_{BF,L} \cdot N_{CMAC,L} \cdot N_{OP,CMAC} = 67.5$$

Given as 1250 Msamples/s x 18 multiply/add x 3 operations = 67.5 Gflops.

Using the same methodology, the computational complexity of all operations was estimated. The results are given in Figure 4-27 for $N_L = 18$ layers, $N_A = 64$ antennas, and $N_{CA} = 3$ carriers within a bandwidth of 20 MHz each (OBW = 3x20 MHz) within 100 MHz band (IBW = 100 MHz). It is also indicated, with which parameter the numbers are scaling.

Next, the required computational power has to be put into relation with the one available from different platform. As an example, for FPGA processing, we are using a Xilinx Ultrascale+ [14]. It features $N_{DSP} = 4272$ DSP48(s (multipliers) and can run approximately ant up to $C_{FPGA} = 500$ MHz. However, usually an FPGA cannot utilise all resources at maximum clock speed, so assuming a maximum utilization of $\mu_{FPGA} = 0.7$, the total computational power is estimated as

$$R_{FPGA} = N_{DSP} \cdot C_{FPGA} \cdot \mu = 1495 \text{ Gops}$$

Power consumption of FPGAs is not straight forward, as it usually depends on many factors like the overall utilisation. From [ref], we estimate a typical FPGA power consumption as $P_{FPGA} = 30$ W.

For GPPs, we use a Xeon 6140 as reference [15]. It features up to $N_{x86,cores} = 18$ cores, each core can perform $N_{x86,FLOP} = 32$ FLOPs per cycle and it runs at a maximum clock speed of $C_{x86} = 2.6$ GHz. Since a certain overhead has to be assumed for program control and the operating system, we estimate that the maximum processor utilisation for signal processing is $\mu_{x86} = 0.7$. this yields a total computational power of

$$R_{x86} = N_{x86,cores} \cdot C_{x86} \cdot N_{x86,FLOP} \cdot \mu = 1048 \text{ Gops}$$

The power consumption of the Xeon 6140 is given as approximately $P_{x86} = 140$ W. With this, we can now estimate the required number of FGPAs for certain or all functions as

$$N_{FPGA/x86} = \lceil R_{function}/R_{FPGA/X86} \rceil$$

Figure 4-28 now compares the number of FPGAs and number of Xeon servers/cores required for lower PHY processing according to Split 7.2.

As can be seen, e.g., 5 FPGAs or 121 Xeon cores would be required for 3x20 MHz carriers. The corresponding power consumption would be 140 W for FPGAs or 934 W for the Xeons. From these, it can be concluded that FPGAs are the only viable option. While an AADU with 5 FPGAs is feasible both from a form factor, power, and thermal perspective, a unit with 121 x86 cores corresponds to 7 powerful data-center-type servers, which is not viable for a rooftop mounted AADU.

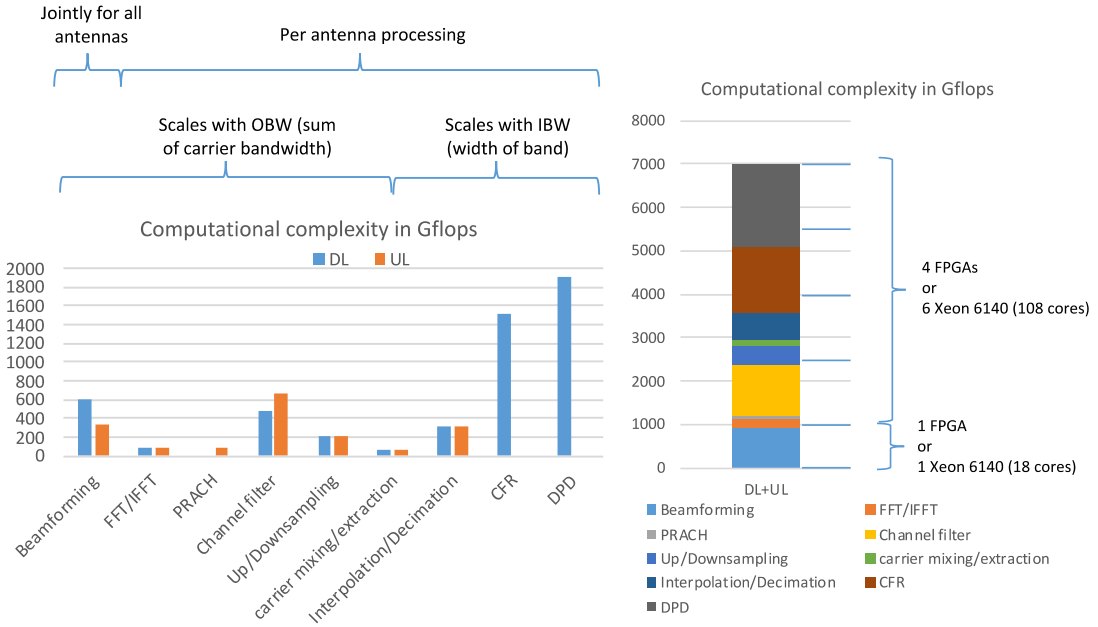


Figure 4-27: Computational complexity of PHY layer functions.

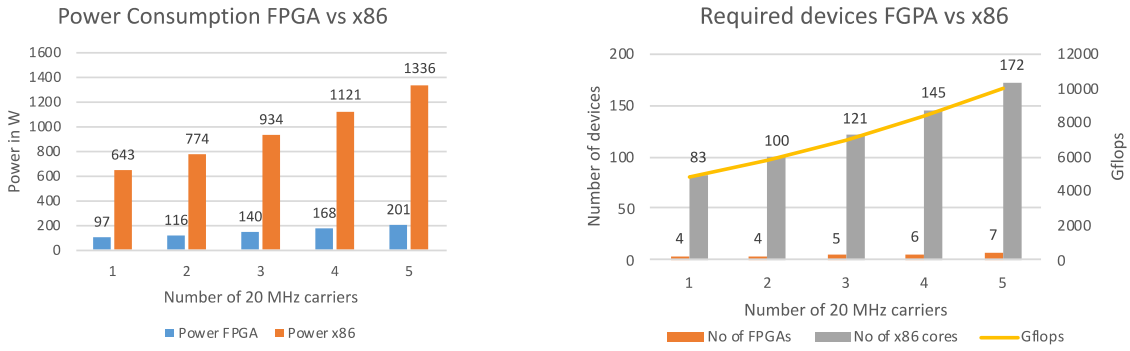


Figure 4-28: Computational complexity: FPGA vs. x86.

4.6.3 AADU Configurability and Control

The AADU mainly includes lower PHY layer processing such as beamforming and FFT/IFFT. As this processing is fixed, the AADU is not a programmable platform. An AADU is rather a part of the 5G-PICTURE physical infrastructure, that instantiates a specific function, i.e. lower PHY processing. However, the AADU can be configured, similar to other infrastructure, according to the 5G-PICTURE network's needs. To give an example, an AADU can support different numbers of carriers. If the 5G-PICTURE OS discovers a need for higher RAN capacity, AADUs can be configured to enable additional carriers to be used on the air interface. Similar, if less capacity is required, carriers can be turned off to save power. At the same time, the AADU can report different status messages for the 5G-PICTURE OS to react to. For example, the AADU could report FH

link errors, which could indicate that more FH capacity needs to be provided. With that, it provides the 5G-PICTURE OS with a view of the current network status.

For configuration and reporting, the 5G-PICTURE AADUs will utilise the standardised Management-Plane (M-Plane) of the xRAN standard [17]. It utilises the NETCONF protocol for communication, and a YANG model as data structure and for remote procedure calls. For this, a NETCONF server will be located in the AADU, and a NETCONF client in the CU.

Examples of possible configurability of the AADU via NETCONF include:

- Rebooting AADU.
- Software update of AADU.
- Adding/removing carriers.
- Increasing/decreasing RF power.
- Setting AADU to idle mode (power save) or active mode.
- Muting/ unmuting RF.
- Modifying beamforming weights for broadcast beams.

At the same time the AADU can report errors, alarms and measurements such as:

- Hardware failures (PAs, RSUs).
- Temperature measurements.
- Fronthaul link status.
- ISU/RSU connectivity status.
- Sync status.

4.7 MIMO at mmWaves

From the literature, it is known that mmWave communications can offer high data rates due to the large bandwidth available. However, they suffer from high propagation losses that can be compensated with high gain antennas or phased array antennas. By combining mmWaves with MIMO techniques, one can achieve very high data rates since the capacity of a system exploiting spatial multiplexing (SM) scales up linearly with the number of antennas.

In deliverable D3.1 we discussed different MIMO architectures at mmWaves. Among the presented ones, the mmWave Line-of-Sight (LoS) MIMO architecture is especially interesting for wireless BH applications, where very high data rates need to be supported. In the following we provide more details this architecture and its feasibility for implementation.

4.7.1 LoS MIMO Theoretical Background

In general, LoS MIMO systems are analysed using the principles of diffractions-limited optics (i.e. image theory) [28]. The number of parallel data streams supported by these systems is determined by two factors, the antenna array arrangement and the wavelength-transmission range product [29]. This means that the spacing between antenna elements is correlated with the achievable link range (i.e. the separation between transmitter and receiver). In other words, when additional streams are needed, either array size has to be increased or, wavelength or range has to be decreased.

4.7.1.1 On the optimal antenna spacing

Figure 4-29 shows different antenna configurations for LoS MIMO, such as uniform linear array (ULA) or uniform rectangular array (URA). In the LoS scenario, assuming parallel ULA with *N* antennas, the optimal antenna spacing at transmitter d_t and receiver d_r satisfies [30]

$$d_t \cdot d_r = \frac{\lambda D}{N}.$$

In the case of symmetric Tx and Rx transceivers, meaning $d_t = d_r = d$, the above equation reads

$$d^2 = \frac{\lambda D}{N}.$$

Similarly, for the parallel URA, the optimal horizontal and vertical antenna spacing at transmitter $d_{t,h}, d_{t,v}$ and receiver $d_{t,h}, d_{t,v}$, are expressed as [30]

$$d_{t,h} \cdot d_{r,h} = \frac{\lambda D}{N_h}, \qquad d_{t,v} \cdot d_{r,v} = \frac{\lambda D}{N_v},$$

where N_h, N_v denote the number of horizontally and vertically spaced antennas, respectively, satisfying $N = N_h \cdot N_v$. In the case of symmetric system, i.e. $d_{t,h} = d_{r,h} = d_h$ and $d_{t,v} = d_{r,v} = d_v$, one obtains

$$d_h \cdot d_h = \frac{\lambda D}{N_h}, \qquad d_v \cdot d_v = \frac{\lambda D}{N_v}.$$

For example, let us assume a 4x4 MIMO system ($N = N_h \cdot N_v = 2 \cdot 2 = 4$) with symmetric transmitter and receiver operating at the carrier frequency of 60 GHz and a transmission distance of 100 metres. Such system has an optimal antenna spacing of $d_h = d_v = 0.5$ metres.

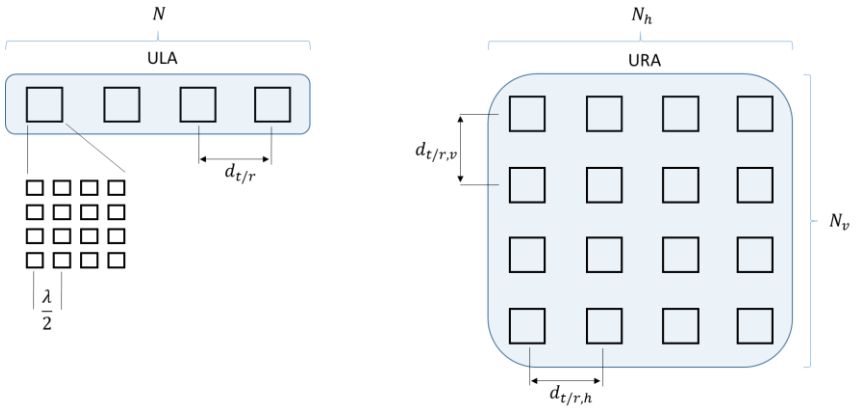


Figure 4-29: Different antenna configurations for mmWave LoS MIMO.

Table 4-4 shows different antenna configurations and optimal antenna spacing for the 60 GHz LoS MIMO symmetric system ($d_{v,t} = d_{v,r} = d_v$ and $d_{h,t} = d_{h,r} = d_h$) at 100 m and 200 m distances.

Table 4-4: mmWave LoS MIMO: optimal antenna spacing.

Index	$N = N_h \times N_v$	D (m)	d_v [m]	d_h [m]	Height × Width
1	4 = 2 × 2	100	0.50	0.50	0.50m × 0.50m
2	6 = 3 × 2	100	0.41	0.50	0.82m × 0.50m
3	8 = 4 × 2	100	0.35	0.50	1.06m × 0.50m
4	9 = 3 × 3	100	0.41	0.41	0.82m × 0.82m
5	10 = 5 × 2	100	0.32	0.50	1.26m × 0.50m
6	16 = 4 × 4	100	0.35	0.35	1.06m × 1.06m
7	4 = 2 × 2	200	0.71	0.71	0.71m × 0.71m
8	6 = 3 × 2	200	0.58	0.71	1.15m × 0.71m
9	8 = 4 × 2	200	0.50	0.71	1.50m × 0.71m
10	9 = 3 × 3	200	0.58	0.58	1.15m × 1.15m
11	10 = 5 × 2	200	0.48	0.71	1.79m × 0.71m
12	16 = 4 × 4	200	0.50	0.50	1.50m × 1.50m

4.7.1.2 Capacity

Let us assume a generic scheme of LoS MIMO system as shown in Figure 4-30. Both transmitter and receiver are equipped with *N* antennas.

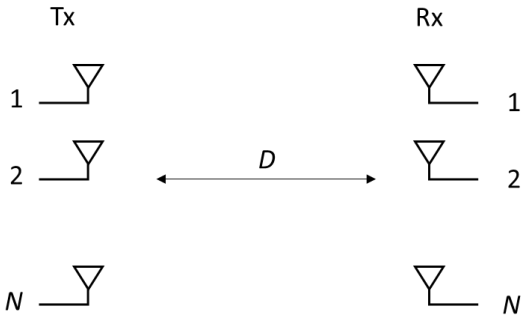


Figure 4-30: LoS MIMO system.

For the MIMO system depicted in Figure 4-30, the equivalent baseband representation is given by

$$\mathbf{y} = \mathbf{H}_{\mathrm{LOS}}\,\mathbf{x} + \mathbf{n},$$

where $\mathbf{x}, \mathbf{y}$ are transmit and receive signal vectors and $\mathbf{n}$ is the additive noise vector whose entries follow complex Gaussian distribution. Further, it is assumed that the channel matrix is determined only by a LoS component which does not vary with time. Then, for a pure LoS channel, which is a fair assumption for a BH link with highly directional antennas, the channel matrix can be represented as [29]

$$\mathbf{H}_{\mathrm{LOS}} = \begin{bmatrix} h_{11} & \cdots & h_{1N} \\ \vdots & \ddots & \vdots \\ h_{N1} & \cdots & h_{NN} \end{bmatrix},$$

where the entries of the channel matrix $\mathbf{H}_{\mathrm{LOS}}$ are given by

$$(\mathbf{H}_{\mathrm{LOS}})_{m,n} \;=\; h_{m,n} = a_{m,n} \exp\left(-j\frac{2\pi}{\lambda} r_{m,n}\right).$$

λ is the wavelength of the carrier, $r_{m,n}$ represents the distance between transmit antenna n and receive antenna m, $m,n = 1,\ldots, N$, and $a_{m,n}$ is the attenuation coefficient which depends on the link distance and the gains of used transmitter and receiver antennas. For large distances, attenuation coefficients should be very similar across different Tx and Rx antenna pairs.

In general, the capacity of the LoS MIMO system is given by [31]

$$C = \log_2(\det(\mathbf{I}_M + \frac{SNR}{N_s} \cdot \widehat{\mathbf{H}}_{\mathrm{LOS}} \widehat{\mathbf{H}}_{\mathrm{LOS}}^{H})),$$

where SNR denotes the average signal-to-noise ratio and N_s is the number of streams supported, which under optimal antenna spacing equals N. $\widehat{\mathbf{H}}$ is the normalized channel matrix with the entries given by

$$(\widehat{\mathbf{H}}_{\mathrm{LOS}})_{m,n} \;= \exp\left(-j\frac{2\pi}{\lambda} r_{m,n}\right),$$

which under optimal antenna spacing satisfies $\widehat{\mathbf{H}}_{\mathrm{LOS}} \widehat{\mathbf{H}}_{\mathrm{LOS}}^{H} = N\mathbf{I}_{\mathrm{N}}$.

The SNR is determined as

$$SNR = \frac{\mathrm{P_T G}_{Tx} \mathrm{G}_{Rx} \mathrm{G}_p}{\mathrm{k T_0 F W}},$$

with P_T being the transmitted power per stream, k = 1.38·10^{-23} J/K being the Boltzmann constant, T_0 = 290 K is the standard absolute temperature, G_{Tx} is the max antenna gain at Tx, G_{Rx} is the max antenna gain at Rx, F is the noise factor of Rx chain and W is the channel bandwidth. G_p represents the propagation losses which

include free-space path-loss $\left(\frac{4\pi D}{\lambda}\right)^2$, atmospheric attenuation (due to oxygen absorption and rainfall) and front-end losses.

The maximum achievable rate, over the bandwidth *W*, is given by

$$R = W \cdot C = W \cdot \log_2(\det(\mathbf{I}_M + \frac{SNR}{N_s} \cdot \widehat{\mathbf{H}}_{\text{LOS}} \widehat{\mathbf{H}}_{\text{LOS}}^H)) \text{ bps.}$$

For the parameters given in Table 4-5, we have estimated the maximum achievable rates for different antenna configurations. The results are given in Table 4-6 and Figure 4-31. These results represent theoretical upper limit on the achievable rate. For practical systems, this rate is somewhat lower and limited by RF impairments. For example, considering IEEE802.11ad standard, with the estimated SNR of 25.95 dB, the highest modulation and coding scheme (MCS12) with the data rate of 4.62 Gb/s could be supported. That means that the aggregated data rates for a system represented with indices 1, 2, 3, 4, 5, and 6 would be 18.48, 27.72, 36.96, 41.58, 46.20, and 73.92 Gb/s, respectively.

Table 4-5: mmWave LoS MIMO: simulation parameters.

Parameter	Value
Frequency [GHz]	60
Transmit power [dBm]	10
Tx antenna gain [dBi]	30
Tx front-end loss [dB]	3
Rx noise figure [dB]	8
Rx antenna gain [dBi]	30
Rx front-end loss [dB]	3
Bandwidth W [GHz]	1.88
Oxygen absorption [dB/100m]	1.5
Rainfall attenuation [dB/100m]	1.8

Table 4-6: mmWave LoS MIMO: results.

Index	$N = N_h \times N_v$	*D* (m)	Height × Width	SNR (dB)	*R* (Gb/s)
1	4 = 2 × 2	100	0.50m × 0.50m	25.95	64.86
2	6 = 3 × 2	100	0.82m × 0.50m	25.95	97.29
3	8 = 4 × 2	100	1.06m × 0.50m	25.95	129.72
4	9 = 3 × 3	100	0.82m × 0.82m	25.95	145.94
5	10 = 5 × 2	100	1.26m × 0.50m	25.95	162.15
6	16 = 4 × 4	100	1.06m × 1.06m	25.95	259.45
7	4 = 2 × 2	200	0.71m × 0.71m	16.63	41.78
8	6 = 3 × 2	200	1.15m × 0.71m	16.63	62.67
9	8 = 4 × 2	200	1.50m × 0.71m	16.63	83.58
10	9 = 3 × 3	200	1.15m × 1.15m	16.63	94.01
11	10 = 5 × 2	200	1.79m × 0.71m	16.63	104.46
12	16 = 4 × 4	200	1.50m × 1.50m	16.63	167.13

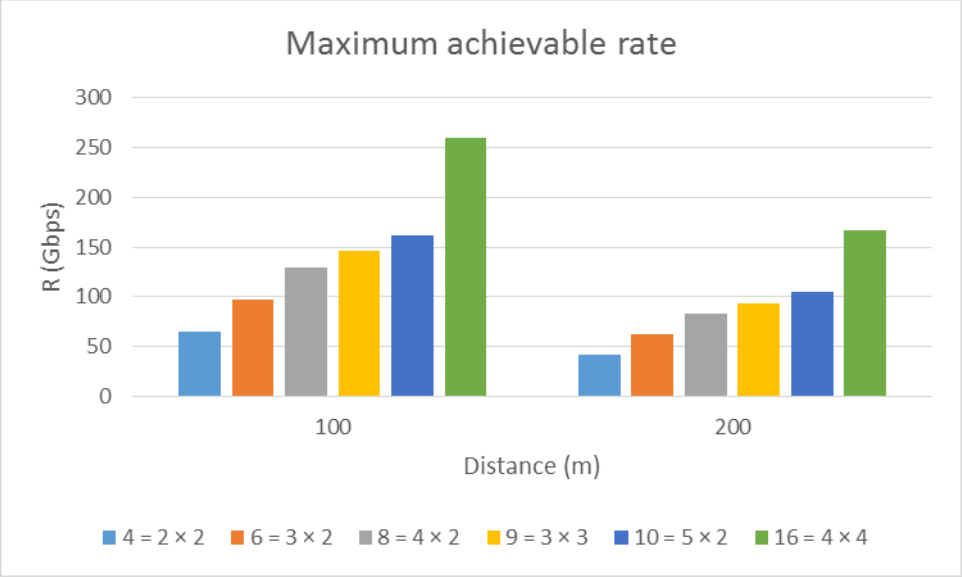


Figure 4-31: Maximum achievable rate of the 60 GHz LoS MIMO system for different antenna configurations at 100 and 200 metres distances.

4.7.2 Experimental evaluation

To assess the feasibility of LoS MIMO at mmWaves, a 60 GHz 2×2 LoS MIMO experiment was performed in an anechoic chamber. For the experimental evaluation we used COTS 60 GHz analogue-front ends (AFE) from Infineon. All AFE modules were equipped with commercial horn antennas of 20 dBi gain and beamwidth of 15 degrees. Distance between transmitter and receiver was D = 5.5 m, for which the free space path loss at the carrier frequency of 60.48 GHz is approximately 83 dB. At the same carrier frequency, the minimum, optimal antenna separation is equal to d = 11.68 cm. However, because of physical constraints, the second best optimal antenna separation, calculated as $d = \sqrt{\lambda D/N \times 3}$ = 20.23 cm, was chosen.

Transmit signal streams were generated with an arbitrary waveform generator (AWG) from Keysight. Transmitted waveforms were two orthogonal repetitive pseudo-noise (PN) training sequences, oversampled 8 times and passed through a root-raised cosine (RRC) pulse shaping filter. The sequences were transmitted at the sampling rate of 10 GSps resulting in a modulated signal bandwidth of 1.25 GHz. The bit rate of uncoded MIMO stream is 1.25 Gb/s and 2.5 Gb/s for BPSK and QPSK, respectively. From the Tx AFE configuration parameters, output power per channel was estimated to -4 dBm. The corresponding EIRP was 16 dBm. Infineon AFEs on both transmit and receive side were using independent local oscillators. At the receiver side single ended I/Q outputs were recorded with the real-time sampling oscilloscope RTO1044 at the sampling rate of 10 GSps. Figure 4-32 show the measurement setup for 2×2 MIMO system.

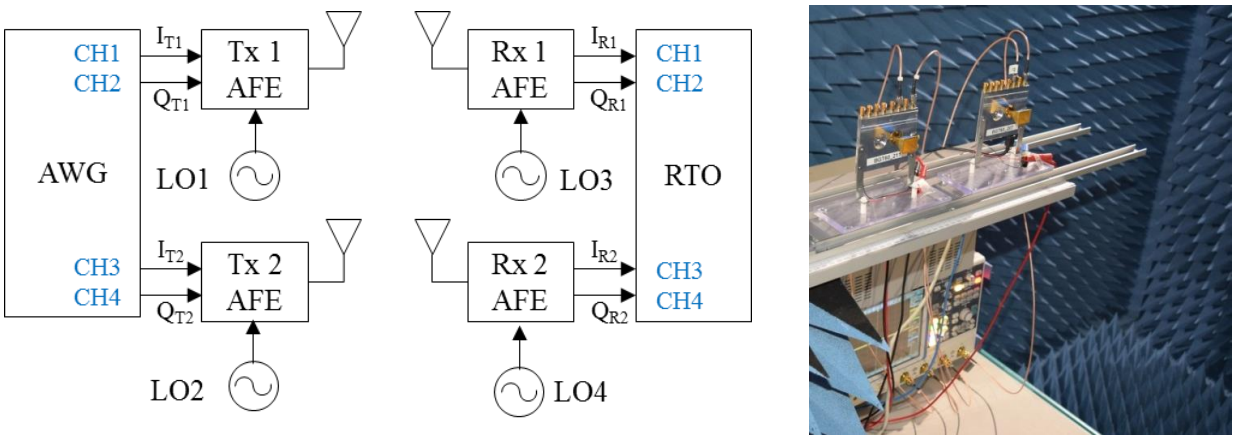


Figure 4-32: (a) Diagram of the 2x2 MIMO setup and (b) receiver side of the 2x2 MIMO setup in the anechoic chamber showing the two AFEs and the RTO.

Post-processing was performed in MATLAB as described below and shown in Figure 4-33. After receive pulse shaping, channel estimation is performed based on the cross-correlation of the received streams with the training sequences. Sampling synchronisation is implicitly performed by selecting the closest-to-optimal point from correlation peaks. There is no explicit sampling frequency offset (SFO) algorithm applied. The received streams are then decimated to the original symbol rate and cropped to a length of 20 training sequences of 127 symbols each, resulting in total of 2540 symbols. Zero-forcing equalisation is applied to each consecutive training sequence based on the pseudo-inverse of the estimated channel matrix using that same sequence (i.e. the estimate is updated for each next sequence for best equalizer results). No explicit carrier frequency offset (CFO) correction is therefore needed as well, since the consecutive channel estimates contain the phase information. Also, no specific I/Q impairment compensation or frequency selectivity compensation is performed for the given results.

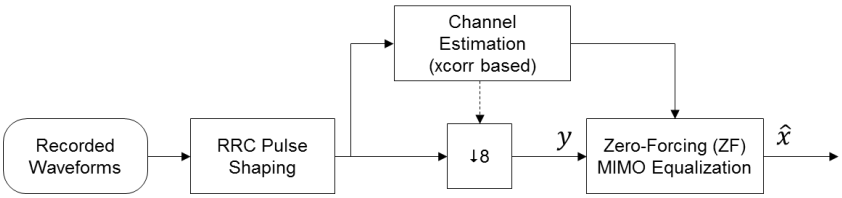


Figure 4-33: Post-processing of the recorded waveforms.

Based on the measurement data, estimated condition number of the channel matrix was 1.2, where 1 means perfect orthogonality. The PSD of the received signal streams are shown in Figure 4-34. The received symbols after equalisation are shown in Figure 4-35. One should note that the post-processing relies on simple zero-forcing, i.e., SFO, CFO, I/Q impairment or frequency-selectivity compensation is not performed. Applying these techniques, the performance can be additionally improved [32].

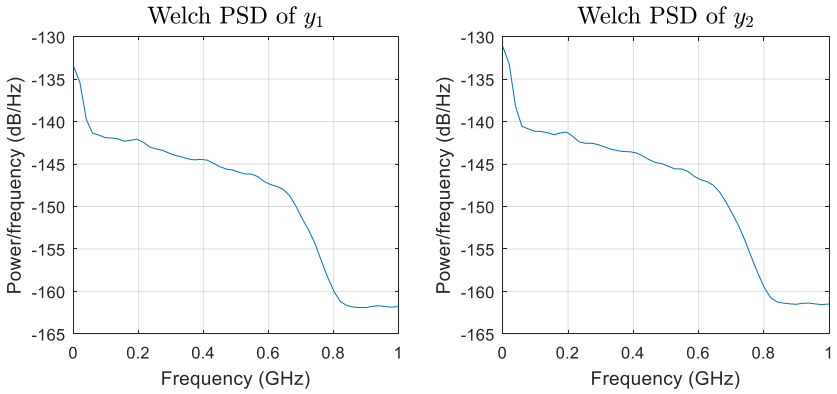


Figure 4-34: Power Spectral Density of received streams y1 and y2.

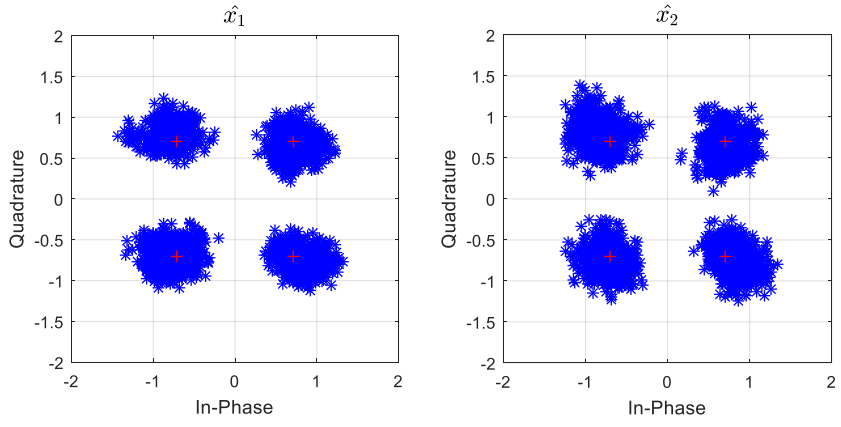


Figure 4-35: Equalised streams x1 and x2 (symbol-spaced) with rms EVM of 12.1 dB and 11.6 dB respectively.

Our future work, to be included in deliverable D3.3, will extend to improving the LoS mmWave MIMO performance by performing additional signal processing techniques (e.g. compensation of hardware impairments and so on). Also, it may include analysis of 3x3 MIMO or higher order LoS MIMO system with measurements in different environments (e.g. outdoors). On the other side, hybrid beamforming mmWave MIMO is to be tackled in the future work.

4.8 Fully programmable white-box edge device: experimental P4 use case for F-PU-5G performance evaluation

As already mentioned in section 3.4.2, ADVA chose the Xilinx-based IAF F-PU-5G FPGA board with pluggable GPP extension options, which was introduced in deliverable D3.1 [1] section 2.10. This board is seen as base for the development of a mainly P4-programmable NVF hosting platform, instead of the pure FPGA Xilinx VCU110 board described in deliverable D3.1 [1] section 2.9. An overview of the F-PU-5G HW layout is given again in Figure 4-36.

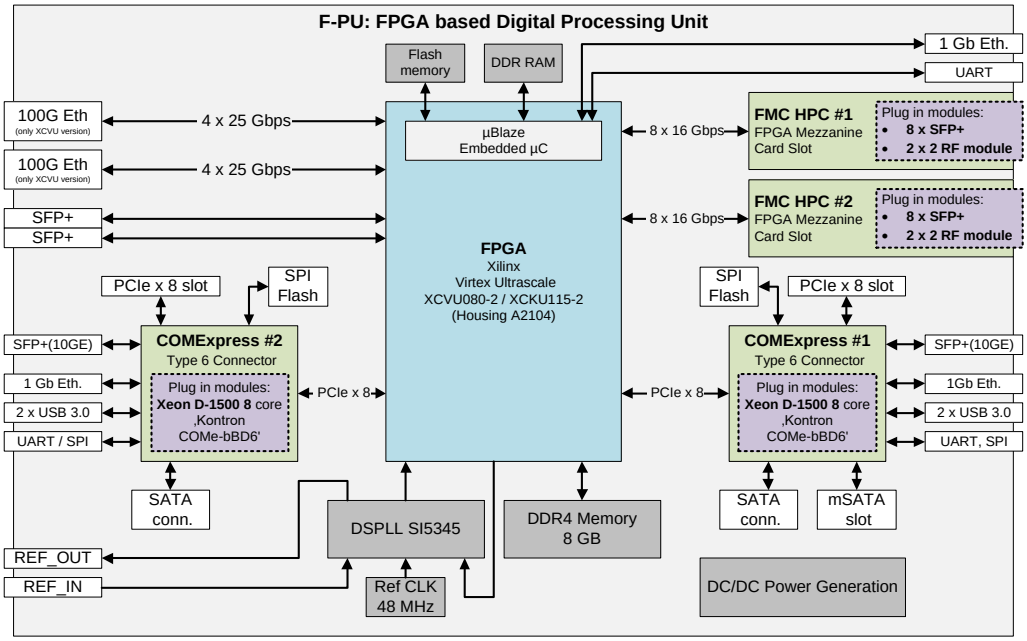


Figure 4-36: F-PU-5G hardware layout.

In this section we present an example from ADVA's ongoing P4-based FPGA design activities developed on top of the IAF F-PU-5G FPGA board. The use case is focusing on resource utilisation and latency measurement while creating and running P4 processing pipelines. The involved FPGA architecture components are shown in Figure 4-37, the FPGA layout for this use case is visualised in Figure 4-38.

The use case is functionally divided in two subtasks:

1. **Package processing** -- On the bidirectional packet streams, two common P4 processing features are applied, based on two exact match-action tables with 64 entries each:
   a. IPV4 Forwarding.
   b. VLAN Tagging.
2. **Communication between P4 Runtime and P4 Pipeline** – The external SDN-integrated P4 Runtime is connected to the FPGA's MicroBlaze[40] Soft Processor Core through its 1Gbit control interface. The MicroBlaze maps the received messages at the HW level into a P4-specific match-action table abstraction for the FGPA's P4-enabled IP core.

[40] https://www.xilinx.com/products/design-tools/microblaze.html

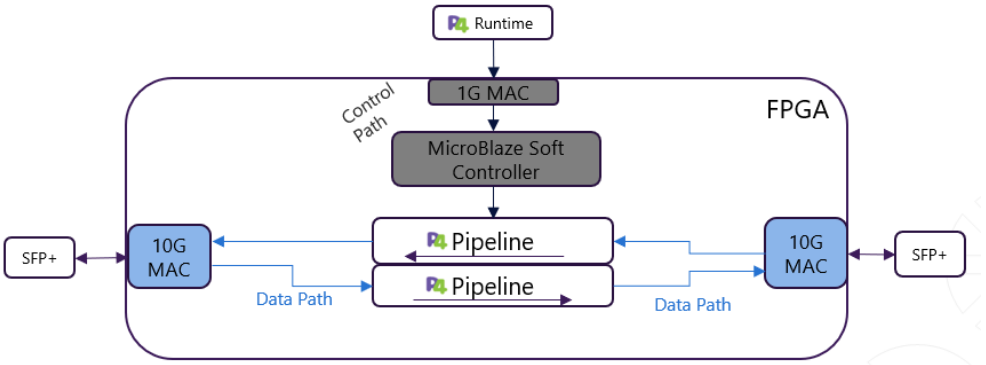


Figure 4-37: FPGA architecture components involved in P4 use case on F-PU-5G.

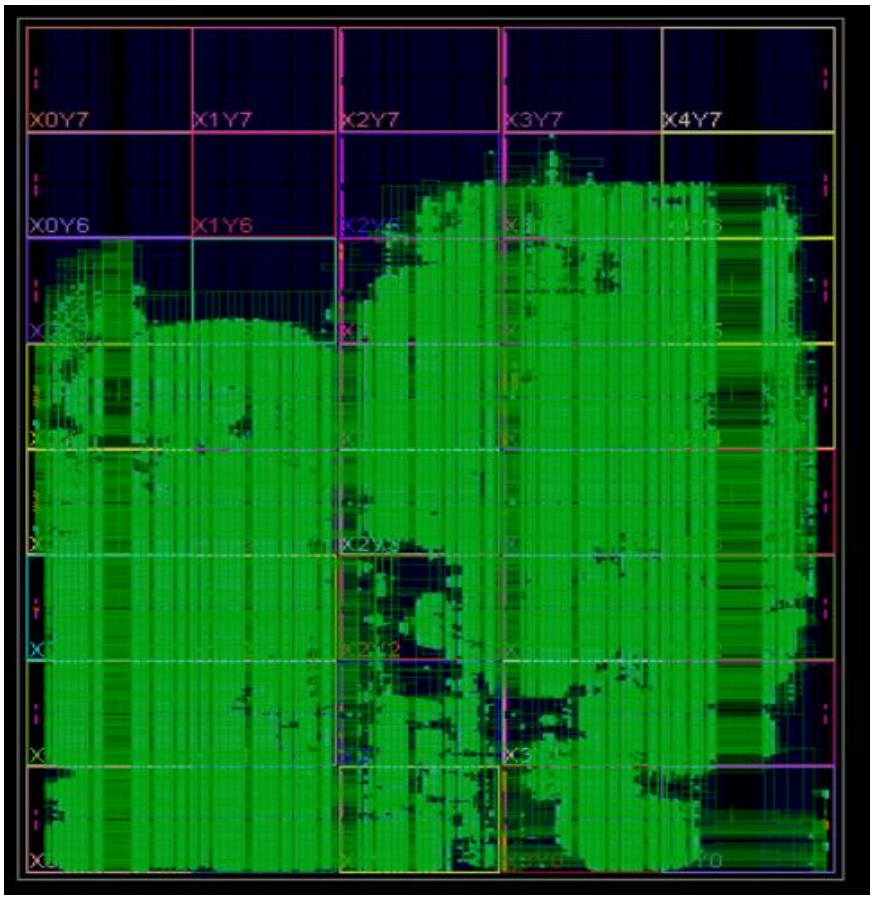


Figure 4-38: FPGA layout for P4 use case on F-PU-5G.

4.8.1.1 Resource utilization

The first investigated metric of the custom FPGA design is resource utilisation. An overall summary is given in Figure 4-39, which shows the percentage utilisation of specific FPGA resources, including Look-Up Tables (LUT), Flip Flops (FF), and Block RAMs (BRAM).

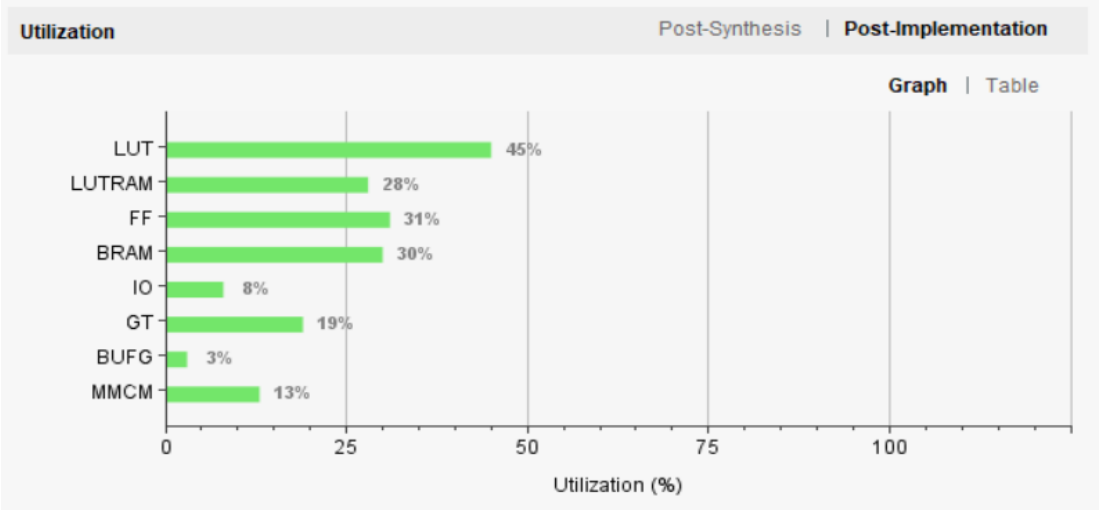


Figure 4-39: FPGA resource utilisation in P4 use case on F-PU-5G.

Figure 4-40 presents a detailed resource utilisation overview for a single P4 Pipeline, which can further be shortly summarised as:

- 20,09% of the available LUTs
- 13,68% of the available FFs
- 13,40% of the available BRAMs

Name	CLB LUTs (445712)	CLB Registers (891424)	Block RAM Tile (1421)
XilinxSwitch_0 (MB_System_XilinxSwitch_0_2)	89546	121968	190.5
U0 (MB_System_XilinxSwitch_0_2_XilinxSwitch)	89546	121968	190.5
Deparser (MB_System_XilinxSwitch_0_2_Depars...	74969	83388	106.5
Parser (MB_System_XilinxSwitch_0_2_Parser_t)	7432	14291	0
Pipeline_M_0 (MB_System_XilinxSwitch_0_2_Pip...	134	5932	0
Pipeline_M_2 (MB_System_XilinxSwitch_0_2_Pip...	121	4546	0
Pipeline_M_1 (MB_System_XilinxSwitch_0_2_Pip...	141	4190	0
ipv4_lpm (MB_System_XilinxSwitch_0_2_ipv4_lp...	1021	1516	12.5
S_SYNCER_for_S_SYNCER_for_Deparser (MB_...	1418	1262	13.5
Pipeline_M (MB_System_XilinxSwitch_0_2_Pipeli...	0	1245	0
S_SYNCER_for_S_SYNCER_for_S_SYNCER_for...	1247	1116	14.5
S_PROTOCOL_ADAPTER_INGRESS (MB_Syste...	23	1101	0
handleVLAN (MB_System_XilinxSwitch_0_2_hand...	830	867	7.5
S_SYNCER_for_S_SYNCER_for_S_SYNCER_for...	818	691	13.5
S_SYNCER_for_Deparser (MB_System_XilinxSwi...	810	687	13.5
S_PROTOCOL_ADAPTER_EGRESS (MB_System...	2	578	0
S_SYNCER_for__OUT_ (MB_System_XilinxSwitc...	287	253	8.5
S_BRIDGER_for_handleVLAN_tuple_in_request (...	185	141	0
S_BRIDGER_for_ipv4_lpm_tuple_in_request (MB...	85	125	0.5
S_RESET_clk_lookup (MB_System_XilinxSwitch...	5	0	0

Figure 4-40: FPGA resource utilisation of a single P4 Pipeline on F-PU-5G.

Also, worth noticing is that according to Figure 4-40, in the Xilinx P4 architecture the most resources are actually utilised by the Deparser:

- 16,82% of the available LUTs.
- 9,35% of the available FFs.
- 7,49% of the available BRAMs.

4.8.1.2 Latency measurement

The second investigated important metric is the latency introduced by a running P4 Pipeline in the FPGA. The P4 Pipelines in this use case are running at 156.25 MHz. The following results also show the impact of adding a new P4 match-action table by comparing a P4 Pipeline with one table to a P4 Pipeline with two tables. A short summary is given in Table 4-7. Figures Figure 4-41 and Figure 4-42 provide detailed overviews of the different variants:

Table 4-7: Latency impact of match-action tables in P4 Pipelines on F-PU-5G

P4 Pipeline	Clock cycles	Time
One match-action table	356	2,278 µs
Two match-action tables	395	2,336 µs

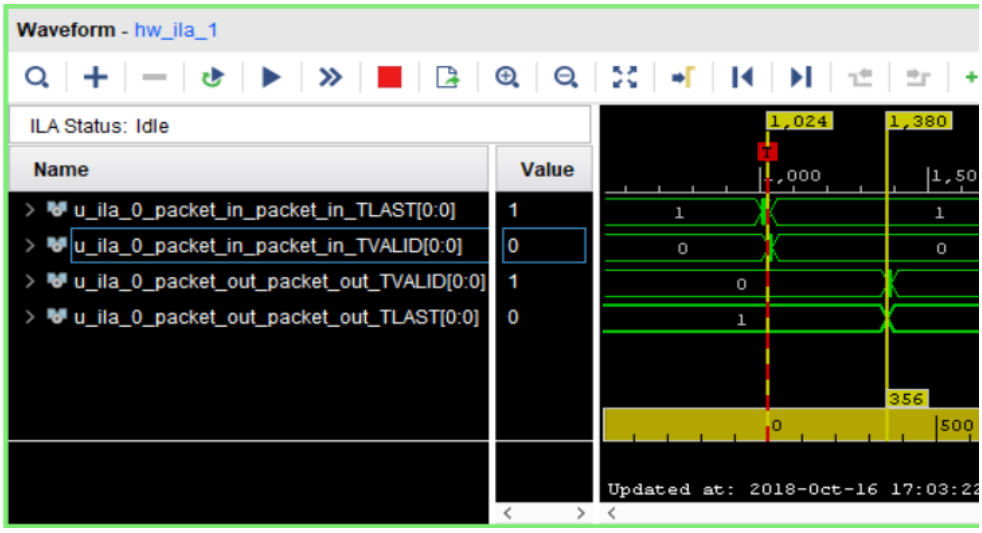


Figure 4-41: Latency of a P4 Pipeline with one match-action table on F-PU-5G.

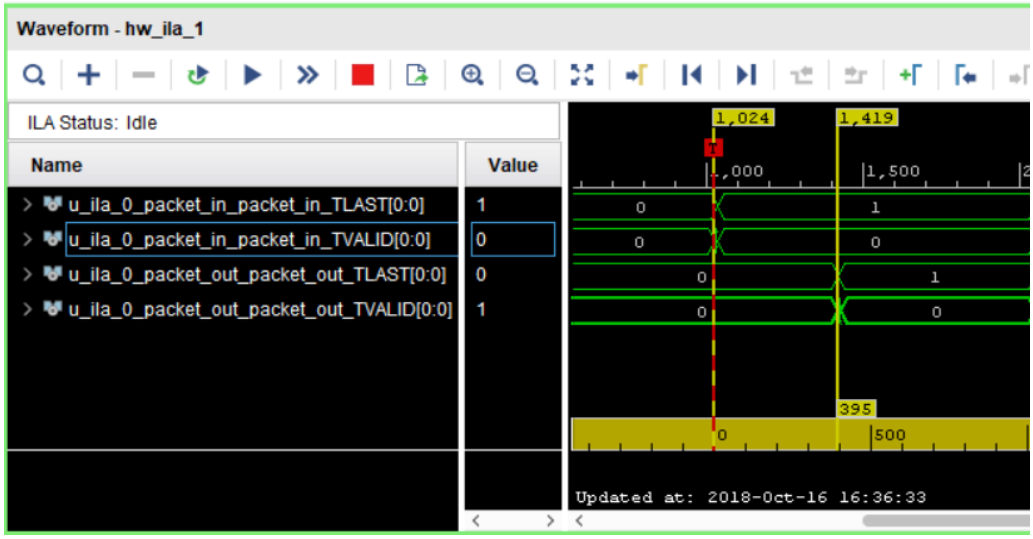


Figure 4-42: Latency of a P4 Pipeline with two match-action tables on F-PU-5G.

5 Summary and Conclusions

This document has presented the Intermediate report on Data Plane Programmability and infrastructure components developed in WP3 of the H2020 5G-PICTURE project. It describes the advancements with respect to the previous deliverable D3.1.

In particular, in section 2, the deliverable exposes the detailed description of the programmable platforms that has been developed during the WP3 activities. This detail description provides insights on the implementation results and on the preliminary evaluation of the programmable platforms that has been defined in deliverable D3.1. A quantitative assessment of the required FPGA resources for those platforms that have been developed using the FPGA as a target hardware architecture has been provided. The deliverable also presented an initial evaluation of the performance that the programmable platforms can achieve. In the rest of the project the functional validation and the performance assessment of these platforms will be carried out. In particular, for the back-haul, the platforms will provide a highly *programmable stateful dataplane* with latency and throughput performances able to sustain the need of the 5G network. For the fronthaul (FH), reconfigurable HW platforms will be used to provide both *SDN-based programmability* and efficient allocation of the processing tasks with flexible *functional splitting* between the radio units and the base station.

Section 3 describes, for the different platforms presented in this deliverable, several hardware abstractions that exposes the programmability to the upper layers. The deliverable presents the advancements with respect to deliverable 3.1 and focuses on the implementation details of the selected hardware abstractions. In particular, the implementation of the API/interfaces for the Typhoon, GateWorks Ventana and OpenAirInterface platforms and the NETCONF/YANG services for the passive WDM platform are described. Moreover, the programming languages for data plane programmability and the development of their compilation toolchains are presented. The detailed implementation of the hardware abstractions is presented, and the preliminary performance assessment and functional evaluation of the developed hardware abstractions is provided. A thorough evaluation of the hardware abstractions together with the integration of the different abstractions will be carried out during the rest of the project and will be documented in the deliverable D3.3.

Section 4 describes optical and radio technologies and components, which provide the integrated physical network infrastructure used in the 5G-PICTURE project. While section 5 of deliverable D3.1 presented the initial design of the hardware technologies, this deliverable discusses the achieved implementation results and shows the testbed implementation of the developed technologies and provides a preliminary performance evaluation.

6 Bibliography

[1] 5G-PICTURE, Deliverable 3.1: “Initial report on Data Plane Programmability and infrastructure components”, March 2018

[2] 5G-PICTURE, Deliverable D4.1: “State of the art and initial function design”, February 2018.

[3] 5G-PICTURE, Deliverable D5.1: “Relationships between Orchestrators, Controllers, slicing systems”, November 2018

[4] OIF, Flex Ethernet Implementation Agreement, OIF, 2016.

[5] V. V. a. B. K. T. Hofmeister, How can flexibility on the line side best be exploited on the client side?, Optical Fiber Communication Conference. Optical Society of America, 2016.

[6] R. Vilalta ; R. Martínez ; R. Casellas ; R. Muñoz ; Y. Lee ; L. Fei ; P. Tang ; V. López “Network slicing using dynamic flex ethernet over transport networks”, European Conference on Optical Communication (ECOC), 2017.

[7] E. e. a. I. Hussain, Gmpls routing and signaling framework for flexible ethernet (flexe), IETF draft-izh-camp-flexe-fwk-02, 2016.

[8] Huawei technologies, RH2288 V3 Server User Guide, Huawei technologies, 2017.

[9] Huawei technologies, OptiX PTN 990 Product Documentation- Product Version : V100R008C10SPC500, Huawei Technologies, 2018.

[10] I. s. f. E. 802.3-2015, Standard for Ethernet, http://standards.ieee.org/about/get/802/802.3.html., 2016.

[11] R. W. C. L. W. J. W. Rixin Li, "X-Ethernet: Enabling integrated Fronthaul/Backhaul Architecture in 5G Networks," IEEE Conference on Standards for Communications and Networking (CSCN), 2017.

[12] IEEE Std 1588-2008. (2008, 07), “IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems.”

[13] IETF Internet Draft draft-ietf-tictoc-1588v2-yang-10, ”YANG Data Model for IEEE 1588-2008 draft-ietf-tictoc-1588v2-yang-10”

[14] Xilinx, “Zync UltraScale+ RFSoC: Product Tables and Product Selection Guide”, 2018. Available: https://www.xilinx.com/support/documentation/selection-guides/zynq-usp-rfsoc-product-selection-guide.pdf

[15] Intel, “Intel Xeon Gold 6140 Processor”, Q3 2017. Available: https://ark.intel.com/products/120485

[16] D. B. Thomas, L. Howes, W. Luk, “A comparison of CPUs, GPUs, FPGAs, and massively parallel processor arrays for random number generation”, *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays*, p. 63-72. 2009.

[17] xRAN Forum, “XRAN-FH.MP.0-v01.00: “xRAN Fronthaul Working Group Management Plane Specification”, Technical Specification, 2018.

[18] C. Cascone, R. Bifulco, S. Pontarelli, and A. Capone.Relaxing state-access constraints in stateful pro-grammable data planes.ACM SIGCOMM ComputerCommunication Review, 48(1):3–9, 2018.

[19] N. Zilberman, Y. Audzevich, G. A. Covington, and A. W. Moore. NetFPGA SUME: Toward 100 Gbps asResearch Commodity.IEEE Micro ’14, 34(5):32–41, 2014

[20] G. S. Zervas, J. Triay, N. Amaya, Y. Qin, C. Cervelló-Pastor, and D. Simeonidou, “Time Shared Optical Net-work (TSON): a novel metro architecture for flexible multi-granular services,” Opt. Express 19(26), B509–B514 (2011).3GPP TS 23.402: Technical Specification Group Services and System Aspects; Architecture enhancements for non-3GPP accesses, Release 10, V10.5.0, September 2011.

[21] Y. Yan, Y. Qin, G. Zervas, B. Rofoee and D. Simeonidou, "High performance and flexible FPGA-based time shared optical network (TSON) metro node," 2012 38th European Conference and Exhibition on Optical Communications, Amsterdam, 2012, pp. 1-3.

[22] K. Nashimoto, K. Kudzuma, and D. Han, “Nano-second response, polarization insensitive and low-power consumption PLZT 4×4 matrix optical switch,” in conference of Optical Fiber Communication Conference and Exposition (OFC/NFOEC), 2011.

[23] 5G-XHaul Project, Deliverable D2.2 ” System Architecture Definition”, submitted on July 1st, 2016.

[24] 5G-XHaul Project, Deliverable D3.1 ”Analysis of state of the art on scalable control plane design and

techniques for user mobility awareness. Definition of 5G-XHaul control plane requirements", submitted on July 15th, 2016.

[25] "IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems", IEEE Std 1588-2008 (Revision of IEEE Std 1588-2002), vol. 1, pp. 1–269, Jul. 2008.

[26] https://www.xilinx.com/products/intellectual-property/axi_10g_ethernet.html

[27] https://www.xilinx.com/support/documentation/ip_documentation/axi_10g_ethernet/v3_1/pg157-axi-10g-ethernet.pdf

[28] Madhow, "Spatial multiplexing over a line-of-sight millimeter-wave MIMO link: A two-channel hardware demonstration at 1.2Gbps over 41m range," 2008 European Conference on Wireless Technology, Amsterdam, 2008, pp. 198-201.

[29] T. Hälsig and B. Lankl, "Array Size Reduction for High-Rank LOS MIMO ULAs," IEEE Wirel. Commun. Lett., vol. 4, no. 6, pp. 649–652, 2015.

[30] X. Song, C. Jans, L. Landau, D. Cvetkovski and G. Fettweis, "A 60GHz LOS MIMO Backhaul Design Combining Spatial Multiplexing and Beamforming for a 100Gbps Throughput," 2015 IEEE Global Communications Conference (GLOBECOM), San Diego, CA, 2015.

[31] D. Cvetkovski, T. Hälsig, B. Lankl, and E. Grass, "Next Generation mm-Wave Wireless Backhaul Based on LOS MIMO Links," in Proc. Ger. Microw. Conf., 2016, pp. 69–72.

[32] X. Song et al., "Design and Experimental Evaluation of Equalization Algorithms for Line-of-Sight Spatial Multiplexing at 60 GHz," in IEEE Journal on Selected Areas in Communications. doi: 10.1109/JSAC.2018.2872286

[33] 5G-XHaul Project, Deliverable D3.3 "5G-XHaul algorithms and services Design and Evaluation", submitted on June 31st, 2018.

[34] 5G-XHaul Project, Deliverable D5.3 "Demonstration and Evaluation of the 5G-XHaul Integrated Prototype", submitted on June 30th, 2018.

[35] https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_1/ug1252-p4-sdnet-translator.pdf

[36] https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_4/ug1252-p4-sdnet.pdf

[37] https://www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf

7 Acronyms

Acronym	Description
AADU	Active Antenna Distributed Unit
ADC	Analogue-to-Digital Converter
AoA	Angle of Arrival
AoD	Angle of Departure
AP	Access Point
API	Application Programming Interface
B2B	Back to Back
BB	BaseBand
BBU	BaseBand Unit
BER	Bit Error Rate
BH	Backhaul
BS	Base Station
BVT	Bandwidth Variable Transmitter
BWT	Blu Wireless Technology
CAN	Controller Area Network
CAPEX	CAPital EXpenditure
CD	Clock Doubler
CDC	Clock Domain Crossing
CFR	crest-factor reduction
CP	Control Plane
CPRI	Common Public Radio Interface
CU	Centralised Unit
C-RAN	Cloud Radio Access Network
DAC	Digital-to-Analogue Converter
DA-RAN	Dis-Aggregated Radio Access Network
DC	Data Centre
DCB	Data Centre Bridging
DDOS	Distributed Denial of Service
FF	Flip Flop
DL	Download
DSL	Domain Specific Languages
DSP	Digital Signal Processing
D-RAN	Distributed Radio Access Network
DU	Distributed Unit
DWDM	Dense Wavelength Division Multiplexing
ECN	Explicit Congestion Notification
EDFA	Erbium Doped Fibre Amplifier
EDP	European Deployment Plan

EFSM	Extended Finite State Machine
FCS	Frame Check Sequence
FDD	Frequency division Duplexing
FH	Fronthaul
FIFO	First In First Out
FMC	FPGA Mezzanine Card
FPGA	Field Programable Gate Array
FSM	Finite State Machine
GPGD	gap-detection scheme
GPP	General Purpose Processor
GPU	Graphics Processing Unit
GWVBBU	Gateworks Ventana (single board computer)Baseband Unit
HARQ	Hybrid Automatic Repeat Request
HIL	Hardware-In-the-Loop
HW	Hardware
HWA	Hardware-accelerated
HYDRA	Hybrid Defined Radio Architecture
IEC	International Electro-technical Commission
IEEE	Institute of Electrical and Electronic Engineers
IHON	Integrated Hybrid Optical Networks
IHP	Innovations for High Performance microelectronics
IoT	Internet of Things
IQ	In-phase Quadrature
NICISU	Network Interface CardSub-Unit
ITU	International telecommunication Union
LAA	Large Aperture Array
LCV	Line Code Violations
OFLTE	OpenFlowLong Term Evolution
LTE-A	Long Term Evolution Advanced
LUT	LookUp Table
MAC	Media Access Control
mDC	micro Data Center
MEC	Multi-access Edge Computing
MIMO	Multiple-input and multiple-output
MME	Mobility Management Entity
MPI	Multi PHY Interfaces / Multi-Protocol Interfaces
MTU	Maximum Transmission Unit
MU	Multiple-Unit
NB-IoT	Narrow Band Internet of Things
NFV	Network Function Virtualisation
OAI	OpenAirInterface

OBSAI	Open Base Station Architecture Initiative
OVSOBSAI	Open vSwitch Open Base Station Architecture Initiative
OLT	Optical Line Terminal
ONU	Optical Network Unit
RESTOPEX	Representational State TransferOPerational EXpenditure
ODL	OpenDayLight
OPP	Open Packet Processor
PCS	Physical Coding Sublayer
PDCP	Packet Data Convergence Protocol
PDV	Packet Delay Variation
PHY	Physical
PMOD	Peripheral Module
PMA	Physical Medium Attachment
PMP	Packet Manipulator Processor
PNF	Physical Network Functions
POE	Power over Ethernet
PON	Passive Optical Networks
PPP	Public Private Partnership
PRB	physical resource block
pWDM	passive Wavelength Division Multiplexing
P2MP	Point-to-Multipoint
PTP	Precision Time Protocol
QAM	Quadrature Amplitude Modulation
QoS	Quality of Service
QoT	Quality of Transmission
QPSK	Quadrature Phase Shift Keying
RISC	Reduced Instruction Set Computer
RPC	Remote Procedure Call
RRH	Remote Radio Head
RSRP	Reference Signal Receive Power
RSU	Radio Sub-Unit
RU	Radio Unit
SDN	Software Defined Networking
SLA	Service Level Agreement
SerDes	Serializer/Deserializer
SIL	server instrumentation library
SINR	signal-to-interference and noise ratio
SM	Spatial Multiplexing
SMA	SubMiniature version A
SMT	Statistically Multiplexed Traffic
SON	Self-Organizing Network

SO-RAN	service-oriented RAN
SSMF	Standard Single Mode Fibre
SW	Software
TC	Transparent Clock
TCAM	Ternary Content Address Memory
TCP	Transmission Control Protocol
TDM	Time Division Multiplexing
ToD	Time of Day
TSN	Time Sensitive Network
TSON	Time Shared Optical Network
TTI	transmission time interval
UDP	User Datagram Protocol
UMTS	Universal Mobile Telecommunications System
UP	Upload
vBBU	Virtual Baseband Unit
VIFSAI	VirtualSwitch Abstraction Interface
VLIW	Very Long Instruction word
VLAN	Virtual Local Area Network
VNF	Virtual Network Function
WDM	Wavelength Division Multiplexing
WDM-PON	Wavelength Division Multiplexing Passive Optical Network
XFSM	Extended Finite State Machine
YANG	Yet Another Next Generation (data modelling language)
YANGSBC	Yet Another Next Generation Single-Board Computer