5G Programmable Infrastructure Converging disaggregated network and compUte REsources

D3.1 Initial 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: 30 months

Call: H2020-ICT-2016-2

Date of delivery: March 31st 2018
April 4th 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:	D3.1
Document Title:	Initial report on Data Plane Programmability and infrastructure components
Version:	1.0
Delivery Date:	March 31st 2018 (April 4th 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), Marcus Ehrig (IHP), Paris Flegkas (UTH), Eduard García-Villegas (i2CAT/UPC), Jesús Gutiérrez (IHP), Jong Hun Han (UNIVBRIS-HPN), Joan J. Aleixendri (i2CAT), Matty Kadosh (MLNX), Vaia Kalokidou (UNIVBRIS-CSN), Nikos Makris (UTH), Nebojsa Maletic (IHP), 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:	Final
Dissemination Level	Public
Project URL:	http://www.5g-picture-project.eu/

Revision History

Rev. N	Description	Author	Date
0.1	Initial Draft	Salvatore Pontarelli (CNIT)	07/12/2017
0.2	First contributions integrated	Salvatore Pontarelli (CNIT)	26/02/2018
0.3	Added IHP-BWT-EUR contribution	Salvatore Pontarelli (CNIT)	01/03/2018
0.4	Added IHP contribution on MIMO	Jesús Gutiérrez (IHP)	13/03/2018
0.5	First content check	Jesús Gutiérrez (IHP)	14/03/2018
0.6	Added UNIVBRIS-HPN contribution on Optical Programmability	Jong Hun Han (UNIVBIS-HPN)	15/03/2018
0.7	First internal review	Stefan Zimmermann (ADVA)	15/03/2018
0.8	Incorporated reviewers comments	Salvatore Pontarelli (CNIT)	26/03/2018
0.9	Incorporated additional authors	Salvatore Pontarelli (CNIT)	28/03/2018
0.95	Final technical review	Anna Tzanakaki (UNIVBRIS-HPN)	03/04/2018
1.0	Final review and submission	Jesús Gutiérrez (IHP)	04/04/2018

Table of Contents

List of Figures

List of Tables

Executive Summary

This document corresponds to deliverable D3.1, “Initial report on Data Plane Programmability and infrastructure components” of the H2020 5G-PICTURE project. The deliverable provides the description of the platforms for data plane programmability and the initial specification of the interfaces, programming models, and hardware abstractions that will be developed during the course of the project.

Several solutions for data plane programmability, dealing with the design and implementation of programming platforms for both fronthaul/signal processing and backhaul/packet processing are presented together with the relevant exposed methods to abstract the underlying platforms. Finally, the deliverable reports the study on infrastructure components and multi-protocol/multi-PHY interfacing technologies.

A more precise and definitive detailed architectural description of the programmable platforms will be provided in deliverable D3.2, with due date November 2018.

Introduction

Network nodes (being wired switches, radio stations, or even end terminals) have been traditionally developed for extremely specific purposes: support a given and possibly small set of **communication or forwarding needs**. Indeed, innovation was (and of course still is) driven by lengthy standardization processes, devised to specify the behaviour and the inter-operation of the nodes and to formalize such specification into one or more protocols, independently implemented once for all by device vendors in closed products. This historical trend allowed the widespread diffusion of several network technologies providing a constant improvement of the network nodes performance.

Unfortunately, networks today are extremely complex and diversified: the original Internet nodes, historically limited to switches and routers, have been massively complemented with a variety of heterogeneous middle-boxes, such as network address translators, tunnelling entities, load balancers, firewalls, intrusion detection systems, traffic monitoring probes, etc. The networks today must be made able to promptly accommodate (in very different environments) very diverse end points, from handheld devices to sensors and actuators integrated into physical objects (the so-called "Internet of Things"), and are called to sustain a dramatic evolution not only in terms of scale ("big data"), but also in terms of complexity and diversity in their traffic generation patterns and relevant requirements (human end points, machine-to-machine relations, content retrieval from caching and replica servers, flash crowd events, etc.).

This complex scenario makes much more difficult to maintain the above mentioned development model of network nodes in which they were designed to efficiently perform an extremely specific task and to support a well-defined set of protocols. Instead, today it is more and more important to have flexible and reconfigurable network nodes that are able to: i) support different sets of functionalities depending on the specific network location in which they are deployed; ii) dynamically change the supported set of functionalities depending on the network condition and/or the type of traffic/service to manage and iii) be easily upgradable if different service/protocols arise. On the other hand, this flexibility cannot be traded off against network performances.

In this scenario programmable network platforms are the enabling factor that allows developing complex network functionalities without compromising the performance levels achievable with fixed purpose network devices. These platforms should provide several common aspects in the different network domains (dataplane, optical, and radio access). 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 P4 language for programmable dataplanes or the OpenCL language for the definition of digital signal processing (DSP) radio functions. Moreover, 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. Finally, a set of hardware abstractions and/or interfaces must be developed to provide to the upper layers of the network management a sort of API to easily manage the configuration of the programmable platforms.

This deliverable discusses the initial specification of the interfaces, programming models, and hardware abstractions of the programmable network platforms that will be developed in the 5G-PICTURE project.

Organisation of the document

This deliverable is structured as follows: Section 1 provides a state-of-the-art review programmability of different network technologies. This includes dataplane, optical network, and radio access network. In section 2, the document presents the various target hardware platforms selected by the 5G-PICTURE partners as foundations for their programmable network platforms. Some of the hardware platforms are result of development work performed by some specific partners. The functionalities of these platforms will be enhanced and/or further exploited for improving programmability capabilities. Some of the other hardware platforms are off-the-shelf platform acquired by the partners. In the latter case, the platform choices are not only based on partner-specific programmability requirements, but also on easy integration of the work done by different partners. Section 3 presents the initial functional definitions of the programmable platforms that are under development in the current phase of the project and will be finalized in deliverable D3.2. The platform programmability requires a set of methodologies allowing common abstractions of the underlying hardware. Those methodologies, or simply called hardware abstractions, are presented in section 4. Section 5 illustrates the different

hardware technologies developed in 5G-PICTURE that provide basic building blocks of the 5G network architecture, comprising novel passive and elastic optical as well as RF and baseband (BB) processing technologies. Finally, section 6 contains the summary and conclusions.

1 State of The Art

This section summarises the State-of-The-Art (SoTA) of the programmability features of different network elements composing the 5G scenario. In particular, it describes the programmability of data plane, optical network, and C-RAN.

1.1 Data-plane programmability

5G networks must efficiently and flexibly support an ever growing variety of heterogeneous middlebox-type functions such as network address translation, tunneling, load balancing, traffic engineering, monitoring, intrusion detection, etc. In the last years, the community has attempted to address the programmability of such network functions with two complementary approaches: Software-Defined Networking (SDN) and Network Function Virtualization (NFV).

However, these approaches have so far remained disjoint. SDN has focused on the clean separation of control and data plane via open interfaces. It has exploited re-configurability of high performance switching hardware only to a very small extent (i.e. due to the limited flexibility of OpenFlow [1] as southbound interface). Conversely, NFV has fostered full programmability of network functions, but mainly via SW on commodity platforms. Hence it is subject to performance limitations, and in general not relying on open programming interfaces. In this section, we first describe the SDN/OpenFlow limitations, and then we discuss the most up-to-date research and industrial initiatives that aim to supersede these limitations.

1.1.1 SDN/OpenFlow: limitations and extensions

Early SDN approaches (and still most of today's real world deployments) rely on the relatively poorly flexible OpenFlow abstraction as southbound (i.e. node-level, using RFC 7426's terminology) programming interface. OpenFlow is perfectly suited to configure forwarding behaviours executed at wire speed in the switches, expressed as switch/router flow tables, but ***shows severe limitations when it comes to deploy more complex (e.g. stateful) flow processing and filtering functions***. For this reason, most of today's network programming frameworks circumvent OpenFlow's limitations by promoting a "two-tiered" [2] programming model: any stateful processing intelligence of the network applications is delegated to the network controller, whereas OpenFlow switches are limited to install and enforce stateless packet forwarding rules delivered by the remote controller. This delegation of intelligence to the centralized controller thus causes performance, latency, and signalling overhead, which ***hinders the deployment of truly scalable software-implemented control plane tasks at wire speed***, i.e. while remaining on the fast path.

This problem is of course not new, and also well-known by the Open Networking Foundation (ONF), the body which standardises OpenFlow since 2011. Indeed, in the course of the OpenFlow standardisation process, we have witnessed a hectic evolution of the standard, but backward compatibility reasons and pragmatism have so far prevented OpenFlow from incorporating the flexibility necessary for implementing advanced packet processing tasks. As a matter of fact, and up to the latest specification version 1.5 [3], several OpenFlow extensions have been devised to fix punctual shortcomings and accommodate specific needs. Such evolution has led to the incorporation of ***extremely specific stateful primitives*** into the OpenFlow standard (such as meters for rate control, group ports for fast failover support or dynamic selection of one among many action buckets at each time - e.g. for load balancing -, synchronized tables for supporting learning-type functionalities, etc. – see details in [3]). Despite such many tailored OpenFlow extensions, we are still very far from being able to deploy typical Middlebox appliance features in an OpenFlow switch. At the same time, another limitation of the OpenFlow approach is emerging. It is related to the way the header fields are defined in the OpenFlow standard. In fact, the standard explicitly defines the specific packet fields that are used as inputs for the OpenFlow *match/action* stages. Therefore, it is not possible to use a custom defined header for the matching stages. This limitation has been faced until now by defining a new set of fields in each revision of the standard. The first OpenFlow version had only 12 fields, while the OF1.5 revision defines up to 42 fields [3]. Since this approach was not sustainable, the proposal of a complete protocol independent packet processing has emerged [4].

1.1.2 Fully programmable data plane switches

As already mentioned, even if NFV in principle addresses full programmability of network functions, it comes with two fundamental caveats. First, approaches proposed so far do not provide an "open" programming model for the data plane operation of the network function itself. Second, fast-path (wire speed) processing in SW for multi gigabit/s links, even if in principle attainable with massive multi-core parallelization and Network Interface Card (NIC)-level accelerations, e.g. Data Plane Development Kit (DPDK), remains extremely demanding, es-

pecially when the design goal is to retain independence from the underlying platform. The point is that ***involving a CPU in packet-by-packet processing comes with an overly severe overhead*** (with multiple micro-instructions frequently necessary to perform a packet-level operation that in HW could be even implemented within just a clock cycle). It is hardly compatible with the very tight requirements of a wire-speed/fast-path processing task (a 64 bytes packet takes only about 5 ns on a 100 Gb/s speed). And, to make things worse, software solutions cannot take advantage of dedicated HW components, such as Ternary Content-Addressable Memories (TCAMs), which can trivially solve problems like wildcard matching in *O(1)* complexity. As of now, it does not yet exists an equivalent *O(1)* complexity software implementation counterpart.

For these reasons, in the last couple of years, a new research trend has started to challenge improved programmability of the data plane via domain-specific packet processing HW platforms or chipsets, which still attempt to expose very flexible and general programming interfaces and languages (somewhat comparable to CPU-based programming), but are specifically designed for packet-level processing tasks.

Probably ***the most popular initiative in this fresh field is P4*** [4]. The P4 initiative started from the observation that while in the past network switches had a fixed and well-known behaviour, today a new generation of fully reprogrammable high speed HW switch architectures is emerging. Representative example architectures include the reconfigurable match tables introduced in [35], the Intel FlexPipe technology [5] and, at least to some extent, the way more flexible header matching promoted by the Huawei Protocol Oblivious Forwarding (POF) initiative [6]. The P4 programming language [4] thus emerged as an attempt to **programmatically describe the packet processing pipeline** [35] via "packet programs" written in a high level language that can be compiled for different HW targets. Still, despite the current P4 hype and the significant attention that P4 has received by the networking community, it is at least fair to say that P4 is not yet "the" solution to "all possible" data-plane programming needs. We specifically see at least three major open issues, which motivate (and give the basis of) our planned work within the 5G-PICTURE project.

First, ***P4 is of course NOT a programmable switch architecture, but it is (and remains) a programming language*** which, as such, requires some underlying HW architectures (chipsets) capable of "running" P4 packet programs. Surprisingly, while P4 "as a language" has been dissected into full details, very little research has been disclosed so far on the platforms devised to support it. In the last year several industrial initiatives started to provide P4 compatible hardware platforms: the Barefoot Tofino™ programmable switch series has been presented few months ago[1], the Netronome Smart NIC is able to use P4 as a programming language[2], and also a Field Programmable Gate Array (FPGA) based board from Netcope[3] has recently appeared in the market. The analysis and the design of novel hardware architectures able to support P4 programmability seems to be a novel and fast growing research activity with still many open questions. One of the scopes of 5G-PICTURE is thus to understand how to compile P4 programs for already existing HW platforms (HW platforms provided by the switch manufacturers involved in 5G-PICTURE), and, complementarily, (ii) how to extend current HW to support P4 packet programs. A second, and perhaps even more fundamental question, relates to ***thoroughly understanding if there are limitations in how P4 permits to describe some subset of wire-speed flow/packet processing tasks*** and, if this is the case, ***how to extend P4 capabilities – or identify novel approaches – to cover such gaps***. It is a fact that P4 was initially devised with "only" packet-level processing tasks in mind, i.e. processing tasks which take as input a given packet and process it (and forward it) on the basis of information associated with the packet itself (e.g. packet structure). Stateful processing was thus initially restricted to "packet-level states" (e.g. states specified while parsing an individual packet), opposed to "flow-level states", i.e. states which persist (and which are updated) across subsequent packets of the same flow. With version 1.0.2 [7], the P4 specification has made a further initial step in ***supporting per-flow stateful processing*** by introducing registers defined as stateful memories, which can be used in a more general way to keep state. However, these stateful constructs seem to be sort of a "side-patch" to P4 (opposed to a native fundamental feature) whose support is mandated to the actual target platform. Indeed, the P4 language does not natively provide means to address and fetch registries (or counters) or means to associate registries with flows without incurring in access collisions[4]. Rather, as explicitly stated in

[1] https://www.barefootnetworks.com/products/brief-tofino/

[2] https://www.netronome.com/technology/p4/

[3] https://www.netcope.com/en/products/netcopep4

[4] The technical underlying problem is how to persistently associate a register to a flow. Without any dedicated primitive or data structure providing such an association, most P4 algorithms circumvent such problem by exploiting an Hash-value generator, provided as P4 language primitive, which associates an integer value to any arbitrary bitstring (e.g. a flow name). Such an associated integer can be then

the latest language specification [8], stateful constructs such as counters and meters are represented using external objects and must be explicitly supported by the target and allocated at compilation-time through a process called “instantiation”. To address such shortcomings, 5G-PICTURE is specifically focusing on innovative abstractions for describing stateful flow processing, and on the design of the underlying architectures devised to support such abstractions as will be discussed in section 3.3.

1.2 Optical Network Programmability

The optical network becomes more and more dynamic in its architecture and requires frequent network reconfigurations. Dynamic optical networks require all kinds of visibility into application data types, traffic flows, and end-to-end connections [9]. In addition to the dynamic optical network, the optical network needs to adopt programmability to address the very diverse and high bandwidth connectivity requirements of the 5G network. Optical network programmability will be based on active and elastic technologies. In terms of active technologies, current commercially available solutions perform optical switching supporting wavelength switching granularity. However, given the very diverse requirements of operational and end-user services in the context of 5G, there is a need for new approaches, deploying more dynamic and flexible solutions to offer higher granularity at the sub-wavelength level and more elasticity in the optical domain. In view of these new requirements, an elastic frame-based WDM active solution in combination with a passive optical network (PON) is proposed. The active optical network solution proposed by 5G-PICTURE is referred to as Time Shared Optical Network (TSON) [12] that provides variable sub-wavelength switching granularity and the ability to dynamically allocate optical bandwidth in an elastic way, while low-cost point-to-point connections with limited flexibility (e.g. between the edge network and remote cells) can be also supported through passive Dense Wavelength Division Multiplexing (DWDM) networks – WDM Passive Optical Networks (WDM-PONs). The elastic optical network discussed in section 5.2 can be enabled by the adoption of a flexible channel grid and programmable transceivers [10][11]. This is especially interesting in the context of supporting greatly varying transport services – both fronthaul (FH) and backhaul (BH) – for the RAN in 5G networks. Furthermore, optimizing the utilization of the available optical bandwidth can increase the optical capacity. Dynamic bandwidth provisioning helps to the overall network spectral efficiency. To enable flexible grid elastic optical networks, two key technologies are required: (1) flexible grid wavelength selective switches or spectrum selective switches which support high bandwidth granularity; (2) elastic transponders with variable data rate (corresponding to the occupied optical spectrum) and adaptable modulation format. 5G-PICTURE focuses on next-generation elastic optical networks to support 5G traffic requirements including research on switching nodes with enhanced flexibility and software-programmable transceivers that are integrated through a control plane in support of optical network programmability.

1.3 C-RAN programmability

The monolithic RAN programmability is mainly enabled by applying software-defined networking principles into RAN, i.e. SD-RAN, for decoupling the control plane (CP) from the data plane processing. Several works discuss the level of centralization of CP functionalities. A fully centralized architecture is proposed in OpenRAN [13] and SoftAir [14]. It will face the challenge of real-time control under the inherent delay between the centralized controller and distributed RANs. While the SoftRAN [15] architecture statically refactors the control functions into centralized and distributed ones based on the time criticality and central view requirement, the SoftMobile approach [16] further abstracts the CP in different network layers based on their functionalities to form the network graphs, and performs control functionalities through the Application Programming Interface (API). As for the data plane programmability and modularity, the OpenRadio [17] and PRAN [18] are pioneered to decompose the overall processing into several functionalities that can be chained. The blueprint proposed as RadioVisor in [19] aims to isolate the control channel messages, elementary resources like CPU and radio resource to provide customized service for each slice. The FlexRAN platform [20] realizes an SD-RAN platform and implements a custom RAN south-bound API through which programmable control logic can be enforced with different levels of centralization, either by the controller or RAN agent.

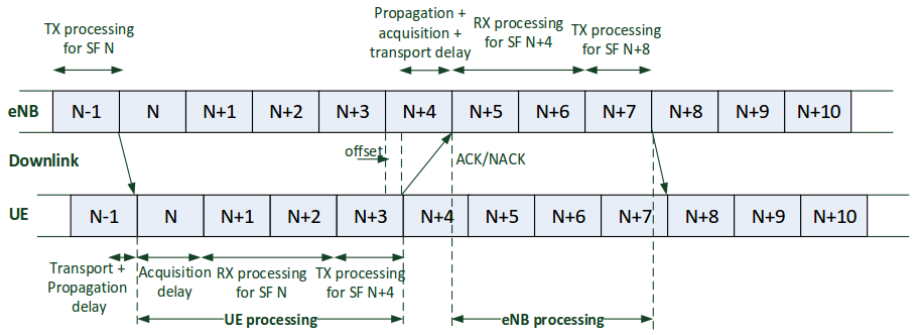
Evolving from a monolithic RAN towards disaggregated modules, the Cloud RAN (C-RAN) vision [21] aims to distribute the radio access network functions from a monolithic base station (BS) among distributed entities,

used to reference a register. However, this is a “patch” which may not satisfy the need of processing tasks which deploy several flow states (in principle up to even one state per each active flow, with the number of flows possibly being in the order of millions!). In fact, P4 does not provide any native mean to handle hash collisions (which are exacerbated by the relatively small number of registers available in the underlying hardware – hence the need to rely on small-domain hash digests). As such either the programmer develops her own collision handling means, hence most likely losing the *O(1)* property of the pipelined processing and making its application infeasible at wire-speed, or she needs to live with hash collisions which may severely impair the semantics of the developed flow processing task.

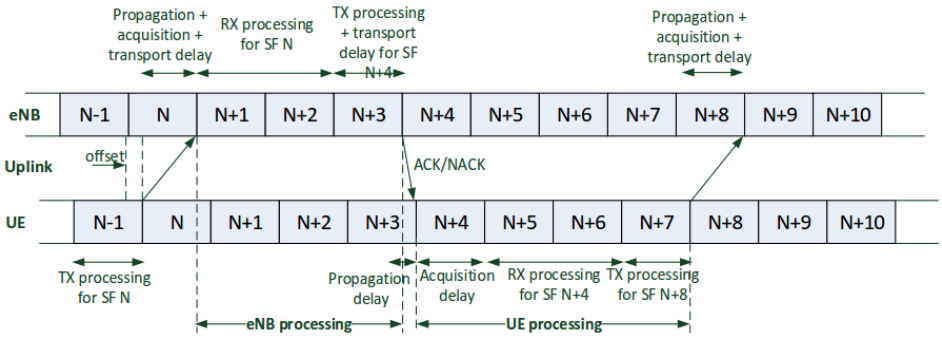
i.e. radio unit (RU), distributed unit (DU), and centralized unit (CU) as mentioned in Deliverable D4.1 [22]. The RAN functionalities are segmented based on the applied functional splits between entities, for instance the split option 1 to option 8 surveyed by 3GPP [23]. Such function distribution enables the flexibility in the future ultra-dense RAN deployment, i.e., densification at the remote RU level while centralize the corresponding DU and CU in the cloud infrastructures. Further, C-RAN retains the benefits of centralization to enable a coordinated and cooperative processing at both DU and CU levels. Nevertheless, the programmability of such C-RAN vision is more challenging as the relation between CN, DU, and RU follows the 1:N:M manner, and thus the remote network functions shall be controlled and managed centrally to properly maintain the end-to-end RAN service. For instance, when applying real-time beamforming and combining operation at the DU in its physical layer, the different FH transportation delay between a single DU to the corresponding RUs may impact the perceived performance.

Further, the chaining of RAN functions among distributed entities shall meet the latency constraint inherent of the underlying radio access technologies (RATs). For instance, in an LTE system of frequency division duplexing (FDD) mode, the Hybrid Automatic Repeat Request (HARQ) round trip time is 8 millisecond, and the deadline shall be met in the two examples shown in Figure 1, with downlink and uplink direction, respectively. In [24], the processing time of FFT, (de)modulation, and (de)coding among different virtualization technologies like virtual machines (e.g., KVM) or containers (e.g., Docker) is measured and modelled based on the experiments conducted in the OpenAirInterface (OAI) platform. Based on such modelling, several works aim to conduct the dynamic resource provisioning to enable C-RAN programmability. The authors of [25] provide an algorithm to dynamically select active RUs and virtual machines to DUs for energy efficiency improvement. In [26], the parallelization of centralized functions (i.e., DU/CU) is surveyed and the different schedulers for executing jobs over available cores are provided.

Besides, in order to support multiple services in the RAN domain, the disaggregated RAN will be able to support the customized network function for each of the instantiated slices. For instance, a service may request a hardware-based accelerator for the channel decoder and a dedicated radio resource scheduling to enable the low-latency communications, whereas another service may only require certain level of performance guarantee (i.e., good-put) without any customization (e.g., over-the-top providers). In this sense, such a chain can be composed horizontally between aforementioned RAN entities (i.e., RU, DU and CU), and/or vertically when customized control/data plane processing is required to create the slice tailored to the service requirements.



(a) DL HARQ timing.



(b) UL HARQ timing.

Figure 1: FDD LTE timing.

Figure 2 shows the input/output forwarding path between CU, DU, and RU to compose the three slice-specific user plane processing chain with 3GPP function splits, option 2 between CU and DU, and option 6 between DU and RU. For slice 1, it first customizes the network functions of Service Data Adaptation Protocol (SDAP) and Packet Data Convergence Protocol (PDCP) at CU, then also customizes the Radio Link Control (RLC) and Medium Access Control (MAC) functions at DU, while utilizing the shared Physical (PHY) layer function at RU. By contrast, slice 2 only customizes the PDCP and RLC at CU, and slice 3 utilizes all shared function without customization. Such data plane forwarding can rely on the match-action abstraction following the SDN principles to establish the input/output forwarding path between the shared and dedicated network functions as mentioned in [27]. Note that forwarding plane is introduced here to compose the input and output data stream for a flexible processing pipeline composition. To this end, the C-RAN will further evolve to provide a multi-service environment towards the service-oriented RAN (SO-RAN) architecture.

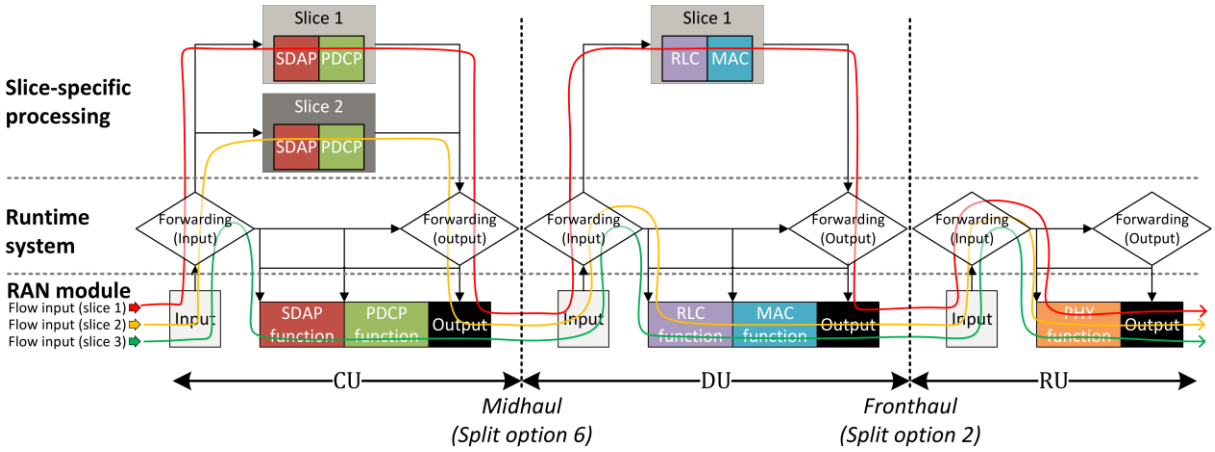


Figure 2: User data plane forwarding path in disaggregated RAN.

Contrary to the traditional networks, where BBUs and analogue front ends are co-located, C-RAN has been proposed as a progressive architecture with the notion of reducing the Capital Expenditure (CAPEX) and OPerational Expenditure (OPEX) as well as providing simple repair and maintenance, and easy system upgrades. In C-RAN there is a centralized processing, whereby multiple BBUs are placed at a single centralized location. This centralization helps to reduce power consumption and enables efficient use of hardware utilization with the help of resource sharing and network function virtualization.

Most of the communication functionalities in the BBU pool are implemented either fully or partially in a virtualized environment hosted over General Purpose Processors (GPPs) [29]. In this project, we are interested in understanding the computational effort involved in carrying out 'split baseband processing' by evaluating which baseband functionalities can be carried out with low cost GPPs and which require dedicated hardware such as ASICs. In addition, we aim to study the potential gains of pooling the hardware resources at a centralized location as opposed to the conventional approach of having dedicated hardware at each BS. The term split based processing refers to the notion of splitting signal processing functionalities between the remote unit and BBU, which lead to different functional splits.

It is very important to understand the CPU utilization of BBU to design efficient resource sharing and proper allocation of appropriate schemes [29]. The total processing time will depend on various signal processing functionalities such as modulation, demodulation, coding and decoding, FFT/IFFT, etc., and the computing resources are spent and dependent mainly on these functionalities.

The CPU utilization is related with throughput of the access link and throughput itself will depend for example on MCS scheme, number of PRBs used. In [29], the authors have shown the percentage of CPU utilization can be approximated as a linear increasing function of the DL throughput as

$$\text{CPU}\ [\%] = c_1 \cdot \emptyset + c_2,$$

where c_1 and c_2 are constants and $\emptyset$ is the throughput measured in Mbps. The values of these constants are calculated in [29] assuming full centralization. However, their values will be different for the different functional split being considered, and the choice of proper split depends on the specific requirement and use cases. Furthermore, the choice of a proper model such as the linear model as in [29] or the nonlinear model as in [31] should be thoroughly investigated and validated.

The authors in [31] have provided a nonlinear model to calculate the computational resource effort in giga operations per second (GOPS), $P_{u,t}$ required to serve UE u at time t

$$P_{u,t} = \left(\frac{30}{50}A_{u,t} + \frac{10}{50}A_{u,t}^2 + \frac{20}{50}\frac{M_{u,t}}{6}C_{u,t}L_{u,t}\right) \cdot R_{u,t},$$

where A is the number of used antennas at BS, M the modulation bits, C the code rate, L the number of spatial MIMO layers, and R the number of physical resource blocks (PRBs), each as allocated to UE u at time t.

However, the choice of constants, functional split and nonlinear dependency needs to be validated with the real software implementation. In [32], the authors have used more complex models for power modeling of the host CPU for both uplink and downlink, where they provided complexity in GOPS for different components such as digital pre-distortion, filtering, CPRI, OFDM, Frequency domain (FD) linear, FD non-linear, FEC. The results presented in [32] are reported in Figure 4. Moreover, due to centralized processing and coordination, the computational complexity involved in acquiring large size channel state information will increase significantly. In [30], the authors have proposed novel approaches in order to reduce the computational complexity in acquiring the channel state information.

Figure 3 shows an example plot of computational complexity dependency on downlink throughput with BS equipped with eight antennas serving different number of users. As can be seen from this figure, the computational complexity increases linearly with the downlink throughput.

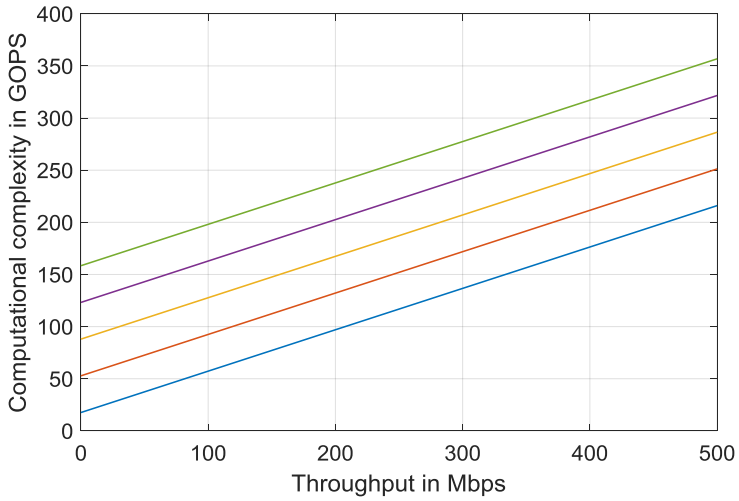
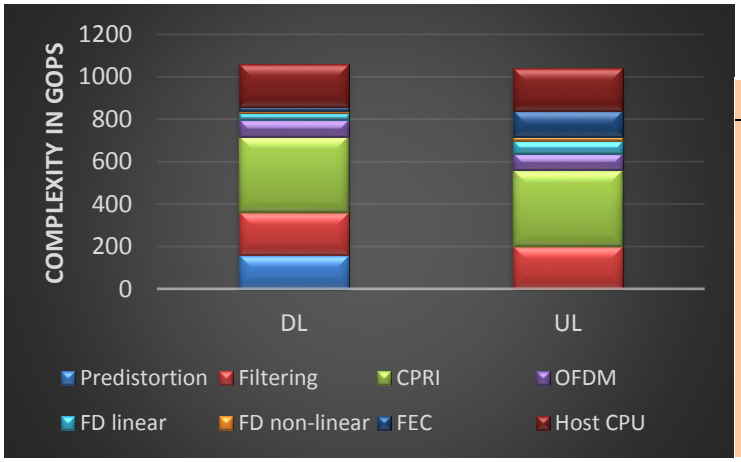


Figure 3: An exemplary plot of computational complexity in GOPS verses downlink throughput for different user certain time t.



Components	DL	UL
Predistortion	160	0
Filtering	200	200
CPRI	360	360
OFDM	80	80
FD linear	30	60
FD non-linear	10	20
FEC	20	120
Host CPU	200	200

Figure 4: An exemplary plot showing computational complexity in GOPS in UL and DL for different components of macro cell site.

2 Description of Selected Platforms

The development of programmable network platforms requires the use of specific HW platforms as technology target to actually implement and validate them. In this section we present the set of platforms that has been selected by the different partner for deploying the programmable platform. Some of these HW platforms are developed by the consortium members themselves and their functionalities are enhanced and/or exploited to improve the programmability of the programmable network platforms. Other HW platforms have been selected and acquired by the partners. Those platforms have been chosen not only to develop the partner specific programmability functionalities but also bearing in mind the integration (if applicable, depending on the use case they are intended to contribute to) of the work done by different partners. For example, most of the FPGA based development board will use the same FPGA manufacturer (Xilinx) and some of them will exploit the same Xilinx IP ecosystem (based on the AXI-4 protocol as internal interconnection bus among different hardware IPs). This can enable an easy integration between the specific hardware blocks developed by different partners.

2.1 VC709 Platform

The VC709 is a FPGA platform developed by Xilinx for high-bandwidth and high-performance applications. In 5G-PICTURE is used by University of Bristol to implement the TSON node described in section 3.1.

The VC709 FPGA card is capable of 40 Gb/s using Xilinx Virtex-7 FPGAs. The card can be installed in a test server into a PCIe slot to enable communication between the server and card. The VC709 has a FPGA Mezzanine Card (FMC) connector that allows to extend the capability of the bandwidth of the system. Figure 5 is a photo that shows the VC709 card with a FMC card having four 10G SFP+ ports. Therefore, the VC709 card in Figure 5 can increase the total bandwidth from 40G to 80G. Also, we can integrate the VC709 with a FMC card supporting 40G QSFP+ ports. In this case, the four 10G SFP+ data stream can be aggregated in 40G data stream in the FPGA. The aggregated data stream is forwarded into the 40G QSFP+ port on the FMC card. The programmable hardware platform allows developers to implement high performance network function in FPGA such as switch, router, and NIC.

The main features of the VC709 card are similar to an Open Source Hardware platform named NetFPGA-SUME. Therefore, the functions and libraries available from the Open Source Hardware platform can be ported into the VC709 platform. Main features of the VC709 are listed below:

- Virtex-7 VX690T FPGA.
- Four-port 10G SFP+ for 40 Gb/s high performance networking applications.
- Memory interface with 2x 4GB DDR3 SODIMM Memory.
- Enabling serial connectivity with SMA pairs, and UART.
- Expand I/O with FMC interface.
- PCI Express Gen3 x8 supporting 8 Gb/s/Lane.

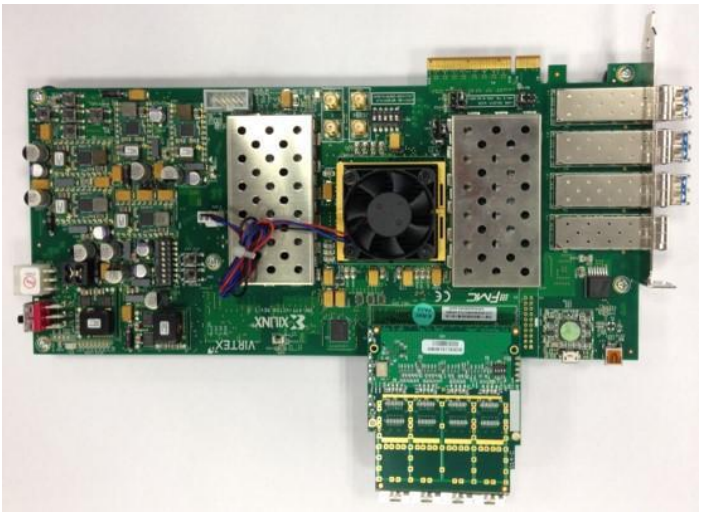


Figure 5: VC709 with 4x 10G SFP+ FMC card.

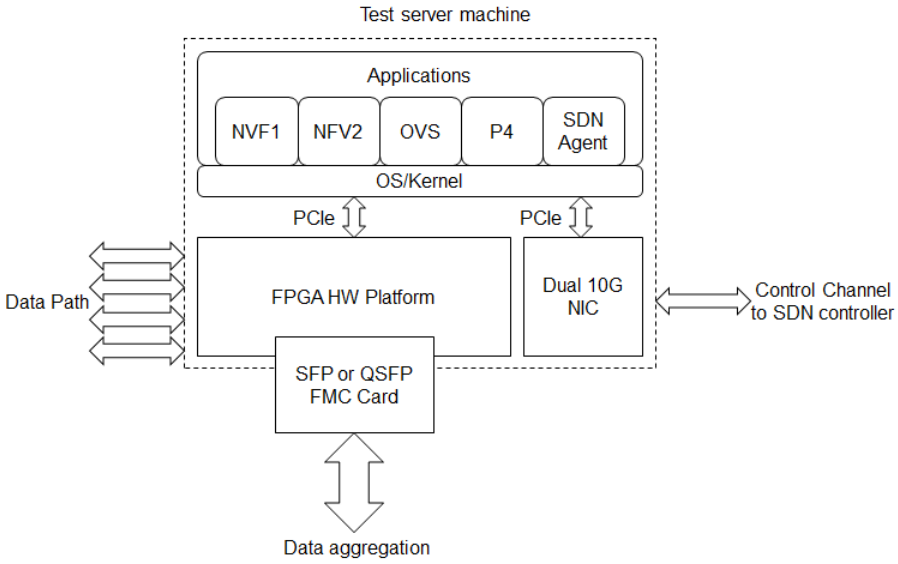


Figure 6: A block diagram of server test machine with the FPGA platform and the FMC card.

In Figure 6 a block diagram of a test server machine equipped with the VC709 FPGA platform incorporating the FMC card is illustrated. In the server, a dual-port 10G NIC card is also installed to connect a SDN controller to receive and send messages for the hardware configuration through the agent. The FPGA card in the test server machine can be programmed with a binary containing programmable hardware functions. The interaction between the FPGA card and the test server occurs through a register API as well as through a DMA engine, using a kernel-space driver. While the DMA engine is exposed in the server OS as a network interface that allows data-plane packet interception and injection, the register API is used to configure the hardware modules implemented in FPGAs, for example to install new rules and policies for routing and forwarding ingress packets.

2.2 NetFPGA-SUME

The NetFPGA SUME is an FPGA-based PCI Express board with I/O capabilities for 10 and 100 Gb/s operation, an x8 gen3 PCIe adapter card incorporating Xilinx's Virtex-7 690T FPGA. It can be used as NIC, multiport switch, firewall, test/measurement environment. CNIT is using the NetFPGA as target device for developing the prototypes of the OPP (Open Packet Processor) and PMP (Packet Manipulator Processor) components. The main characteristics of the device are listed below:

- FPGA Logic Xilinx Virtex-7 690T.
  - 693,120 logic cells.
  - 52,920 Kbit block RAM.
  - Up to 10,888 Kbit distributed RAM.
  - 30 GTH (up to 13.1Gb/s) Transceivers.
  - 4x10-Gigabit Ethernet networking ports.
  - Connector block to 4 external SFP+ ports.
- Memories
  - Three parallel banks of 72 MBit QDRII+ memories working at 500 MHz.
  - Two replaceable DDR3-SoDIMM modules, 933 MHz clock (1866 MT/s) for a total capacity of 8 GBytes.
- Third generation PCI Express interface, 8 Gb/s/lane, 8 lanes (x8).
- Expansion Interfaces:
  - Fully compliant VITA-57 FMC HPC connector.
  - Digilent Peripheral Module (PMOD) expansion connector.

A Detailed description of the NetFPGA SUME is available here at the product webpage[5].

2.3 Mellanox Spectrum™ Ethernet Switch

Mellanox will use their Mellanox Spectrum™ Ethernet Switch as target platform for the development of the P4 compiler for programmable dataplanes. The development of a P4 compiler for this platform will enable end-user to fully exploit the high programmability of such device with very limited design effort and without the need for Mellanox to expose the microarchitectural details of the ASIC switch chip.

The Mellanox Spectrum™ is a 10/25/40/50 and 100Gb/s Ethernet Switch solutions fully programmable and SDN-Optimized that enable efficient data centres fabrics. It provides the most efficient performing server and storage system Ethernet interconnect solution for Enterprise Data Centres, Cloud Computing, Web 2.0, Data Analytics, Deep Learning, High-Performance, and embedded environments. Spectrum, the eighth generation of switching IC family from Mellanox, delivers leading Ethernet performance, efficiency and throughput, low-latency and scalability and programmability for data centre Ethernet networks by integrating advanced net-working functionality for Ethernet fabrics. The main characteristics of the device are listed below:

- Industry leading, true cut through latency.
- Forwarding database sized for hyperscale.
- Optimized for SDN.
- Dynamically shared, flexible packet buffering.
- Flexible and programmable pipeline.
- Comprehensive overlay and tunneling support including VXLAN, NVGRE, Geneve and MPLS.
- Data Center Bridging (DCB) supporting PFC, DCBX, ETS protocols.
- Advanced load balancing.
- Advanced congestion management, Explicit Congestion Notification (ECN).
- Advanced PTP support.
- Flexible Port Configurations:
    - Up to 32 40/56/100GbE Ports.
    - Up to 64 10/20/25/50GbE Ports.

A detailed description of the Mellanox Spectrum™ Ethernet Switch is available at the product page[6].

2.4 Xilinx Zynq UltraScale+ MPSoC ZCU102 (OAI target)

University of Thessaly (UTH) is involved in building a new target platform for OAI towards optimizing its operation and allowing the implementation for achieving higher bandwidths than those currently supported. Although OAI is a pure software implementation relying only on GPPs for its operation, yet some computational intensive operations taking place in the physical layer (PHY) or higher (e.g. PDCP) of the platform may significantly consume more time, and thus prevent the platform from reach new levels of performance. Examples of such functions are the turbo decoding procedure in PHY, or the PDCP encryption/decryption at the PDCP layer. To this aim, UTH team is investigating several candidate solutions in order to maximize the performance of the platform. One of these is the integration of the Xilinx Zynq UltraScale+ MPSoC ZCU102 platform with the host platform running the OAI service, and the offloading of specific intensive tasks to the FPGA. In the following paragraphs, we detail the platform characteristics.

The Zynq UltraScale+ MPSoC device is equipped with a quad-core ARM Cortex-A53 and a dual-core Cortex-R5 real-time processors. A Mali-400 MP2 graphics processing unit (GPU) based on Xilinx's 16nm FinFET+ programmable logic fabric is also available. The ZCU102 supports all major peripherals and interfaces enabling development for a wide range of applications.

[5] https://reference.digilentinc.com/_media/sume:netfpga-sume_rm.pdf

[6] http://www.mellanox.com/related-docs/prod_silicon/PB_Spectrum_Switch.pdf

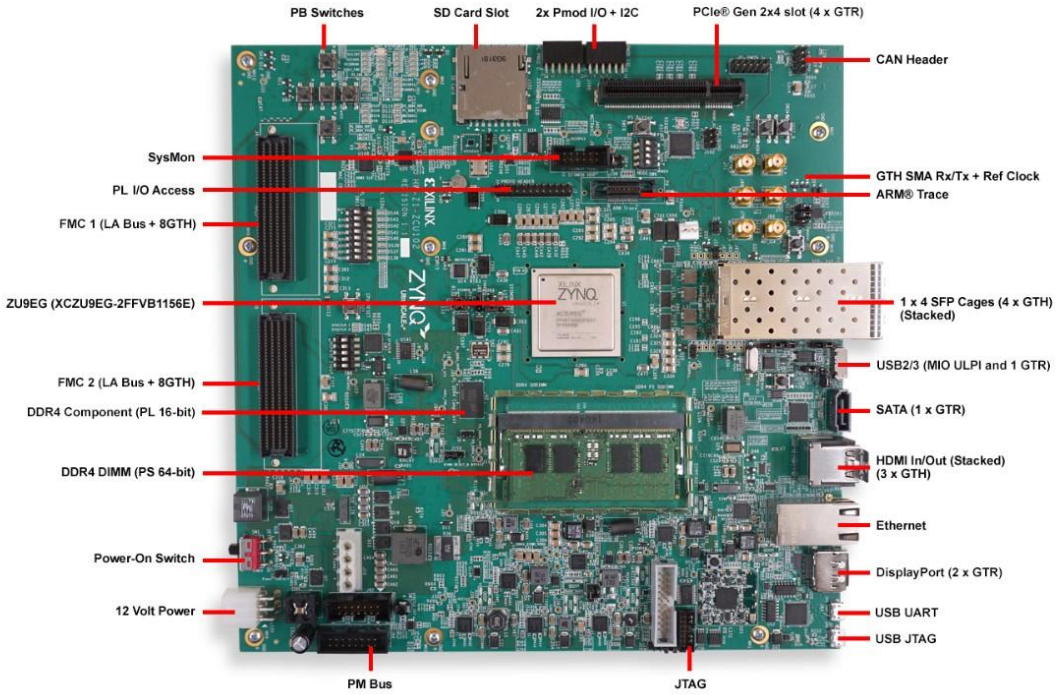


Figure 7: Xilinx Zynq ZCU102 interfaces.

Table 1 summarizes some of the key interfaces and features of the platform.

Table 1: Key characteristics of the Xilinx Zynq ZCU102.

Interfaces	Characteristics
Memory	• PS 4GB DDR4 64-bit SODIMM w/ ECC • PL 512MB DDR4 component memory ([256 Mb x 16] devices) at 1200MHz / 2400 Mb/s DDR • 8KB IIC EEPROM • Dual 64MB Quad SPI flash • SD card slot
Control and IO	• 6x Directional Push Buttons (5x PL, 1x PS) • DIP switches (8x PL) • PMBUS & System Controller MSP430 for power, clocks, and I2C bus switching • USB2/3 (MIO ULPI and 1 GTR)
Expansion Connectors	• 2x FMC-HPC connectors (16 GTH Transceivers, 64 differential user defined signals) • 2x PMOD headers • IIC
Communication & Networking	• RGMII communications at 10, 100, or 1000 Mb/s. Serial GMII interface-supports a 1 Gb/s SGMII interface • 4x SFP+ cage • SMA GTH support (4x SMA Tx/Rx connectors) • UART To USB bridge • RJ45 Ethernet connector • SATA (1 x GTR) • PCIe Gen2x4 Root Port

Clocks	• Programmable clocks • System clocks, user clocks, jitter attenuated clocks • 2x SMA MGT input clocks

2.5 Typhoon Platform

Blu Wireless Technology (BWT) will use its Typhoon Platform for data plane programmability investigations in WP3, for the investigation of virtualized synchronisation functions in WP4, as well as for the implementation of millimetre-wave (mmWave) testbed and demo setups in WP6, especially railway demonstrations. More specifically, BWT will use its DN101LC gigabit communication module, which is a member of the Typhoon family of highly programmable wireless communications modules for multi-gigabit prototype 5G infrastructure links.

The BWT Typhoon mmWave platform (made available in 2017) comprises two dual IEEE 802.11ad wireless mmWave modems interconnected by a network processor (NPU). It is based on the RWM6050 chip from IDT Systems Inc., which includes patented silicon IP from BWT. The referred chip, in turn, consists in a single-chip baseband solution for mmWave wireless infrastructure applications, which integrates a dual MAC, dual PHY and analogue baseband front-end functions. The Typhoon module allows evaluation of the RWM6050 and is able to deliver up to 4 Gb/s per link at ranges of 400 metres.

The Blu Wireless IP that is included within the Typhoon module consists in the patented HYDRA technology (**Hy**brid **D**efined **R**adio **A**rchitecture). A key feature of the HYDRA is that both the PHY and the MAC layer of its IEEE 802.11ad modem combine optimized hardware accelerators with programmable parallel processing. Namely both MAC and PHY are software-defined, which allows the performance of novel mmWave wireless algorithms to be explored and continuously tailored in the context of advanced research platforms. The HYDRA PHY DSP Modem is implemented as a SoC in 28nm LP CMOS and includes 2.6 Gs/s IQ ADC/DAC data convertors to support processing of 2 GHz wide radio channels as specified by the 802.11ad standard.

The Typhoon utilises the latest integrated electronic beam steering phased array antenna, RF and baseband technologies and is available in various configurations, including single, dual and quad RF, with or without NPU support and in either the default unlicensed 60GHz standard or other licensed bands. Figure 8 presents the block diagram specifically of the DN101LC module that will be used in 5G-PICTURE. The RF module consists in a 60 GHz radio transceiver with phased array antenna comprising separate active Rx and Tx antenna arrays with 12 elements each, gain of 16 dBi and Tx EIRP of over +26 dBmi. Note that the module can operate with an external host and a single mmWave radio-frequency integrated circuit (RFIC) device. An option to support larger antenna sizes to achieve increased gain is also possible.

Lastly, the DN101LC module features an integrated Cavium CN8130 NPU, where the standard Linux network stack is augmented with OpenFlow and Network Configuration Protocol (NETCONF) support. Furthermore, it allows remote software upgrades such as the introduction of customized APIs. This provides a suitable environment for experimentation and investigations on data plane programmability in 5G-PICTURE.

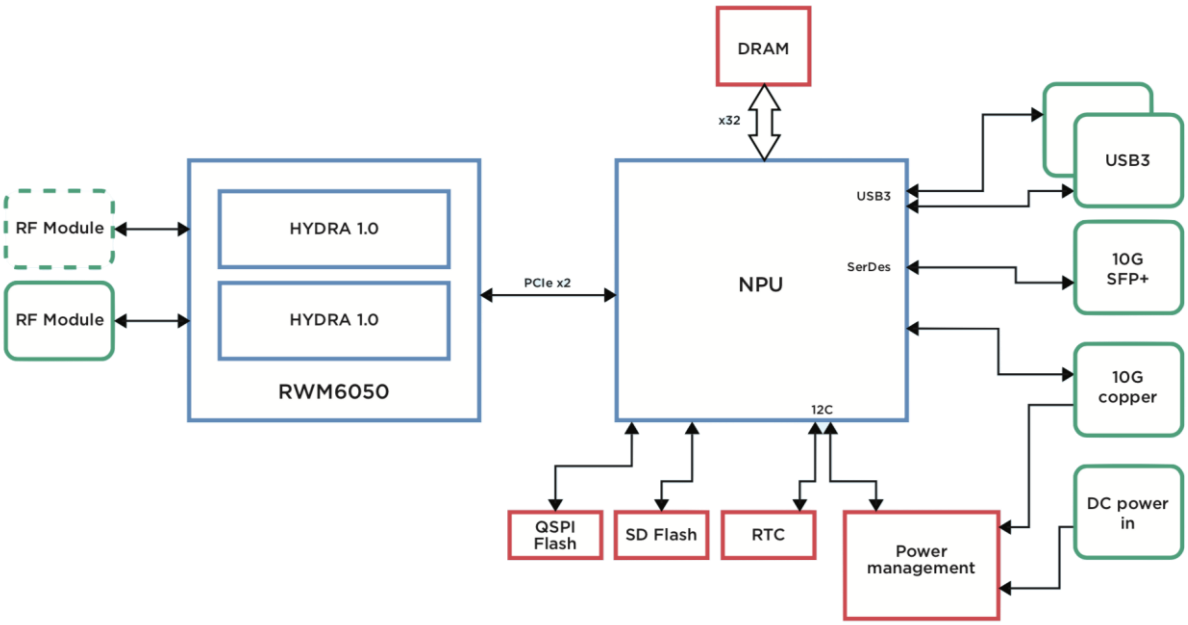


Figure 8: BWT Typhoon block diagram.

Figure 9: ADVA FSP 150 ProVMe series.

2.6 ProVMe

The FSP 150 ProVMe (Figure 9) is an existing ADVA product series and open platform for Virtual Network function (VNF) hosting and true multi-layer business service demarcation. Besides an integrated OpenStack[7]-based virtualization layer, it already offers full direct SDN-enabled configurability via standardized REST, NETCONF/YANG, and OpenFlow interfaces, and general Carrier Grade Ethernet Switch features like:

- Layer 2 service classification according to IEEE 802.1p, 802.1Q and IP-TOS/DSCP.
- Syncronisation services according to ITU-T G.8261/G.8262/G.8264 SyncE, G.8265.1/G.8275.1 PTP, and IEEE 1588v2.
- End-to-end data encryption and IEEE 802.1X authentication services.

In addition, it is well integrated with ADVA's FSP Network Manager Software[8], which can further act as standardized SDN mediation layer. Full ProVMe platform specifications are listed on its ADVA product page[9].

[7] https://www.openstack.org

[8] https://www.advaoptical.com/en/products/automated-network-management/fsp-network-manager

[9] https://www.advaoptical.com/en/products/packet-edge-and-aggregation/edge-computing/fsp-150-provme-series

ADVA's ProVMe development team is currently evaluating plans for adding P4-programmable FPGA resources to the platform for future product variants. Together with the already existing NFV and Carrier Grade Ethernet Switch features, and standardized SDN integration interfaces, this offers an opportunity to collaborate with the product development team for providing a completely open and fully featured white-box edge platform for the 5G-PICTURE network.

2.7 Gateworks Ventana

For implementing and testing 5G-PICTURE's Sub-5 GHz wireless transport node, i2CAT uses the Gateworks (GW) Ventana 5410 Single-Board Computer (SBC). The chosen platform shows the following main characteristics:

- FreescaleTM i.MX6 1GHz Quad Core ARM ® CortexTM-A9 Automotive Grade Processor.
- 1Gbyte DDR3-1066 SDRAM Memory and 256 Mbytes System Flash Memory.
- Six High-Power Gen 2 Mini-PCIe Sockets.
- Support for SIM Socket, mSATA Disk Drive, and USB Signaling.
- Two GbE Ethernet Ports.
- 8 to 60VDC Input Voltage Range.
- -40C to +85C Operating Temperature.

This ARMv7-based SBC is of particular interest because of the number of PCI-E slots, allowing the installation of up to six NICs. This brings the flexibility to instantiate multiple wireless access and/or backhaul interfaces with a single board. The SBC runs Linux (Ubuntu 16.4 LTS) with a 4.9.65 kernel that has been modified for better support of IEEE 802.11 features.

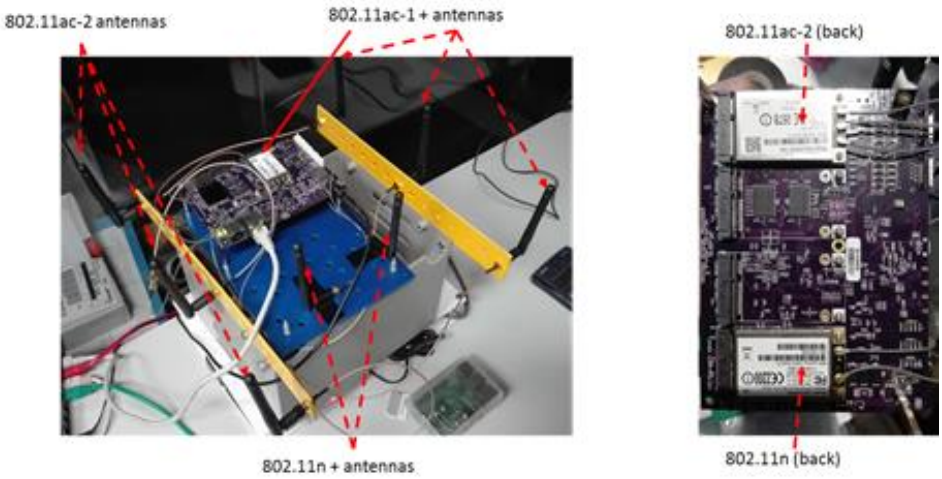


Figure 10: Gateworks Ventana platform equipped with 3 wireless NICs and omnidirectional antennas.

Figure 10 shows a prototype node for on-street or indoor deployment, featuring a GW Ventana 5410 equipped with 2x IEEE 802.11ac and 1x IEEE 802.11n NICs. In this example, the IEEE 802.11n is used to provide wireless access (Access Point mode), whereas the IEEE 802.11ac cards are used to create wireless backhaul links with other nodes thereby forming a mesh network. In outdoor deployments, the use of directive antennas for backhaul connections allows to cover wider ranges and to maintain a high throughput even with a large separation between nodes.

2.8 digiBackBoard

The digiBackBoard is a universal platform for high-data rate communication systems based on a high-performance FPGA-ARM-SoC, GS/s data converters and Gigabit Ethernet transceivers. IHP plans to use the digiBackBoard both as a wireless communication node and to provide a multi-PHY/multi-protocol interface among different technologies.

2.8.1 digiBackBoard as a wireless communications node

The digiBackBoard is especially suited for demonstration and evaluation of millimetre wave (mmWave) signal processing algorithms and for small-footprint prototypes. Nearly all kinds of Analogue Front-End (AFE) modules can be connected to the universal analogue IQ signal interface. A MATLAB-based software framework supports a fast start with a hardware-in-the-loop (HIL) setup.

Figure 11: IHP's digiBackBoard.

The main components and features of the digiBackBoard are:

- High-performance Xilinx Zynq-7045 FPGA (SoC) with embedded dual-core ARM processor
- Dual 2.16 GSps 10 bit ADC, dual 2.16 GSps 14 bit DAC
- Differential IQ-Interface for AFE connection
- Auxillary 180 MSps 8-bit ADC (for RSSI)
- 1 GB DDR3-RAM
- 4 Gigabit-Ethernet-Ports
- 76 digital GPIOs + 4 GTX transceivers externally available
- Serial (RS232) and JTAG interface through USB
- Prepared for support of SyncE and IEEE 1588v2
- AFE output power supply with configurable voltage
- Single 12 V power supply
- Boot from Micro-SD card or on-board SPI-Flash
- Matlab framework, GUI and firmware example design for SDR application available
- Ready-for-Linux (e.g. Peta-Linux)
- 100 x 160 mm footprint

IHP plans use the digiBackBoard for:

- Implement point-to-multipoint (P2MP) MAC processor (HW/SW codesign): A preliminary concept is already available and is included in Section 3.6 of this deliverable.
- Wireless synchronization, related as well to the work to be done in WP4.
- Upgrading of baseband to 802.11ad, having increased data rates for both OFDM and Single-Carrier (SC).
- Investigate the adoption of OpenCL abstractions to enhance the programmability of the digiBackBoard.

2.8.2 digiBackBoard as an interface across technologies

The digiBackBoard will be used to provide fast Multi-PHY and Multi-Protocol interfaces (MPIs) based on state of the art FPGA allowing great flexibility to mix and match a variety of protocols and technology solutions, e.g. PCIe 4.0, USB 3.1, Ethernet 100G, 40G, 10G, and 1G, SFI/XFI, SATA, Q/SGMII and CPRI.

The objective is two-fold:

- enable the support to heterogeneous networks
- enable mapping of traffic across infrastructure domains

As an example the digiBackBoard could be deployed at the edge nodes to support a variety of multi-protocol/PHY interfaces. This allow mapping very different traffic streams coming from the wireless access domain to optical frames/streams, capitalising on a number of HW programmable building blocks including OPP, BVTs and exploiting the benefits of FPGA based HW. Figure 12 shows the integration of OPP and BVTs (two different programmable blocks that will be described in section 3), which can be achieved by using the MPI.

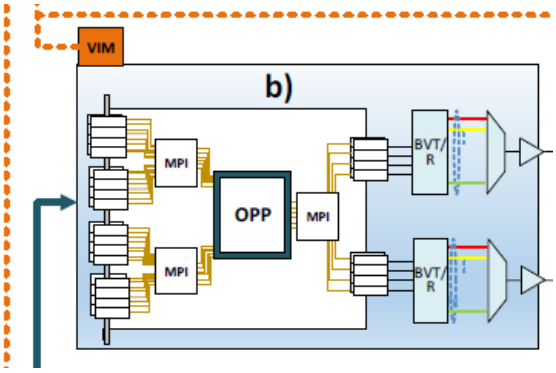


Figure 12: Possible use of the MPI to interconnect different programmable blocks.

2.9 Xilinx VCU-110

This section presents an overview of the VCU110 evaluation platform, shown in Figure 13, which is used as target for TransPacket's FUSION 100G IP Core R1.6. TransPacket plan to use this platform for evaluation of low latency scheduling, shaping and priority mechanisms for Ethernet Time Sensitive Networks (TSN).

2.9.1 Evaluation platform

The VCU110 evaluation board is documented in the Xilinx product page[10]. It is equipped with a Virtex Ultrascale FPGA and four CFP4 optical modules applicable for 100 Gb/s Ethernet. Furthermore, two FMC connectors can be found on the board, applicable for connecting FMC boards offering 10 Gb/s interfaces.

2.9.2 Faster technology SFP+ interface board FM-S18

Optical 10G Ethernet interfaces are provided by a FM-S18 Octal SFP/SFP+ transceiver FMC board (Figure 14). The FMC board is documented at the Faster Technology product page[11].

Since the VCU110 board only supports a subset of signals at a FMC high pin count connector, control of the SFP+ modules is not possible. The FM-S18 board is therefore modified by connecting *TX_DISABLE* to ground. Monitoring of *RX_LOS* from the SFP+ modules are not possible due to limited number of signals in the connector.

[10] https://www.xilinx.com/support/documentation-navigation/design-hubs/dh0046-vcu110-development-kit-hub.html

[11] http://www.fastertechnology.com/products/fmc/fm-s18.html

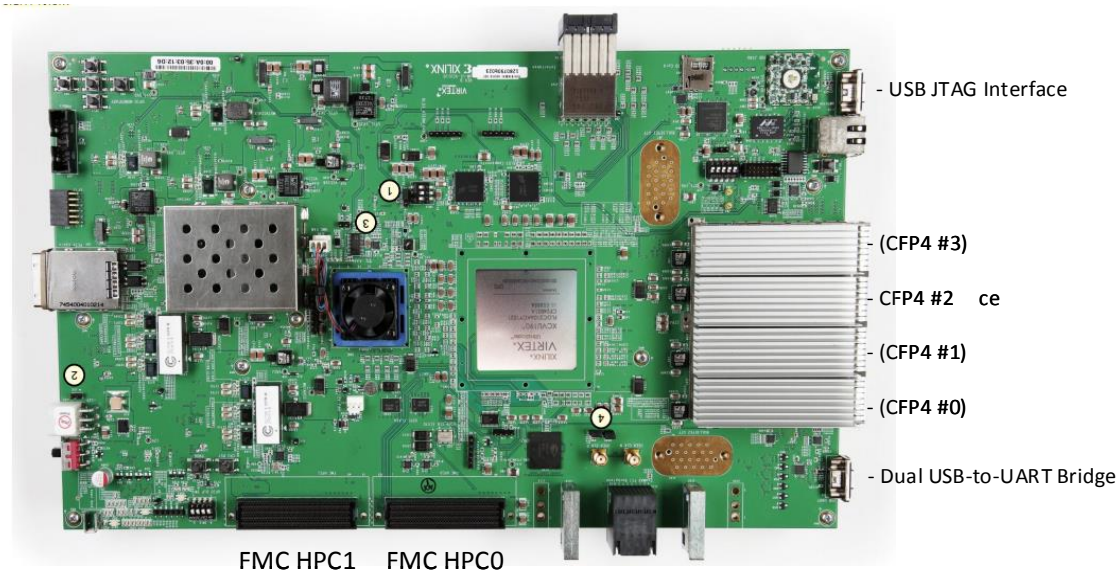


Figure 13: VCU110 prototype board.

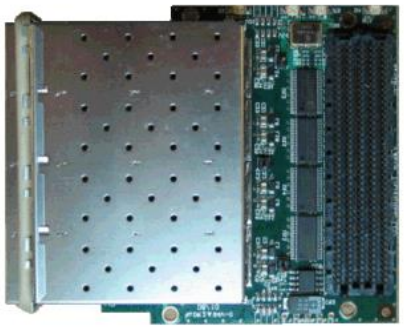


Figure 14: FM-S18.

2.9.3 Block diagram

The block diagram in Figure 15 shows the layout of the evaluation platform. Support for configuration/control of the 100G IP Cores is provided by MicroBlaze SW. Connections between VCU110 board and the Lab-PC is carried out by an individual USB-UART connection to the board.

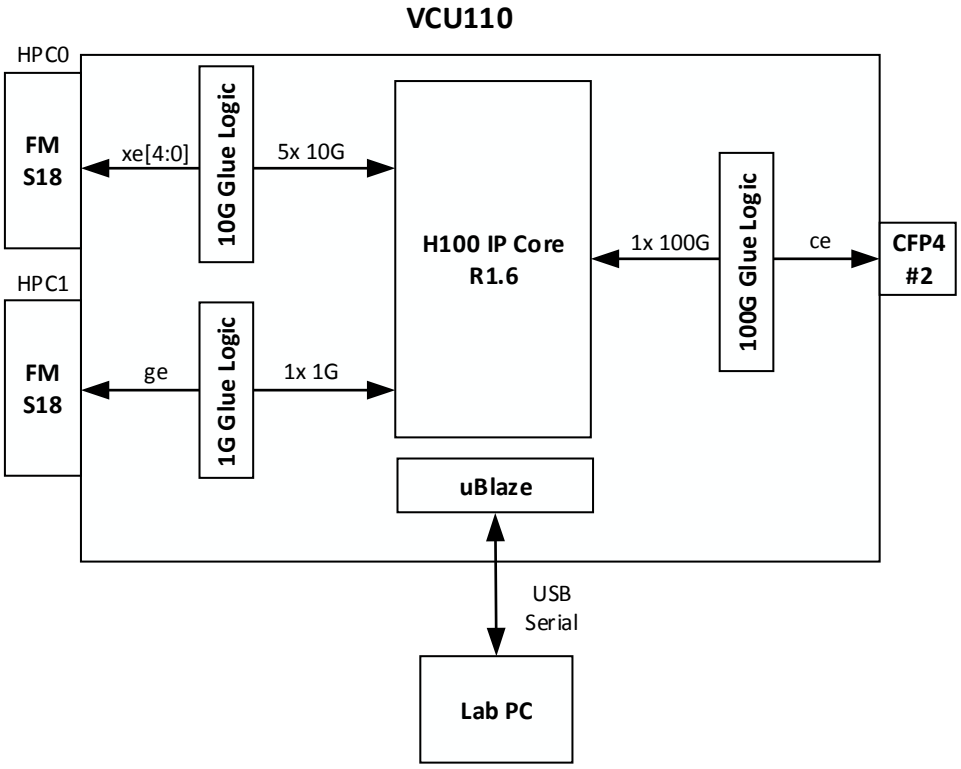


Figure 15: 100G IP Core evaluation platform.

2.10 IAF 5G Development Platform (F-PU 5G)

The F-PU 5G by IAF[12] GmbH (Institute for applied radio system technology) is an FPGA evaluation platform specifically targeted at SDN+NFV use cases. Just like the Xilinx VCU110 board, it is based on a Xilinx Virtex Ultrascale FPGA, but features extension slots for up to two 8-Core Intel Xeon processor modules.

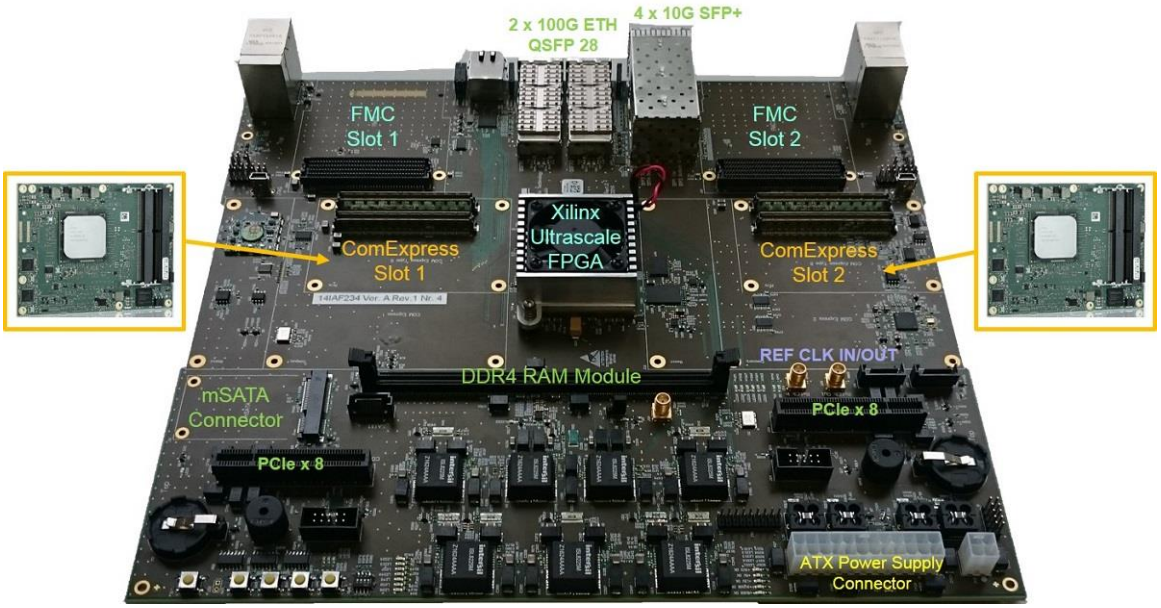


Figure 16: IAF 5G Development Platform.

Further details are given on the according Xilinx partner product page[13] and in the block diagram in Figure 17.

In addition to the use of the existing ADVA ProVMe product line, (described in section 2.6) as integrated VNF hosting and 5G-PICTURE network support solution and ADVA's already ongoing evaluation of the Xilinx VCU110 board (described in section 2.9) as pure FPGA VNF hosting platform, ADVA is considering the evaluation of the F-PU 5G as an alternative, standalone NFV platform. The advantage over pure FPGA platforms like VCU110 is the ability to flexibly distribute and combine VNFs across integrated FPGA and CPU processing resources. The integrated CPUs further avoid the need for an external PC running management software for user-programmability, VNF control and deployment, as well as SDN controller integration. Also, development of VNFs and management software components can be performed directly on the board.

[12] http://www.iaf-bs.de/en/

[13] https://www.xilinx.com/products/boards-and-kits/1-8dyf-1506.html

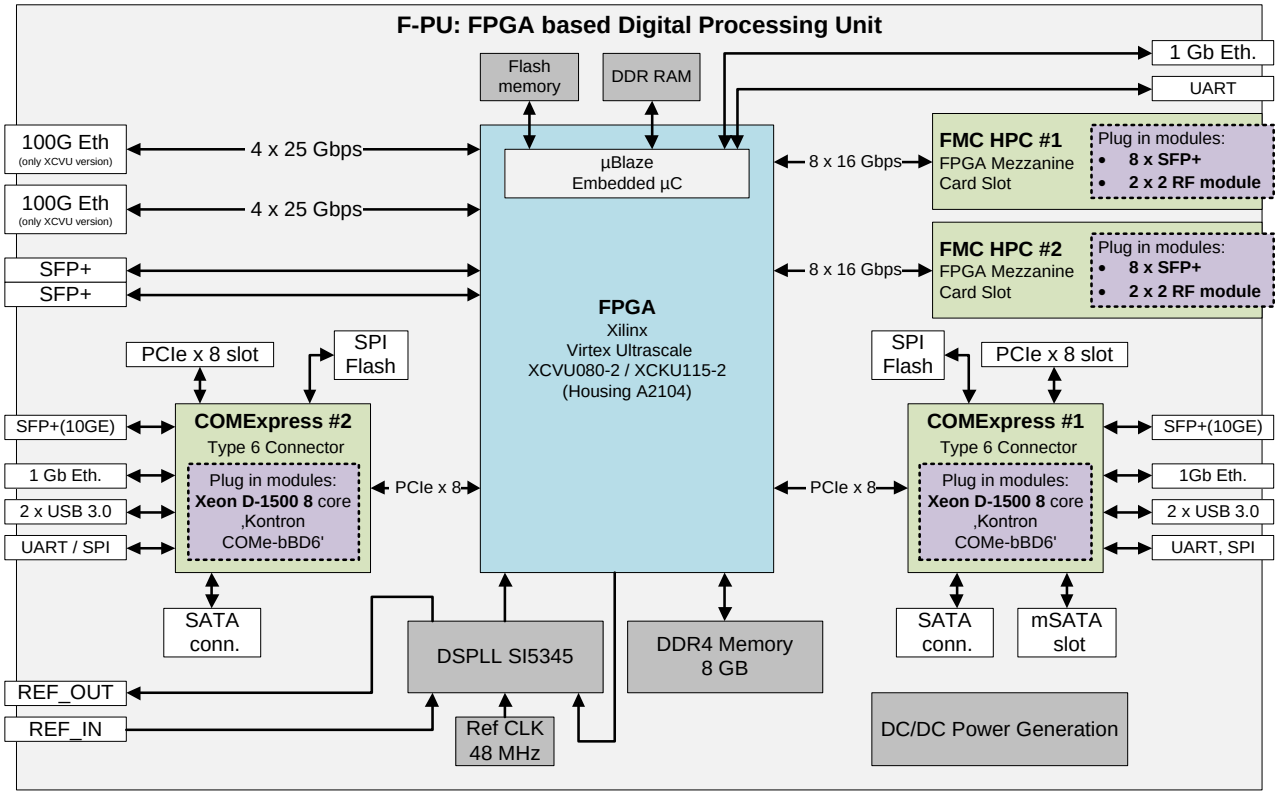


Figure 17: F-PU 5G block diagram.

3 Functional Definition of Programmable Platforms

This section reports the progress 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. These primitives are thus decoupled from the programmatic ways of combining them to provide those network functions. Different types of platforms will be used/developed during the project: i) for the BH, the platforms will provide a highly *programmable stateful dataplane* with latency and throughput performances able to sustain the need of the 5G network; ii) These platforms will also exploit the *integration of optical (Elastic and Passive)* network solutions; iii) In the FH, reconfigurable hardware 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 (BS).

3.1 SDN agent and controller development for control- and data-plane in Optical transport in support of joint FH/BH

The SDN architecture employs the concept of controllers and agents for network programmability. 5G networks adopt the SDN concepts in both FH and BH for programmability purposes. The SDN agent is responsible for carrying out the command of the controller and notifying the controller regarding events that are specified by the controller.

In 5G-PICTURE, the SDN agent is implemented at the RUs and BBUs to enable communication with the SDN controller. In the project, the TSON node supporting FH services is implemented on the VC709 FPGA platform that is developed by Xilinx. The platform has 4 10G SFP+ ports. As a prototype platform for the testbed, one of the 10G ports or the NIC card described in Figure 5 is used as a control channel to communicate with the SDN controller through the SDN agent. The 10G port used for the control channel can also be connected to a 1G channel through a media converter. The rest of the available ports on the platform are used to support data-plane functions controlled by the SDN controller. For example, the number of time slots on the data-plane for data transmission will be configured and controlled by the agent according to a message sent by the SDN controller. Also, the buffers inside the FPGA enable the assignment of timeslots to the output ports based on different matching fields such as: destination address, source address, and VLAN ID.

In 5G-PICTURE, the agent is implemented in C and Python which constructs devices configuration messages through a predefined Ethernet frame. The module implemented in FPGA parses incoming packets as a packet switch, which implies that it could forward traffic according to their ingress port, L2, L3 or other layer features. This module implemented in FPGA not only schedules packets in Time Division Multiplexing (TDM) fashion but also supports allocation of TDM slots on different wavelengths in a flexible manner. From a resource allocation perspective, a flow table (instead of a cross connection matrix) with flow entries including match field and its associated actions (e.g., forwarded in which timeslot) is maintained in control plane to indicate the existing flow forwarding rules and resource availability.

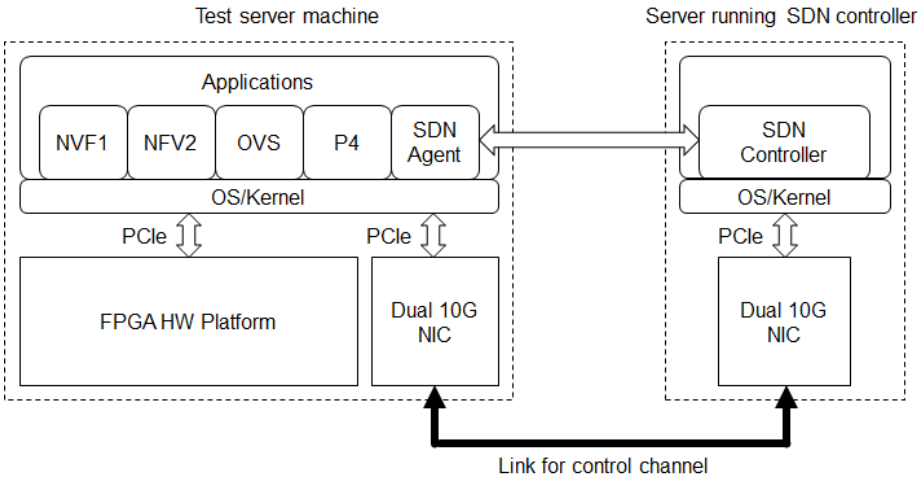


Figure 18: Configuration for the development of an agent and controller for SDN.

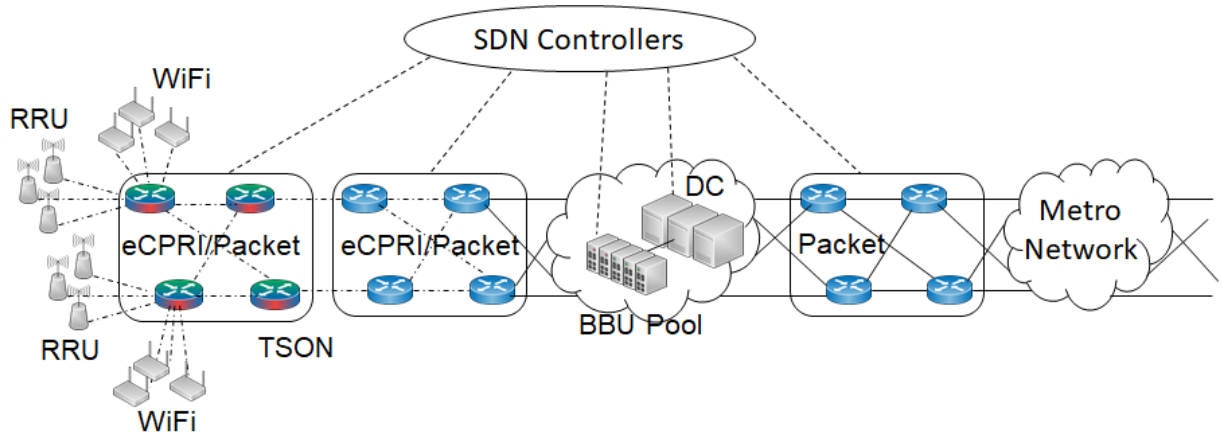


Figure 19: Network connection for SDN enabled RU and BBU.

3.2 Architectural definition of programmable C-RAN

5G wireless access solutions will support a heterogeneous connectivity of integrated air interfaces shown in Figure 19. These will coexist with legacy wireless access, LTE and Wi-Fi technologies, to allow broader coverage and availability, and higher network density and increased mobility. To further enhance spectral efficiency and throughput, small cells can be deployed, either adopting the traditional Distributed Radio Access Network (D-RAN) paradigm, where Baseband Units (BBUs) and radio units are co-located, or the concept of Cloud Radio Access Network (C-RAN). The remote units (RUs) in C-RAN are connected to the Central Unit (CU) – where the BBU pool is located – and are interconnected through high bandwidth transport links known as FH. Through its pooling and coordination gains, C-RAN can address the limitations of D-RAN, such as increased capital and operational costs, as well as limited scalability and flexibility. However, C-RAN (depending on the wireless technology adopted) may require tremendous transport bandwidth and impose strict latency and synchronisation constraints. Optical network solutions can play a key role offering the advanced transport capabilities. To limit the stringent transport requirements of the C-RAN approach alternative architectures have been proposed. These alternative architectures rely on concepts such as the flexible split options. The introduction of flexible splits allows dividing processing functions between the CU and the RUs. Through this approach, a set of processing functions can be performed at the RUs deploying local dedicated compute resources and the remaining functions can be performed centrally, through shared compute resources.

5G-PICTURE will design and develop a converged infrastructure for FH and BH, integrating a variety of advanced wireless access and radio network technologies for transport through novel optical network solutions. 5G-PICTURE will adopt hybrid optical network solutions deploying advanced passive and high capacity elastic optical networks exploiting the benefits of packet networks to address the limitations of the C-RAN approaches. In the solutions, a novel architecture exploiting flexible functional splits is designed and developed to evaluate not only configurability and programmability but also performance in terms of latency and data throughput. The optimal “split” in C-RAN will be flexibly decided, based on a number of factors such as the transport network and the service characteristics, yielding significant resource and energy efficiency benefits. The introduction of these splits allows to divide the processing functions between the CU and the remaining BBU processing functions, through shared compute resources.

The required flexibility can be provided in programmable digital hardware, such as FPGA, to support flexible reconfiguration of hardware-accelerated (HWA) and software-realized BBU functions, which can be partitioned at different levels. The shared “BBU pool of resources” required to support such activities mitigates the needs of owning hardware as it can be hosted either at publicly available micro Data Centers (mDCs) – referred to as Multi-access Edge Computing (MEC) – or at remote regional and central large-scale DCs. This alternative programmable C-RAN approach introduces the need to develop new technology solutions able to achieve increased performance and high levels of power efficiency, flexibility and density. 5G-PICTURE proposes the concept of “disaggregation of resources” as a key enabler in this direction.

Disaggregation relies on physically decoupling components and mounting them on remote locations, instead of tightly coupling all components in a conventional integrated system. Disaggregation facilitates independence across technologies and subsystems, offering increased granularity in the control of resources and the way they are allocated and provisioned. Apart from increased flexibility and programmability, due to its modular

approach, dissagregation offers enhanced scalability, upgradability and sustainability potential that are particularly relevant to 5G environments supporting enormous and continuously growing number of end-user devices and services. To exploit the concept of disaggregation in the programmable C-RAN environments, it needs i) hardware programmability: allowing HW repurposing to enable dynamic on demand sharing of resources and ii) network softwarisation: enabling migration from the traditional closed networking model, focusing on network entities, to an open reference platform instantiating a variety of network functions. Such novel networking approaches can facilitate increased functionality and flexibility infrastructures, offering simplified management and advanced capabilities including slicing and virtualisation that allow the disaggregated resource pool to be shared and accessed remotely.

To exploit the concept of disaggregation in the programmable C-RAN architecture, 5G-PICTURE also designs and develops the "Dis-Aggregated RAN" (DA-RAN) architecture deploying the FPGA platforms described in section 2. In 5G-PICTURE, DA-RAN will adopt the notion of "disaggregation" of HW and SW components across the wireless, optical and compute/storage domains in FH. In the DA-RAN architecture, resource disaggregation allows to decouple HW and SW components creating a common pool of resources that can be independently selected and allocated on demand. The HW and SW components described in the following sections form the basic set of building blocks that, in principle, can be independently combined to compose infrastructure services required.

3.3 Open Packet Processor (OPP)

CNIT is developing a programmable dataplane that will be able to execute several stateful network functions (NFs) without the intervention of the control plane. The OPP dataplane will be used inside the 5G-PICTURE project to enhance the programmability of the transport nodes as detailed in deliverable D4.1. The hardware constraints in terms of memory amount and number of operations that can be executed for each packet that must be processed by the network, pose severe limitations to the programmable dataplane architecture. These limitations are in contrast to the requirements in terms of flexibility of the NFs. Luckily, there is some recent research work showing that the main hardware elements composing the dataplane could provide enough flexibility and programmability to realize several network functionalities directly in the dataplane [33]. Recently, programmable dataplanes [35] emerged as ideal target devices to implement these complex NFs. These programmable dataplanes can be configured using specific programming languages (such as P4 or POF). However, the current scenario presents several limitations that prevent the nowadays-available programmable dataplanes to act directly for stateful functionalities. Here we recall the major limitations that will be addressed during this research activity in order to provide an efficient stateful network function (NF) engine that we named Open Packet Processor (OPP). The envisioned architecture of OPP is presented in Figure 20. This architecture is conceived to supersede the above mentioned limitations of programmable dataplane retaining the ability to sustain wire speed packet processing. Some of the elements of the architecture are slightly modified versions of the same elements used in the original OPP model [36], while others will be developed during this research project.

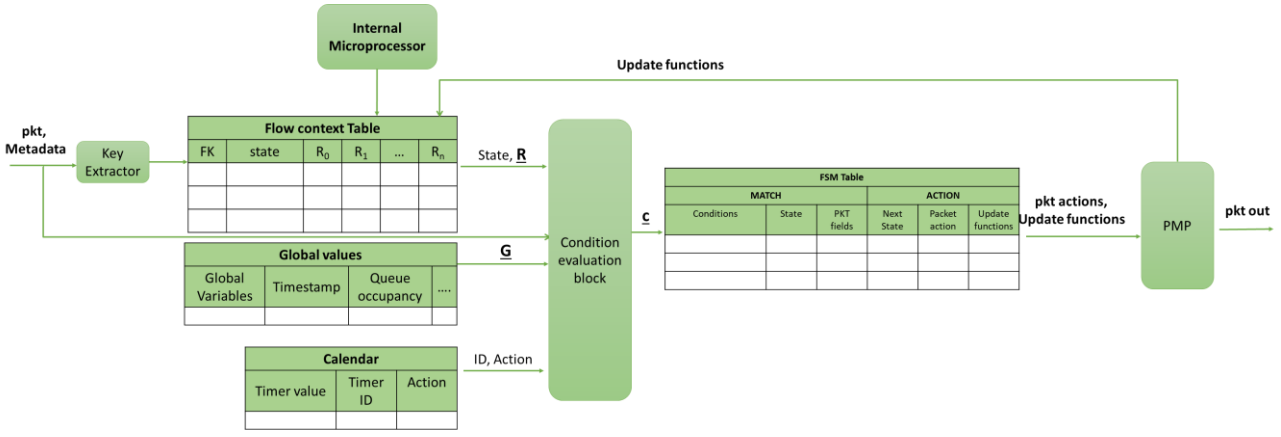


Figure 20: Architecture of OPP.

The blocks presented in Figure 20 performs the following functionalities:

- **Lookup extractor:** it extracts the packet header fields used to define a flow.

- **Flow context table:** this memory is used to store/retrieve the stateful context associated to each flow. The characteristics of the Flow context table has been extensively upgraded during this project and as reported in section 3.3.1.
- **Global values:** global variables represent the internal state of the network node. The global values can provide different type of information depending on the type of network node and of the specific implementation, (e.g. it could provide the queue occupancy of an input port, the link utilization but also the power consumption of the device if the hardware is able to provide this metric).
- **Condition Evaluation Block:** the per-flow registers (stored in the flow-context table) and global variables are evaluated by the Condition Evaluation Block. The results of the evaluation are given as input to the Finite State Machine (FSM) table that execute a FSM algorithm.
- **Internal Microprocessor:** this soft core processor is used to perform several tasks that we collect under the name of "aggregation tasks and lazy evaluation functions". These tasks are under development inside the 5G-PICTURE project and the details about these tasks are reported in section 3.3.2.
- **Calendar:** this block provides fine grained timer management. This block has been completely developed inside the 5G-PICTURE project activities and the details of the design are reported in section 3.3.3.
- **Packet Manipulator Processor (PMP):** the PMP is a Very Long Instruction word (VLIW) based processor tailored to perform packet manipulation tasks such as header encapsulation/decapsulation, CRC/checksum re-computations and other functionalities related to packet processing. The initial idea of the PMP together with a software simulator based on a Reduced Instruction Set Computer (RISC) architecture has been presented in [40]. CNIT is developing the hardware implementation of the PMP inside the WP3 activities of 5G-PICTURE. Details about this block are reported in section 3.3.4.

3.3.1 Per-flow stateful model

One of the bigger limitations in the current programmable dataplane is the absence of a clear per-flow stateful model for storing directly in the dataplane the information gathered on the different flows under analysis. There are two types of stateful elements in programmable dataplanes: tables and counters/registers. Nowadays tables can be controller only from the control plane (insertion/update/delete operation can be executed only using specific control-plane commands). Registers/counters array can be updated directly in the dataplane, but it is hard to map a row of the array to a specific flow. The currently available solutions use hash functions to provide the mapping between the flow and the array elements, but there is no automatic collision resolution when multiple flows hit the same array element. The mapping between the flow and the array elements can be also realized using a matching table, but the use of this approach prevent the dataplane update of the table (e.g. when a new flow arrive or a flow expire). Even if some workarounds to update tables can be realized (see e.g. [33]), they are quite complex to realize, requires a significant amount of resources and are focused on a specific application. We solve these issues providing some specific tables (that we will call *flow-context table* in Figure 20) that can be updatable directly in the dataplane. The block called *lookup extractor* in Figure 20 extract the flow-key using a programmable subset the header fields coming from the packets. This flow key is used to retrieve from the table the state and the registers associated each flow. The preliminary hardware feasibility of these tables has been discussed in [36]. The improvements with respect the preliminary proof-of-concept are discussed below.

Large Flow Context Tables: The efficiency of the hash tables developed in [36] was limited by the use of d-left hash tables for hash collision resolution. Even if this collision resolution strategy is easily implementable in hardware it can provide a maximum load factor before failing that is in the range 25-50% depending on the specific configuration (number of tables and number of cell in each bucket). In order to increase the maximum load factor we developed a pure hardware cuckoo hash tables [37], which support higher load factors, therefore improving on SRAM usage efficiency. However, an entry insertion in a cuckoo hash table may actually need multiple operations when there is a hash collision. This makes insertion times variable and potentially long for a loaded table, severely impacting performance. While current designs usually perform entry insertion and collision handling in the device's control plane [1], we need to handle such complex logic in the data plane, while guaranteeing quick and constant insertion times. Second, a flow-oriented memory addressing implies that a given memory location is accessed only when a given flow's packet is processed. Therefore, a given memory location is accessed at a rate that may be just a fraction of the overall packet processing rate (i.e., one access per cycle). This could enable read/modify/write operations that span multiple clock cycles, therefore increasing device's operations potential complexity and flexibility. For example, one could implement more complex ALU's operations or, as in our case, implement an FSM execution logic. However, this introduces a

state consistency problem, since a memory location access times may vary depending on the traffic pattern, potentially leading to a concurrent read/write of the same memory location.

The Flow Context Tables store the state for the processed flows, i.e., state label and registers, and are in principle similar to regular forwarding tables. However, unlike forwarding tables, the OPP uses the Flow Context Table as a transactional memory, which is challenging because of the need to handle entry insertion at line rate, even in presence of hash collisions. OPP solves the issue implementing the insertion logic completely in hardware, and extending the hash table with a small stash memory to hold entries waiting for insertion. The stash allows the OPP to hide the variable insertion time of a cuckoo hash, when properly dimensioned. In particular, we implement a four-choices cuckoo hash table [38] that offers a 97% load factor, using a dual port (read-after-write) SRAM to support two memory accesses in the same clock cycle, for concurrent read and update operations. The table is coupled with a stash memory that can host 8 entries. Differently from a typical cuckoo with stash, a new entry is always first inserted in the stash, which guarantees constant insertion time (1 cycle). In parallel, the insertion logic moves entries from the stash to the hash table and operates as a regular cuckoo+stash implementation. The insertion logic can execute an entry insertion/movement per cycle, i.e., in about 6.4 ns in our implementation. Movements are required in case of hash collisions, since in such case the cuckoo algorithm inserts a new entry in the occupied position and moves the old entry in a new position. For a very loaded table the number of movements may grow 100x [38]. This rises a concern the stash may become quickly full. However, in typical workloads insertion operations are relatively uncommon. In fact, an insertion happens when a new network flow starts, therefore the insertion logic has to operate at the flow arrival time scale, instead of the packet arrival time scale.

Handling full context tables: Despite the optimization in place, a Flow Context Table (and the stash) may become full and, unfortunately, there is no universal strategy to handle such an event. For instance, consider the example of a stateful firewall. One strategy could be the rejection of any new connection. This would guarantee that ongoing connections are not interrupted. However, assume the firewall also handles high priority connections. Such connections may be dropped before establishment, causing e.g., a service level agreement violation. In this last case, we may want to insert a new entry, evicting an already existing one. OPP exposes to the programmer a flow context table full notification, in the form of a flag that further extends the EFSM table's entries match.

Also, the insertion of a new flow context entry always happens explicitly, associating the corresponding operation to an EFSM table's entry. Therefore, an FSM describing a network function can also describe the logic to handle Flow Context Table full events. Furthermore, to configure the eviction strategy, the insertion logic can be configured to read the entries' flow register R0 and to select for eviction the entry with the highest (or lowest) value. That is, a NF's FSM can use R0 to enforce a custom eviction logic. For example, R0 can be used to store a last seen packet's timestamp, to implement a logic that evicts the less active flows; or R0 could store a packet counter to evict the smallest (biggest) flow.

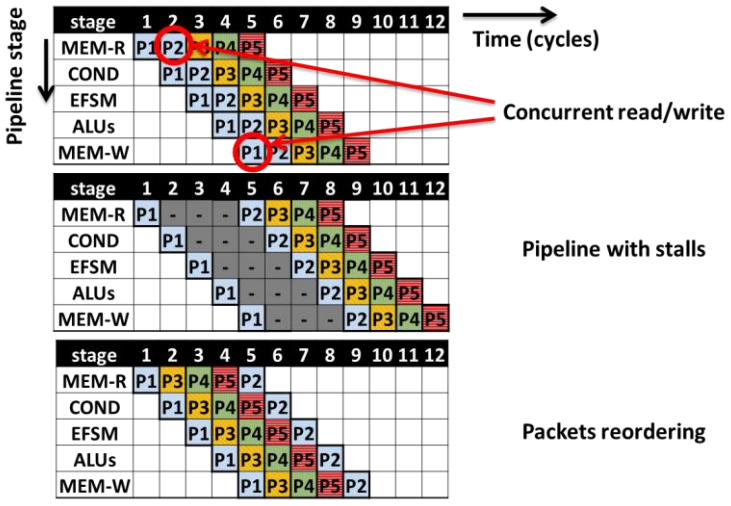


Figure 21: Stateful element scheduling options. P1 and P2 belong to the same flow therefore they use the same flow context. P3, P4 and P5 belong to different flows and can concurrently access memory.

Guaranteeing consistency: Like other data plane designs, an OPP's stateful element is a pipeline by itself, composed by multiple stages. However, differently from the design of [35] the state read, modification, and write feedback loop uses multiple clock cycles, introducing potential consistency issues. More specifically, when two packet headers that access the same flow context entry are processed in short sequence, e.g., back-to-back, the second packet's flow state read happens before the first packet's flow state update has been written back to memory (cf. first case of Figure 21). Stalling the pipeline while waiting for the state to be updated would guarantee consistency at the cost of performance (cf. second case of Figure 21). In fact, [35] uses read/modify/write state operations that are performed in a single clock cycle, to provide both consistency and performance, at the cost of constrained update operations. OPP, instead, leverages the parallelism given by the presence of different flows, which access different flow context entries, hence, memory areas. In particular, the scheduler of Figure 21 guarantees flow context consistency by conservatively locking the pipeline only when two packets may need to access the same flow context. Otherwise, packets belonging to different flows can be processed by the pipeline with no harm for consistency (cf. third case of Figure 21). The scheduler recognizes the flow a packet belongs to by using one of the hash keys (FK1) generated by the key extractor. The key extractor is configured to generate FK1 selecting a (programmable) subset of the packet's header fields, which are then used as input to the hash function. When a new packet arrives, the scheduler feeds it to the pipeline if no other packets with the same hash key are being processed. Otherwise, the scheduler stalls the pipeline. Since this mode of operations introduces a head-of line blocking issue, our general design mitigates the problem including multiple waiting queues for packets belonging to different flows. In effect, the scheduling block uses FK1 to assign packets to Q different queues, guaranteeing that packets belonging to the same flow are always enqueued in the same queue. The scheduler is work-conserving and serves the queues in a round-robin fashion. When a queue is selected, the scheduler verifies if a packet with the same hash FK1 is already in the pipeline. If that is the case, the scheduler examines the next queue, otherwise it extracts the queue's first packet and feeds it to the pipeline. The queues do not completely solve the head-of-line blocking issue. In fact, the number and length of the queues are characterizing parameters for the system's achievable forwarding latency and throughput. We study in [39] the related trade-offs. It's worth noticing that the scheduler re-orders the packets as they are being forwarded. However, packets belonging to the same flow keep their relative order.

3.3.2 Aggregation tasks and lazy evaluation functions

Most of the operations performed by network applications can be divided in two types. The first type is per-packet update that must be performed at wire speed and are usually composed by simple arithmetic/logic operations (counters update, comparison, etc.) that are used to update the state of the flow corresponding to the packet under analysis. The second type of operations usually perform aggregation and global analysis on the set of flows that are under measurement. Examples of this type of operations are the search for the maximum values among the active flows for heavy hitter detection or the entropy computation for Distributed Denial of Service (DDOS) detection. These operations can be performed with a time scale that is slower than the one of the per-packet operations. On the other hand, the complexity required for performing these operations can be higher and can require to read/update the whole stateful memory (the flow-content table).

We can also identify some per-flow operations that do not require to be updated for each packet, but that can be performed lazily and provide approximate results. Evaluating the average packet size of a flow is a perfect example of this lazy functionality. Performing the division between the number of received packets and the overall amount of transmitted bytes require considerable HW resources (that corresponds to integrate an HW divider in the datapath) and is probably an overshooting target. Instead, it should be sufficient to store for each flow both the packet and byte counters and performing the operation in a more relaxed timescale. The block called **Internal Microprocessor** in Figure 20 will be responsible of the execution of these tasks. The use of an internal microprocessor that has direct access to the flow-content table will enable the execution of a wide range of functionalities that can be used by the network applications.

3.3.3 Calendar

The FSM execution is usually triggered by the arrival of a packet, but we foresee that a flexible NF engine should be also provide a method to schedule events that can occur after a specific time interval (or at specific time). An example of the use of timers in a monitoring application is the tracking of Transmission Control Protocol (TCP) connections, where it is useful to check if the receiver of a TCP packet sends back the ACK of the received data in a certain time window. A specific action (called *create_timer*) of the FSM instructs the *calendar block* in Fig. 1 to schedule a timer. The timer will have some parameters like the timer ID that identify which type of timer has been scheduled and which action should be executed on when the timer expires. The calendar block is in charge to manage the insertion of timers coming from the FSM execution and to check the

expiration of the scheduled timers. When a timer expires, the calendar can trigger an event that is processed as a sort of “virtual packet” by the other elements of OPP.

3.3.4 Packet Manipulator Processor

PMP, the Packet Manipulator Processor is based on a VLIW processor with a custom Instruction Set Architecture designed to perform very efficiently the operations required by packet processing. In particular, the interface between the switch memory, where packets are stored, and output queues has been specifically designed, using extensively multi-port memories to efficiently accomplish packet manipulation tasks. A memory designed to handle data from an external IP parser is also included in the system for a better integration. The reason behind a VLIW processor, able to execute more than one instruction in parallel, is justified by the fact that a single-core RISC processor, although very simple and tunable for our needs, is not able to sustain high throughput, as discussed in [40]. In particular, the fact that a RISC processor has a 32 bits’ memory alignment, while packets are 8 bits aligned, makes this option not feasible since many clock cycles will be wasted on re-aligning the memory. The proposed architecture differs from a many-core CPU in many aspects. In particular, it occupies less area since data structures, such as register files, are shared between the execution units and does not require any extra hardware to synchronize the cores and to verify the parallelizability of instructions. In addition, a VLIW approach avoids the need of reordering packets, since in a many-core architecture different packets are scheduled on different cores and maybe committed to output queues out of order. The complexity of proper instruction scheduling, in a VLIW architecture, is demanded to the compiler. Since the routines involved in packet manipulations are very simple, such a compiler maybe kept basic.

PMP Architecture: A VLIW CPU architecture is able to execute multiple instructions in parallel through the use of *n* syllables, each long *m* bits, concatenated to form a unique instruction. All syllables are executed in parallel by a dedicated lane of the CPU which is composed by a fetch stage (IF), a decode stage (ID) and an execute stage (IE). In the particular case of PMP, the width of the instruction is 256 bits, allowing the simultaneous execution of 8 syllables on 8 different lanes. The top-view architecture of PMP is depicted in Figure 22.

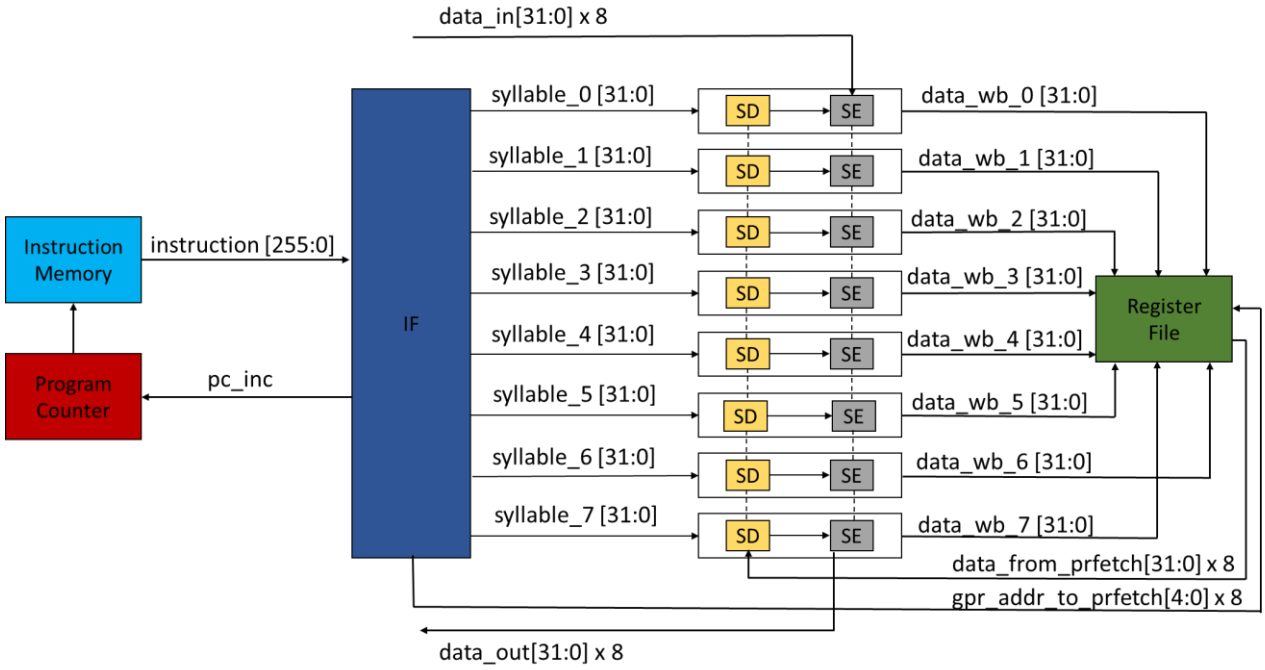


Figure 22: PMP top architecture.

Taking as reference a standard CPU (SCPU), such as the MIP SCPU used in [40], this architecture can provide a theoretical 8x instruction throughput[14].

[14] The actual improvement can be less than 8x, since the 8 syllables can be executed in parallel only if they are mutually independent. If there are less than 8 independent syllables to execute, the VLIW instruction is filled by NOP operations, and the actual improvement of the VLIW architecture with respect to a standard architecture decreases.

The PMP architecture differs from a standard VLIW processor for several design choices that we identified as mandatory in order to obtain high throughput in the specific task of packet manipulation, such as:

- Memory & registers prefect.
- Multiple memory units.
- Short pipeline.

The need of memory and internal registers prefect arises from the high memory pressure required in header rewriting and data movement. Since requesting data from memory and registers may take several clock cycles, data is queried to these units in advance, in order to have the requested data ready when needed in the syllable execution phase. In particular, for the operations involving operands stored in the register file, operands are queried already in the fetch stage, directly latched in the pipeline register of the decode stage and finally provided to the execute stage without stalls. In the same fashion, for load operations from memory the data is queried from memory already in the decode stage. For the same reason, PMP allows 8 memory units to execute operation on the memory in parallel, allowing 256 bits per clock cycle to be moved from/to memory. As in almost all the pipelined CPU architectures, the management of branches can significantly affect the performance of the PMP. In fact, in theory the next instruction to be executed is known only at the end of the execute stage, thus if a branch is executed, the instructions that are in the fetch and decode stages could be invalid, resulting in a misprediction. Although stalling the pipeline and wait for the branch instruction to give the result can be a tentative solution, we implemented speculative execution, meaning that every time the PMP encounters a branch it assumes that the branch is not taken, starting executing instruction right after the branch instruction. The misprediction costs a number of clock cycles proportional to the depth of the pipeline. For this reason, PMP implements a very short pipeline, having the branch misprediction penalty quantifiable in 3 clock cycles. Every stage takes exactly one clock cycle to be executed.

The use of pipelining registers between the fetch, decode and execute stages allows the execution of an entire instruction (8 syllables) per clock cycle. Another issue arising from the use of a pipelined architecture is the fact that consecutive dependent instructions will encounter a data hazard: if an instruction in decode stage depends on the results of another instruction that is in the execute stage, it will get the old value of the data stored in the registers. In fact, the new one will be written on the register file only on the next clock cycle. In order to overcome this issue, lane forwarding has been implemented. In this way, when a syllable is in the execute stage, it sends its result not only to the register file, but also to the decode stage. This allows to the instruction that is in decode stage to have always the updated value. The details of each lane are depicted in Figure 23.

The control unit is responsible for handling branches, and is present only on the first lane, since we cannot have multiple branches in a single instruction. The control unit operates on special registers, called branch registers, containing results of compare operations.

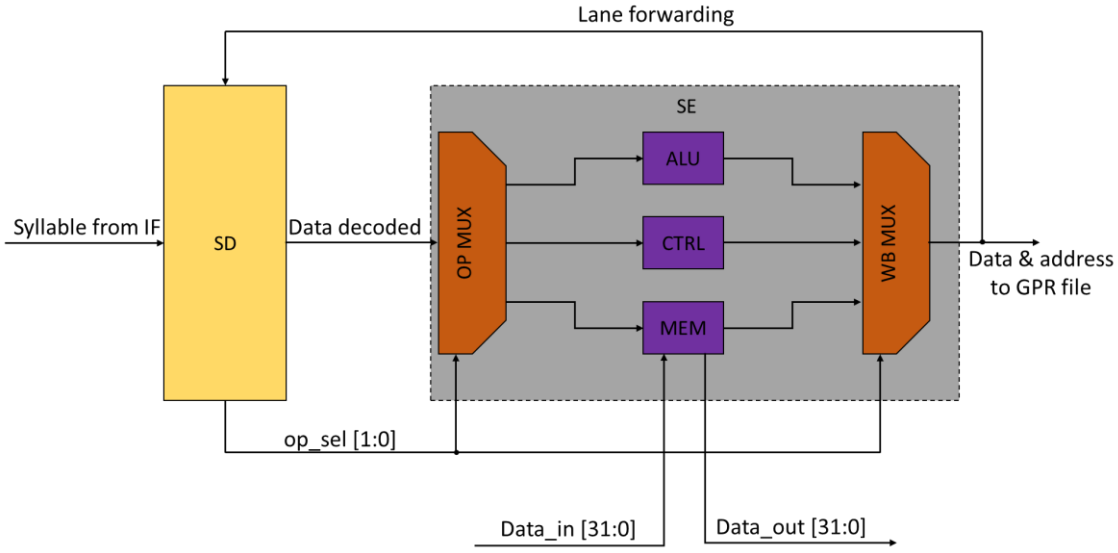


Figure 23: PMP Single lane structure.

3.4 OAI platform for SDN-based programmable network functions

OAI [45] is an open open-source software-based implementation of the LTE/LTE-A system spanning the full protocol stack of 3GPP standard both in RAN and core network domains. Such platform enables innovation in the area of mobile/wireless networking and communications. With OAI, the transceiver functionality (of a BS, access point, mobile terminal, core network, etc.) is realized via a software radio front end connected to a host computer for processing.

As the RAN is the most complex part of the mobile network infrastructure, it offers many opportunities to benefit from the SDN principles. The OAI platform is upgraded to support the SD-RAN platform. For this purpose, we implemented the FlexRAN [20] as an SD-RAN platform extending from OAI, which provides separation of the control plane and data plane through a new custom southbound API. FlexRAN provides a flexible control plane designed with support for real-time RAN control applications, flexibility to realize various degrees of coordination among RAN infrastructure entities, and programmability to adapt control over the time.

The FlexRAN platform is made up of two main components: the **FlexRAN Service and Control Plane**, and **FlexRAN Application Plane**. The former follows a hierarchical design and is composed of a Real-time Controller (RTC) that is connected to a number of underlying RAN runtime, one for each RAN module (e.g., one for monolithic BS, or multiple for a disaggregated RAN). The RAN runtime provides a flexible execution environment to run multiple virtualized RAN instances, monolithic or disaggregated, allowing them to monitor and control the underlying RAN with the required level of isolation and sharing. The control and data plane separation is provided by RAN runtime environment which acts as an abstraction layer with the underlying RAN module on one side and RTC and control apps on the other side. The FlexRAN protocol facilitates the communication between the RTC and the RAN runtime at each RAN module. RAN control applications can be developed both on the top of the RAN runtime and RTC Software Development Kit (SDK) allowing to monitor, control and coordinate the state of RAN infrastructure. Such applications could vary from soft real-time applications including monitoring that obtain statistics reporting to more sophisticated distributed applications that modify the state of the RAN in its runtime phase. All the produced Edge data and APIs are open to be consumed by third parties as a second level north-bound APIs exposed by the network control apps.

Figure 24 shows the FlexRAN architecture that is composed of two entities, where RTC controls a small network area (order of 100 BS) and is in charge of time-critical operation on a small time scale (order of ms or tens of ms), and can be connected to a number of RAN runtime, one for each data plane, and RAN runtime acting as an execution environment with a local controller, providing a limited network view, handling control delegation by the RTC or in coordination with other RAN runtime and/or RTCs. Note that the RTC itself can be connected to a more centralized controller for a large network area (on the order of 1000 BS) with a larger time-scale (order of hundreds of ms).

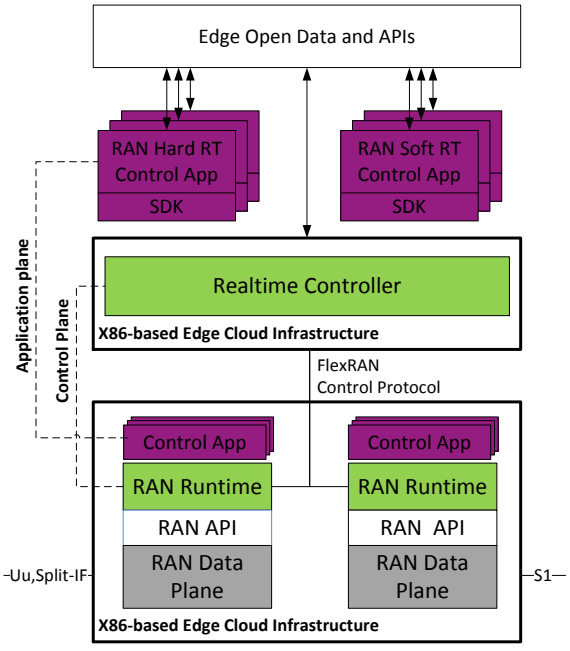


Figure 24: OpenAirInterface and FlexRAN platforms to support SD-RAN.

To control and manage the BS data plane actions, the RAN runtime API is introduced to provide a set of functions that constitute the southbound APIs. These functions allow the control plane to interact with the targeted data plane in five ways:

- Get and set configurations like the uplink/downlink bandwidth of a cell;
- Request and obtain statistics like transmission queue sizes of UEs and signal-to-interference and noise ratio (SINR) measurements of cells;
- Issue commands to apply control decisions (e.g., calls for applying MAC scheduling decisions, performing handovers, activating secondary component carriers);
- Obtain event notifications like UE attachments and random access attempts;
- Perform a dynamic placement of control functions to the RTC or the RAN runtime (e.g., centralized scheduling at the RTC or local scheduling at each RAN runtime).

These API calls can be invoked either by the RTC through the FlexRAN control protocol or directly from the RAN runtime if control for some operation has been delegated to it. Table 2 provides a list of some exemplary RAN-specific API calls. Note that the RAN runtime is in charge of retrieving the cell and user related information from the underlying RAN such as cell bandwidth, and user Reference Signal Receive Power (RSRP) and Reference Signal Receive Quality (RSRQ) through the API calls, and can trigger events when a state changes, e.g. user attachment and TTI (frame and subframe). In addition, such API calls may be related to the network functions, resources, users, etc. belonging to a particular slice. It can be seen that different types of network applications can be developed ranging from monitoring for a better decision making (e.g., adaptive video optimization) to control and programmability for a better adaptability and flexibility to services (e.g., by controlling resource allocation, adjusting the handover logic, changing functional splits, updating precoding matrix, or even disabling/enabling ciphering/deciphering etc.).

To deploy such RTC and runtime relation in a disaggregated RAN, the multi-agent model can be applied among distributed entities, i.e., CU and DUs. These multiple agents are aware of the applied functional splits and can be controlled centrally (or delegated) to change the applied functional split in between. In particular, Figure 25 shows the model with multiple FlexRAN agent on the top of RAN entities (i.e., CU and DUs). These agents provide the programmability on the data plane as it not only enables the composition of shared and

Table 2: FlexRAN API calls.

API	Target	Direction	Example	Applications
Configuration (Synchronous)	eNB, UE, Slice	RTC → Runtime	UL/DL cell bandwidth, reconfigure Data Radio Bearer (DRB), Measurements	Monitoring, Reconfiguration, Self-Organizing Networks (SON)
Statistic, Measurement, Metering (Asynchronous)	List of eNB, UE, Slice	Runtime → RTC	Channel Quality Indicator (CQI) measurements, SINR measurements, Reference Signal Receive Power (RSRP) / Reference Signal Receive Quality (RSRQ) / UL/DL performance	Monitoring, Optimization, SON
Commands (Synchronous)	runtime	RTC → Runtime	Scheduling decisions, Admission control Handover (HO) initiation	Real time Control, SON
Event Trigger	Master	Runtime →RTC	TTI, UE attach/detach, Scheduling request, Slice created/destroyed	Monitoring, Control actions
Control delegation	runtime	RTC → Runtime	Update DL/UL scheduling, Update HO algorithm	Programmability, Multi-service

dedicated network functions requested by the network slice but also collaborates with others for programmable functions split. For instance, when changing the functional splits between CU and DUs via deploying the PDCP from CU toward the DUs, related agents are reconfigured by the controller and the runtime can recompose the end-to-end RAN data plane forwarding path for each instantiated slice. Further, these multiple agents are split-aware and thus they only provide the corresponding statistics and measurements of the deployed VNFs toward the controller.

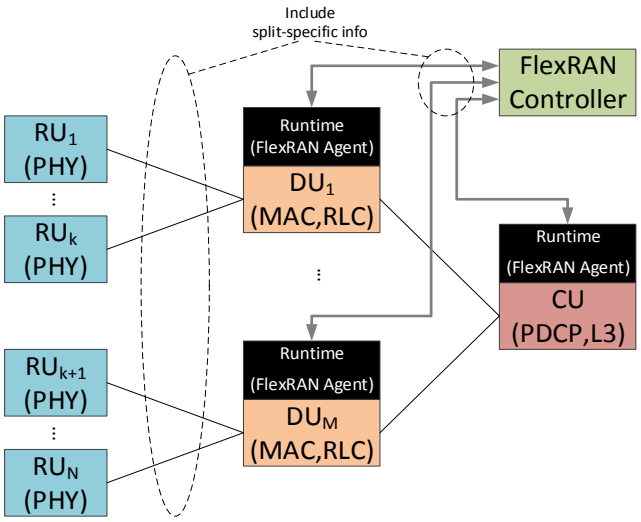


Figure 25: Multi-agent model and FlexRAN controller in a disaggregated RAN.

3.5 Porting of OAI on the Zynq platform

As the requirements for the RAN become more complex, requiring higher speeds with channels operating with more than 100MHz widths, the efficient real time processing becomes a challenge. The current architecture of solutions designed to run over commodity GPP, such as OAI, are limited by such factors. To this aim, UTH is developing some new target hardware that will enable the real time handling of such channels, by offloading computationally intensive tasks to the Zynq FPGA platform. This is expected to reduce the execution time of the offloaded functions by a factor of 3, and the overall layer execution (e.g. PHY) by a factor of 2. For the efficient implementation of the suggested functionality, the respective interfaces must be developed in order to allow the communication from the user-space OAI execution to the FPGA implementation of the functions. To this aim, several components need to be designed and developed in order to allow this offloading process.

For the purpose of offloading specific functions to the FPGA, the OAI code will be overridden to use functions from an API that will be developed. This API will match the subsequent call of the corresponding offloading functions on the FPGA, through the mapping that is done over the driver. The driver is in charge of providing the interface to the FPGA from the OAI eNB/gNB point of view. The driver will be handling all the IO processes with the FPGA, providing the requested input for the FPGA functions, and retrieving the results to OAI. Since OAI is strictly timed in terms of execution, as receive and transmit threads have to be executed in less than 1 ms, the driver shall communicate with the FPGA in significantly less time. In [47], this communication overhead is calculated to range from 2-4 μs, thus rendering such process to be feasible. Similarly, since the FPGA offloading tasks take up significantly less time to be completed, the driver shall maintain their output in a cache memory, so that to deliver them to OAI executable when needed.

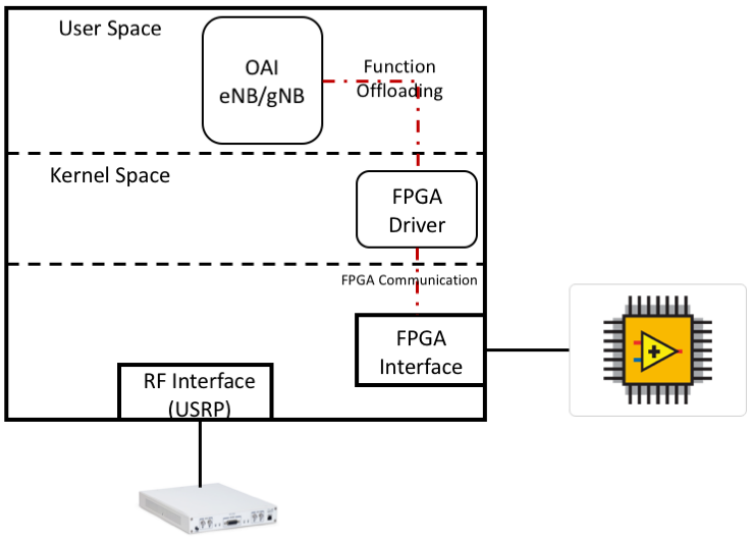


Figure 26: FPGA offloading process for the OAI RAN.

3.6 Point-to-Multipoint (P2MP) MAC processor

The characteristics of the 60 GHz band strongly benefit directive communications in the physical layer (PHY), both for steering the signal energy to the appropriate direction as well as to reduce the inter-symbol interference (ISI). The use of directional antennas efficiently at 60 GHz require of special MAC protocols that take into consideration the singular characteristics of 60 GHz channels.

Beamforming represents the key technique to compensate the severe channel attenuation and to reduce interference in mmWave networks. The 5G-PPP 5G-XHaul project put an enormous effort on developing a full analogue front-end (AFE) with beamforming capabilities [48],[49], which will be leveraged in 5G-PICTURE for extending the programmability of the mmWave meshed BH solution. The P2MP MAC processor accounts on the architecture of this AFE (up-/down conversion stage plus analogue beamforming capabilities).

The existence of programmable platforms poses a tremendous impact in mmWave communications, in a way that they allow low-cost setting up and configuring links. Stations with available steering capabilities allow cheap configuration/installation and P2MP connection schemes. In terms of performance, it is suitable to make available the best beam steering configuration (beam code book) before the communication is established. In meshed backhaul networks it is indeed important to search for possible communication links between stations which are expected to communicate between each other. The directionality of the communications allows simultaneous communication between the stations but, in that case, proper scheduling of the transmissions is necessary, having the stations to be synchronized.

3.6.1 Objectives

One of the goals of scheduling is to calculate the proper communication/link setup to achieve the maximum throughput and low latency depending on the established network topology and the current load of the network. To that end it is important to extract routing tables which have to be maintained to enable efficient frame/packet forwarding.

In a mmWave meshed network, one of the expected features of the mesh itself is to be able to calculate alternative communication/link setup schedules for unavailable links. Is the MAC layer who is responsible of detecting these unavailable links and to react accordingly pursuing maximum throughput and low latency.

On the other hand, the high throughput in such conditions (including beam switch duration or any additional guard intervals of the system), is only possible when minimizing PHY and MAC protocols overhead. This is normally achieved with short packets aggregation or with retransmissions on MAC layer. We aim at keeping functionalities into the MAC which were supposed to lay in higher layers in a sense to be able to recover packets

3.6.2 Installation and link establishment

Communication using directional antennas (beamforming / beam steering) can only be established when both communication end points steer the beam to the other. Therefore, before a communication can be established,

beam training between the stations must be accomplished. Depending on the scenarios, the complexity of the neighbourhood discovery changes according to the possible beam positions respective the opening window. It can be distinguished between low distance and high distance scenarios.

Pencil beams with an opening window of about 2 degrees (2°) allow data communication in scenarios with high distances of many hundred meters. In the worst case, all beam positions of a station have to be tested for each beam position of the other station. In low distance scenarios the opening window can be increased which reduces the amount of beam positions to be tested.

The beam setup phase can be done at the beginning as a separate procedure or during the running MAC protocol. After knowing the positions of the station(s), represented by the beam positions, a normal communication mode can be triggered. The switching to the beam positions has to be done before communication and of course “updated” during communication. The AFE, and concretely the beamformer or vector modulator (VM), should support storage of more than one weight set in order to get low switching durations between different beam directions without reprogramming the VM.

3.6.3 Mesh network architectures/topologies

Different network architectures and topologies are considered in a meshed BH, such as:

- single-hop point-to-point,
- single-hop point-to-multi-point ,
  - star
- multi-hop
  - tree

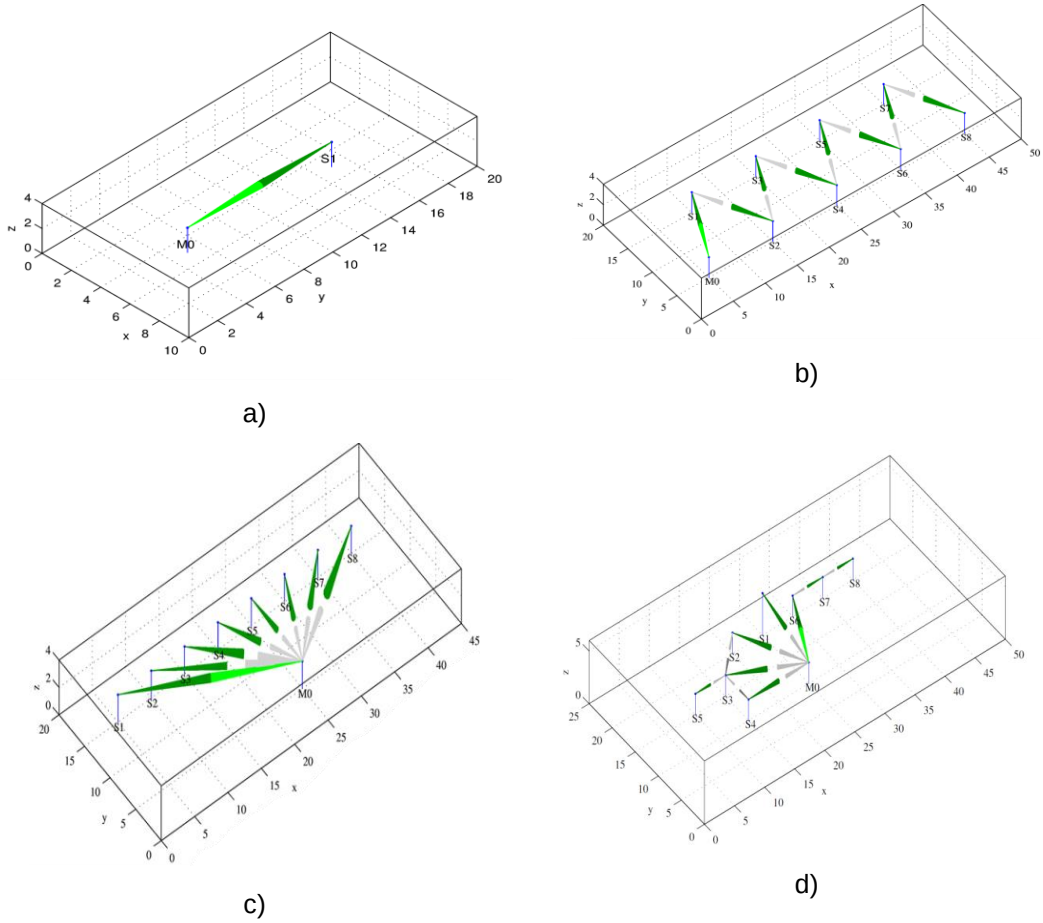


Figure 27: a) single hop point-to-point, b) daisy chain, c) single hop point-to-multipoint, d) complex mesh.

3.6.4 Medium access

The common used access scheme is half-duplex TDMA, given the design of the AFE. This design consideration ruled out adopting a full-duplex mode, which would increase considerably the complexity of the AFE; the use of polarized antennas, which would double the size of the antennas and, hence, the form factor of the solution; and the use of FDD, which requires very good channel separation and a filter bank when channel bonding is supported.

A channel access scheme has to be defined in order to reduce protocol overhead in the MAC and PHY layer. This depends on current network architecture (scenario) and the requirements on data throughput and latency.

In a single-hop P2MP scenario, only one link can be served at the same time. Depending on the latency and/or throughput, the service time and repetitions of each link has to be specified. A longer service time increases the throughput but also increases the latencies for the other links.

Some aspects have to take into account for multi-hop P2MP which can be structured as chain (Figure 27b), tree (Figure 27c), and/or “true” mesh network (Figure 27d).

3.6.4.1 Chain / Tree

This architecture is used in scenarios where more (LTE) stations on a lamp poles served by a feed point at start of street/chain. Not all stations are visible for the feed point/station. Only half of the links can be active on the same time in order to move the packets through the stations. The maximum latency of the last station depends on the amount of stations/links and service time of a link configuration. There is no alternative route possible in case of blockage of a link. Worst case is the blockage of the first link.

3.6.4.2 Mesh

Mesh architecture combines all architectures single-hop, multi-hop, chain/tree. This allows additional feature like independent stream and using alternative routes when links are blocked. Due to the use of directional antennas the beam switching has to be done before communication and has to be known for all stations. A switching scheme (link configuration over time) has to be distributed to all stations. The link configuration depends on the requirements of the streams/flows (throughput, latency) and has to be calculated by a central module. Additional link configurations can support blocked links automatically when a mechanism is implemented which detects link blockage. Rain can also degrade link throughout which requires a reducing of modulation scheme or increasing the FEC. The new channel conditions also need recalculation of the switching scheme.

Switching scheme calculation should be aware of MAC and PHY parameters of each link which results in different frame transmit durations. The service time of a link should be a multiple of the frame durations including additional overhead or the frame. Otherwise, frame length/duration has to be calibrated.

3.6.5 Functionalities of the MAC layer

The MAC layer should support mechanisms reducing the PHY and MAC overhead like frame aggregation. The length of such an aggregated frame depends on the current channel conditions and the allowed transmission duration time which depends on the switching scheme.

Frame retransmissions for important or all frames should be supported. This avoids propagating errors (missing frames, e.g. ACKs) to higher layers. The response time increases and time consuming error mechanisms of higher layers not necessary (e.g. TCP goes into slow-start when frame is missed).

Switching scheme (TDMA) should also select TX and RX mode of station of current switching configuration. This avoids negotiation before communication/transmission which results in increase of overhead and reduces the throughput.

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

TransPacket’s IP Cores can be integrated in SDN systems through TransPacket’s NETCONF/YANG based data-models. TP will contribute in Task 3.2 by making available to the project the yuma123 NETCONF/YANG open source framework, and, for demonstrator purposes, a NETCONF interface and a YANG model for low and fixed latency Ethernet. For the purpose of configuration and monitoring of the Ethernet-based TP IP cores, a yuma123 NETCONF/YANG framework will be used. The NETCONF server framework is developed as

yuma123, a free BSD licensed system[15] which is made available to this project and maintained by TP. Its purpose is to provide an open source – currently the only NETCONF/YANG toolchain in Debian – YANG API in C and YANG based CLI (*yangcli*) and server (*netconfd*) applications. Branching from the last BSD licensed branch of the now proprietary YumaWorks[16], yuma123 is being evolved in two key directions: (1) creating a stabilized and mainstream system by fixing critical bugs and a build based on autoconf/automake; (2) extensions added for support of new standards as e.g. IETF standards (ietf-nacm, ietf-system, etc.), new YANG extensions, ietf-yang-library and partial support for YANG 1.1, NMDA support, etc.

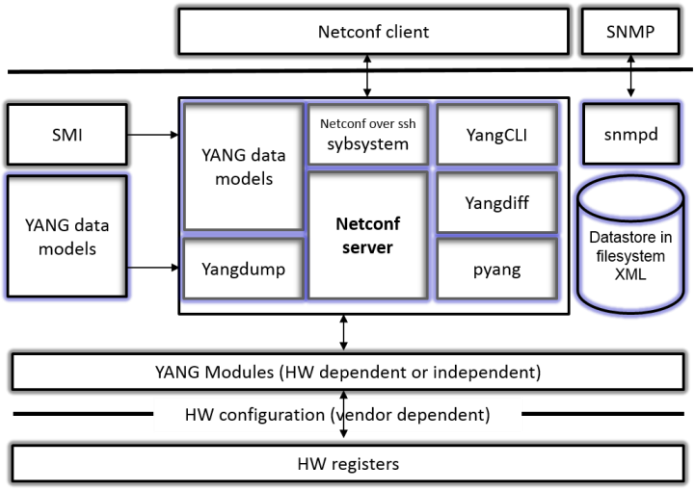


Figure 28: The Yuma123 framework.

The detailed framework is illustrated in Figure 28. The main components of the framework are:

- netconfd: NETCONF server engine which is the interpreter of the YANG data model and NETCONF Protocol, current version based on RFC 6241 and 6242, including data types, abstract object for PDU components and remote procedure calls (RPCs).
- yangcli: YANG-driven NETCONF client application that supports scripts, XPath, and many automated features to simplify management of NETCONF servers.
- Yangdump: used to generate the template C code for the server instrumentation library (SIL) for the YANG modules.
- Yangdiff: compare revisions of YANG modules.
- Subsystem for NETCONF over SSH transport.
- pyang: verification/validation of YANG modules by reusing existing tools written for alternative schema languages.

The NETCONF datastores contain YANG model based data structures (running, candidate and start-up configurations) which represent the configuration of the device containing the NETCONF server. This configuration can be saved in a non-volatile storage so the configuration can be restored upon reboot.

The Server Instrumentation Libraries (SIL) are compiled loadable modules implementing the YANG model behavior (managed by the NETCONF server) by controlling the networking device. It is the glue between hardware and software and is generated by parsing the YANG data models (yangdump in yuma123). The code is implemented in C and it contains functions that are hardware dependent, e.g. interface and protocols configuration, and independent, e.g. SNMP, NTP, RMON, etc. E.g. for a new functionality or vendor support the following simplified steps would be required:

1. Create the YANG module data model definition, and/or use the existing YANG modules.
   - Validate the YANG module with the yangdump.

[15] http://yuma123.org/wiki

[16] https://www.yumaworks.com

2. Test the interface by loading the YANG modules in netconfd without corresponding implementations. A dry-run validation without actual target hardware and SIL implementation.
3. Create a SIL that implements the YANG model.
   - C file implementing the model using the server side yuma123 APIs and the device side instrumentation and hardware access APIs.
4. Compile and install the newly generated SIL library.
5. Run the *netconfd* server and load the new module.

Currently, IEEE 802.1, The Ethernet TSN group is standardizing YANG models for the IEEE 802.1Qbv, IEEE 802.1Qbu and IEEE 802.1Qci in PAR IEEE 802.1Qcw, while IEEE 802.1Qcr will include its YANG model. Thus for being compatible with- and building on the same framework as standard TSN protocols, new YANG models will be developed for supporting the configuration of the TP Ethernet-based low and fixed latency IP cores.

4 Hardware Abstractions

5G-PICTURE is defining several hardware abstraction methodologies to enhance usability of the programmable network platform and to abstract the implementation details of each platform. In particular, this section will describe: i) the Application Programming Interfaces (APIs) for the wireless platforms and to provide configurability of data plane implemented with OpenFlow/NETCONF interfaces as well as with low-level API for FPGA register configuration via Ethernet-based commands. ii) Interfaces for connecting Physical Network Function (PNF) running LTE Layer 1 to VNFs running LTE layer 2 and above. iii) programming languages for data plane programmability like P4 for packet level programmability and OpenCL for signal processing level. The above mentioned abstraction methodologies are described in detail in the rest of this section.

4.1 APIs

In this section will be described the set of APIs that are under development under the activities of Task 3.2. In particular, the APIs will be used both to provide SDN-like configurability to several wireless platforms like the BWT Typhoon and the GateWorks Ventana and to provide configurability for the dataplane via Ethernet-based commands.

4.1.1 APIs for the BWT Typhoon platform

The BWT Typhoon platform described in section 2.5 features an NPU with an open Linux platform, where data plane forwarding can be enabled with an implementation such as Open vSwitch and data plane programmability can be achieved by running OpenFlow. The approach for controlling and abstracting the hardware functionality will rely on a control application running in user space (in the Typhoon NPU). The goal is to prepare this application to respond to the control plane in the network and, then, route commands or requests internally to other user space applications as well as the network device driver developed by BWT for its HYDRA modem. A specific set of API will be developed to communicate with the device driver that is responsible for handling the configuration command to the hardware.

One example of virtualisation functionality that is going to be explored by BWT in 5G-PICTURE is that of synchronisation functions in WP4. The aim is to abstract the synchronisation functionality in the Typhoon platform by relying on an OpenFlow-based controller. The controller will be prepared to receive requests from a synchronisation harmoniser, to be developed in WP4. When receiving control plane packets, the control application will hand the suitable configurations or information requests towards both the user-space synchronisation application running in the Typhoon's NPU as well as the network device driver.

4.1.2 APIs for the GateWorks Ventana platform.

Configurability of Sub-6GHz transport nodes is achieved through the implementation of different interfaces: i) a northbound interface 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. The relationships among the different interfaces and their applications is further discussed in the following paragraphs.

For the management of Sub-6 GHz transport nodes, an SDN controller based on the OpenDayLight (ODL) Boron distribution is implemented. The controller is responsible for managing and configuring the IEEE 802.11-based wireless interfaces according to the requirements of the different tenants sharing the backhaul infrastructure. Physical interfaces (IEEE 802.11ac/n) are used either as access network interfaces or to form wireless backhaul links with other transport nodes. The physical radio resources used by those interfaces are then assigned to different tenants by means of virtual interface instances. For example, a given wireless access interface can be used to provide access to two (or more) users of two (or more) different tenants' networks. While the physical device to which those users are connected is the same, a different virtual interface is used for each tenant. The physical device controls physical parameters such as frequency channel, modulation and coding scheme, transmitted power, etc., which will be common to all virtual interfaces instantiated thereof. Virtual interfaces can be configured to have less/more radio resources (e.g. airtime) or different security parameters.

The instantiation, configuration or shutdown of wireless interfaces (physical or virtual) is not done manually on the devices, but via a dedicated NETCONF API, where the NETCONF servers are running on the Gateworks (GW) SBCs (cf. section 2.7) and the client accessing those services is a software module running in the ODL controller. NETCONF is a protocol that allows the definition and execution of RPCs on the platform running

the server, i.e. the GW SBCs. For example, setting up a physical wireless interface, generating a virtual interface on top of it and configuring the transmission power of the interface can all be done via the NETCONF protocol, assuming that the underlying mechanisms have been implemented in RPCs. The NETCONF API development is based on CESNET-Netopeer[17]. The server being used is based on the version from the Netopeer repository with slight modifications to make it compatible with GW's ARMv7 architecture. The API is being entirely developed by I2CAT from scratch, specifically designed to enable the different Sub-6GHz network slicing functions envisioned in 5G-PICTURE's WP4 [22] and, at the same time, aiming to remain flexible and easily extendable.

The ODL controller is responsible for connecting the different servers and to reconfigure them on demand. Changes of the wireless node's configuration can be triggered automatically or via specific commands issued to the ODL controller via a northbound interface implemented with a REST API (which is not further detailed here). A reason for reconfiguring a wireless node could be, for example, the instantiation of an access point in the area covered by a 5G-PICTURE node, where clients of a particular tenant start requiring connectivity services. Such a request would be handed to the ODL controller which, in return, would use NETCONF to configure the nodes (the access node and transport nodes in the path between the client and the tenant's network). The wireless nodes within Netopeer are defined via a Yang-model, called "i2cat-box". This Yang Model is shown in Figure 29.

The i2cat-box's TransAPI consists of the following elements:

- A set of callbacks: functions which will be invoked every time a change on the configuration is requested (this change can be external, due to the interaction with the controller, or internal, due to an administrator's manual – e.g. through *ssh* – connection to the server).
- A database: stores the current status and current configuration for the node.
- Transaction queues: these store the changes being made to the device before the changes are committed to the database.
- A set of scripts: Invoked by the TransAPI to reconfigure the node.

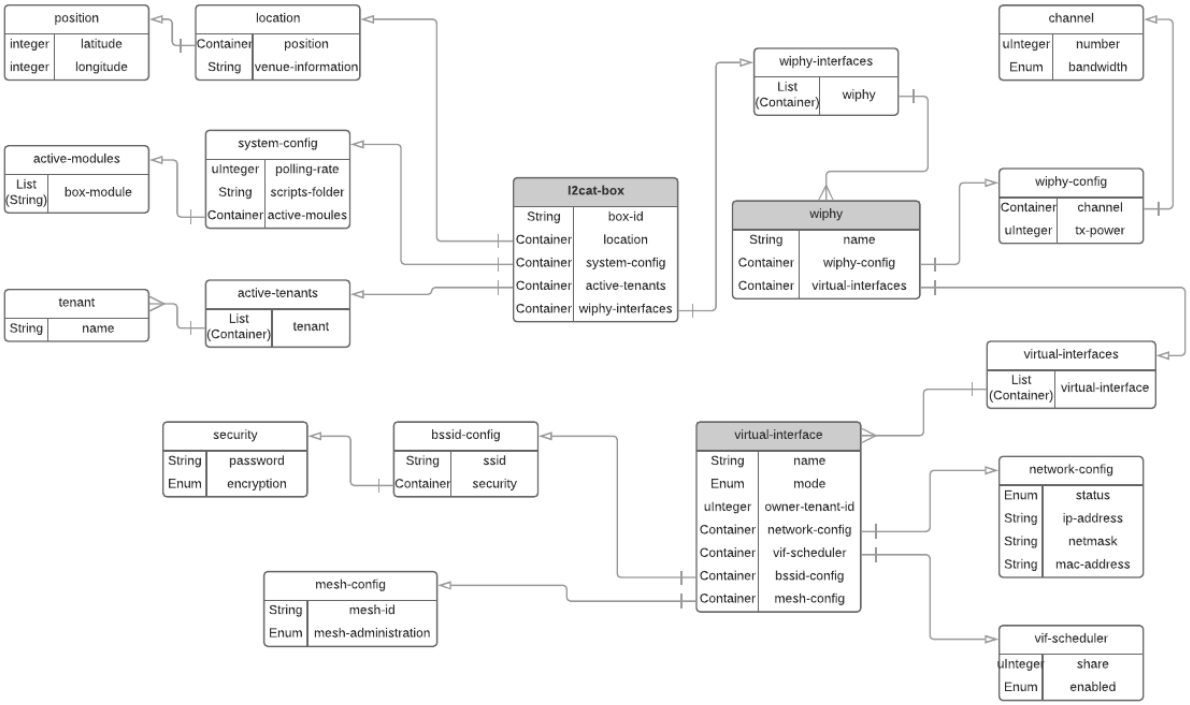


Figure 29*:* **Yang model of the I2cat-box.**

[17] https://github.com/CESNET/netopeer

4.1.3 Ethernet-based API for Read/write of FPGA registers

TransPacket makes available a custom protocol specification which can be used for communication between a controlling CPU hosted in server/workstation and the FPGA chip(s) of the system. The communication between the embedded FPGA Microblaze soft core microcontroller and the external CPU is running over an Ethernet-based protocol. The protocol is designed to be usable in a multi-CPU, multi-FPGA environment. In this version it is not supported to send information from FPGA to CPU. If the FPGA needs to send asynchronous messages to the CPU(s) due to e.g. a network event or failure, this protocols needs to be extended. For now, we assume that the CPU(s) is responsible for polling the FPGA for status information.

All information exchanged between the CPU and the FPGA is contained in ordinary Ethernet frames with a specific EtherType value, e.g. currently 0x1515 is being used. An example of the Ethernet frame is illustrated in Figure 30. The fixed size of all packets is 320 bytes including the 14 byte header and excluding the Frame Control Sequence (FCS). The reserved fields are put in place for padding when necessary, to simplify the FPGA design as the first 32-bit word, the magic cookie will be at word boundary.

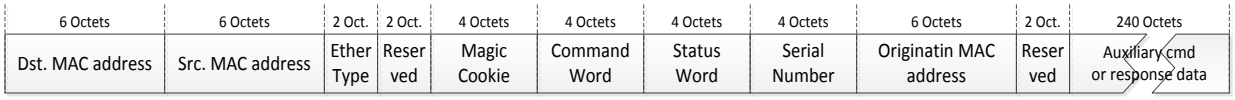


Figure 30: the Ethernet frame for Read/write of FPGA registers.

For addressing the FPGA, the CPU uses either the broadcast address (all-ones), or the link layer (MAC) address of the particular FPGA as the destination address for packets going to the FPGA(s). The EtherType field of the frames is set to 1515 and the Command Word and auxiliary data fields are used to instruct the FPGA.

Packets that have a destination MAC address different from the MAC address of the FPGA, and different from the all-ones broadcast address, must be discarded (they are not intended for the FPGA).

A set of conditions is set for discarding incorrect packets based e.g. on the magic cookie value and set matching values of other fields. If the packet is accepted for processing, the entire contents of the incoming packet is copied into the output packet buffer, while the source address is copied into the destination MAC address of the output packet buffer. This makes sure any responses are sent back to the CPU that initiated the command. The following commands are available:

0. POLL – Do nothing except retransmitting the output buffer;
1. NOP – No Operation, i.e. do nothing, but initialize the output buffer and transmit it;
2. CQ – (Seek You – Reply contains device info and MAC);
3. REG – Register Access, read or write one or more registers.

The last and most important command enables programming (write) of the IP Core and reading the register values for e.g. statistics. The auxiliary data field is paired up as two 32-bit variables for each register to be read/written. The two 32-bit integers contains:

1- aux[2*n]: WriteEnable + RegisterBank * 65536 + RegisterNumber
2- aux[2*n + 1]: The value to be written to the register

Where n is between 0 and 34. The register bank typically identifies what kind of operation is needed to address the correct register(s). The value to be read/written may not be a 32-bit value. In such cases, it is dependent upon the particular register or register-bank how the value is represented (and whether there are don't cares or reserved bits). The *WriteEnable* is 0 for read operations, and 0x80000000 for write operations. This allows for 32768 register banks, with 65536 registers in each. This division is only arbitrary and can be changed. Should a register bank have more registers than 65536 (for instance the buffer memory), several adjacent register banks can be used to cover the required number of registers.

4.2 OAI or Interface for Physical Network Functions

As OAI is providing a state-of-the-art solution for beyond 4G and 5G prototyping based on Open Source code, the project will utilize the solutions provided by the platform in order to demonstrate the potential of the provided technology solutions. Although the high-level interplay between the different network functions will be provided in WP5, the respective interfaces shall be incorporated within the platforms that built in WP3 in order to handle

the low-level configuration of the components. Towards this aim, we plan to extend OAI in order to allow its real-time configuration as VNFs.

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 [46]. nFAPI defines a network protocol that is used to connect a PNF running LTE Layer 1 to a 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.

In this context, the project will pursue the integration of hardware integrating the PHY layer and accessible through the nFAPI interface with the VNFs implementing the rest of the stack. For this purpose, an agent software will be developed that will allow the configuration of the nFAPI parameters, as well as the PHY RAN parameters of OAI through a high-level REST based API. The solution will leverage existing code capabilities that exist in OAI, such as asynchronous networking interfaces based on TCP/UDP connections. Apart from the nFAPI interface, 5G-PICTURE will engage in the configuration of the respective agents configuring other versions of the OAI implementation, such as the IF4p5, IF5, CU/DU that is implemented in WP4, etc.

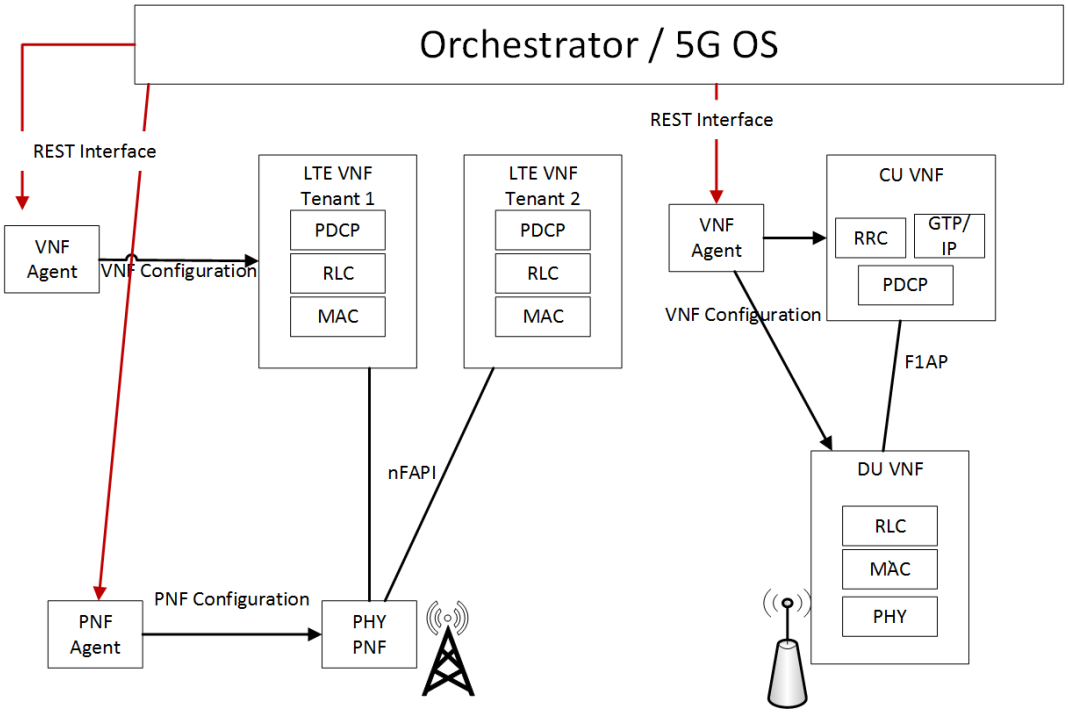


Figure 31: High level Agent description for handling OAI PNFs and VNFs.

4.3 Programming languages for data plane programmability

4.3.1 OpenCL development of network functionalities

OpenCL[18] (Open Computing Language) is 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. It further offers a huge set of freely available community-driven programming libraries for various tasks in fields like signal processing, data analysis, and machine learning.

OpenCL is also well integrated in the Xilinx SDx[19] family of FPGA development environments. ADVA will therefore evaluate the use of OpenCL for rapid high-level development of VNFs to be defined in WP4 in a WORE

[18] https://www.khronos.org/opencl

[19] https://www.xilinx.com/products/design-tools/all-programmable-abstractions.html

– Write once, (compile and then) run everywhere – manner, and is further considering performance comparisons to similar VNFs developed with optimized architecture-specific programming languages and frameworks.

4.3.2 Development of P4 compiler for Spectrum device

As stated in section 3.3, 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 [8].

4.3.2.1 Mellanox P4 Compiler main components

The main components of the P4 compiler that are currently under design are:

1) P4 architecture model:

the P4 architecture identifies the P4-programmable blocks (e.g., parser, ingress control flow, egress control flow, deparser, etc.) and their data plane interfaces. The P4 architecture is a contract between the program and the target. Manufacturer must therefore provide both a P4 compiler as well as an accompanying architecture definition for their target. The P4 architecture goal is to come up with a uniform target architecture in that way we can benefit flexible and programmable pipeline but in the same time make it HW agnostic to the application designer. All ASIC vendors must accept this target architecture. In order to achieve this goal we want to use the Switch Abstraction Interface (SAI) pipeline as a reference[20].

2) Mellanox P4 target architecture:

The current Mellanox P4 target architecture consists of from 5 programmable blocks (1 parser block, and 4 control – match action).

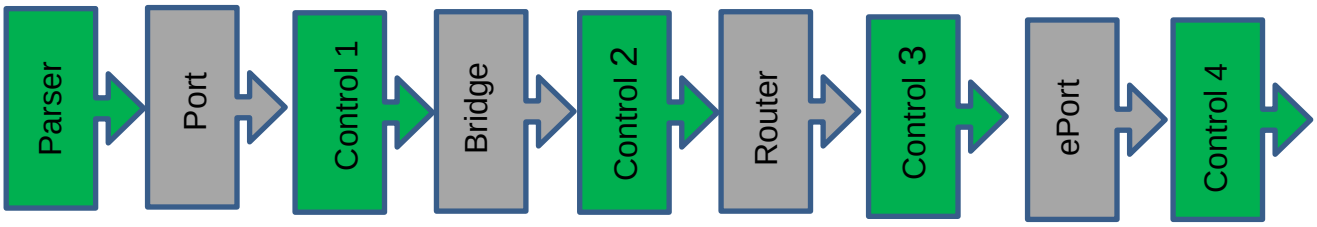


Figure 32. Mellanox's P4 target architecture.

Programmable block 1 – parser: Mellanox provides parsing graph base line user will be able to add up to 4 new nodes to the packet-parsing graph.

Programmable block 2 – ingress port: ability to define chain of multiple match action tables supported actions – drop, forward to port , mirror, packet modification, routing – including Equal-cost multi-path (ECMP) routing, tunnels encap ,tunnel decp , set QoS, counters, meters ,go to table.

Programmable block 3 – ingress router: ability to define chain of multiple match action tables supported actions – drop, mirror, packet modification, routing – including Equal-cost multi-path (ECMP) routing ,tunnels encap ,tunnel decp , set QoS, counters, meters ,go to table.

Programmable block 4 – egress router: ability to define chain of multiple match action tables supported actions – drop, mirror, packet ,forward to port , packet modification, set QoS, counters, meters ,go to table.

Programmable block 5 – egress port: ability to define chain of multiple match action tables supported actions – drop, egress mirror, packet modification, set QoS, counters, meters ,go to table.

3) P4 compiler architecture

The P4 compiler architecture is depicted in Figure 33. The front end is target independent and is related only to the language constructs. The front end functionalities are split in two phases: (i) A Syntactic phase that is BNF based. Currently the P4 compiler under development is able to read the source files and provide as output the symbol tables; (ii) Semantic phase – Verifies the Symbol tables. Mellanox is extending the default semantic checks with platform specific ones. The second part of the P4 compiler architecture is the Mid-end. The Mid-

[20] https://github.com/opencomputeproject/SAI/tree/master/doc/behavioral%20model

end provides platform independent API generation. In particular, the Mellanox compiler will generate two sets of APIs:

1. SAI-Flex API: auto generated SAI API for each newly created P4 match action table;
2. P4runtime: p4runtime API that will enable configuring the box for an SDN Controller as defined by the P4runtime specifications[21].

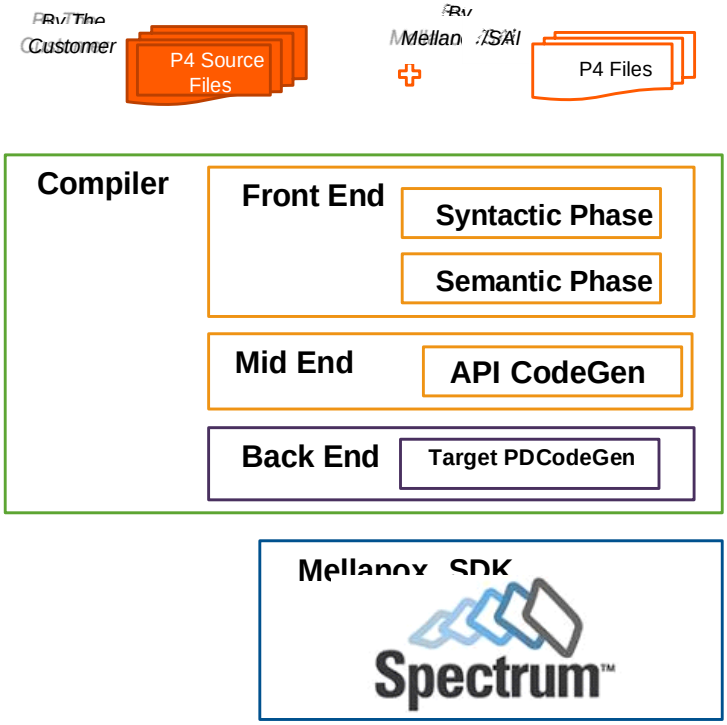


Figure 33. P4 compiler architecture.

Finally, the Back End is devoted to generate the specific commands to configure the target. This phase will use the already developed Mellanox SDK to create the actual configuration for the switch.

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

5 Hardware Technologies

This section describes optical and radio technologies and components, which provide the integrated physical network infrastructure used in the 5G-PICTURE project. In particular, the 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 channel modelling and comparison between Sub-6 GHz and mmWave frequencies, iv) the Active Antenna Distributed Unit (AADU) architecture with functional split options, v) SDN enabled routing and forwarding/C-RAN as programmable network function, vi) use of MIMO technologies at mmWave wavelength and vii) the Multi-Link and Multi-Protocol PHY interfaces (MPIs) allowing flexibility to mix and match a variety of protocols and technology solutions used in 5G-PICTURE.

5.1 Passive optical technologies

ADVA has many years of experience in developing WDM-PON (Passive Optical Network) technology[22]. There are currently several new prototypes in development. ADVA will offer the reuse of the WDM-PON prototype introduced in the 5G-XHaul project[23] for any kind of PON needs in the 5G-PICTURE network, e.g. the connection of BBUs and RRUs as shown in Figure 34.

The Optical Line Terminal (OLT) is operating as a pluggable module of the ADVA FSP 3000 AgileConnect[24] scalable optical transport product line, using a Red Pitaya STEMlab 125-14 board[25] as independent control unit. It can dynamically connect with up to 8 Optical Network Units (ONUs). Each ONU wavelength in the prototype system is able to reach bidirectional bit rates of at least 10 Gb/s over a 20 km transmission distance. The downlink operates at the L-band. The uplink operates at the C-band.

A dedicated management port of the OLT will offer a WDM-PON specific NETCONF/YANG based control and monitoring service for SDN integration [60].

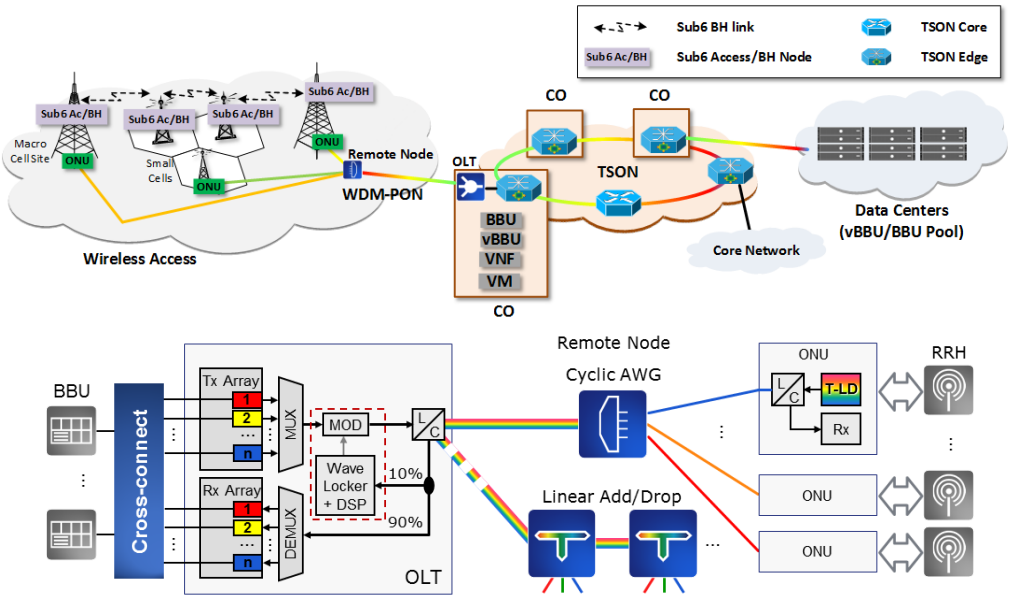


Figure 34: ADVA's WDM-PON in the 5G-XHaul network.

[22] https://www.advaoptical.com/en/products/technology/wdm-pon

[23] https://5g-ppp.eu/5g-xhaul

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

[25] https://www.redpitaya.com/f130/STEMlab-board

5.2 Elastic optical technologies

To support the varying degrees of bandwidth and latency requirements introduced by the various RAN deployments, the use of dynamic elastic optical network solutions supporting a higher degree of time granularity (i.e. subwavelength) can be achieved by the TSON technology. TSON includes two different types of nodes, namely edge and core nodes, incorporating different functionalities and levels of complexity. The edge nodes provide the interfaces between wireless, PON and data centre domains to the optical domain and vice versa. The ingress edge nodes are responsible for traffic aggregation and mapping, while the egress edge nodes support the reverse functionality. The edge nodes process the incoming data streams and generate optical time-slices at the ingress edge node, and also regenerate the original information from time-sliced optical frames at the egress edge node. These technologies allow handling of Ethernet frames, natively supporting a broad range of framing structures and communication protocols including CPRI (or eCPRI), Open Base Station Architecture Initiative (OBSAI), 10G Ethernet as well as protocol extensions required in support of the functional split concept under discussion in the framework of 5G. In 5G-PICTURE, at the optical network ingress TSON edge node, the interfaces receive traffics generated by fixed and mobile users. The incoming traffic is aggregated into optical frames, which are then assigned to suitable time slots and wavelength for further transmission. At the egress point, the reverse function takes place, disaggregating the traffic to forward next nodes.

In 5G-PICTURE, the optical edge node is also equipped with elastic bandwidth allocation capabilities supported through the Bandwidth Variable Transceivers (BVTs). In addition to providing BH functionalities, this infrastructure also interconnects a number of DUs and end users with a set of general purpose servers and specific purpose HW. The use of general purpose servers is to enable the concept of virtual BBUs (vBBUs), allowing for an efficient sharing of compute resources. This joint functionality is facilitated by the edge nodes that comprise a hybrid subsystem of an I/Q switch and an Ethernet switch. The I/Q switch handles different functional split options with strict synchronization and bandwidth constraints, while the Ethernet packet switch handles BH traffic and relaxed FH transport classes. At the ingress part of the Ethernet switch module, the interconnection of DUs and CUs is provided, whereas at the egress part of the module, traffic is aggregated and placed in a suitable transmission queue. The elastic optical network solution will comprises BVTs, bandwidth-variable optical cross-connects and fast optical switching modules. This approach enables the allocation of variable-size spectral/time slots, thus supporting services with continuous channel allocation at various bit rates and services with sub-wavelength time slot allocation.

The advent of elastic optical networking, enabled by the adoption of a flexible channel grid and programmable transceivers, opens the door to a truly dynamic active management of optical networks. This is especially interesting in the context of supporting greatly varying transport services (both FH and BH) for the RAN. Furthermore, active networking enables to transparently set a RAN network over the optical network. For example, a pool of BBUs concentrated in a CU can be located at a selected node of the metro network segment, while the DUs can be attached to different other nodes. To support this functionality in a cost-effective manners, it has been proposed to use programmable sliceable-bandwidth variable transceivers (S-BVTs), which could be present at the 5G optical nodes in order to concurrently serve different cell sites. The (S-)BVTs can be remotely configured by the control plane (CP) for optimal management of the network resources. The parameters to be configured at each (S-)BVT include wavelength, spectral occupancy and modulation format/power per flow according to the network and path conditions.

5.3 Time sensitive Ethernet

5.3.1 Deterministic delay mechanisms for Ethernet

Ethernet bridges were originally designed for best-effort traffic with no requirement on maximum delay through a network. Due to the need of using Ethernet for audio and video transport in professional studios, there has been a drive in IEEE 802.1 Ethernet standardization for mechanisms ensuring zero congestion packet loss, as well as control on delay and Packet Delay Variation (PDV). Recently, main drivers for further evolvement in standardization include industrial control and automotive applications, with mobile fronthaul as the most recent.

In the IEEE 802.1 work, TSN mechanisms include both mechanisms for minimizing delay and for controlling the delay variation, ensuring that all packets always receive low and bounded delay. The IEEE 802.1Qbu defines a preemption mechanism enabling minimized delay on deterministic traffic when mixed with best-effort traffic within the same network. By disrupting the transmission of best-effort packets when a deterministic high priority packet arrives, packet delay caused by packet contention is lower than, for example, the strict priority mechanism where delay corresponds to the duration of a best-effort Maximum Transfer Unit (MTU) packet. Preemption is only performed if at least 60 bytes of the preemptable frame are already transmitted and at least

64 bytes of the frame remain to be transmitted. This results in a worst case of 1240 bit times (155 Bytes) of delay, and a best case of zero delay. Hence, the PDV corresponds to the duration of transmitting 155 Bytes. The preemption mechanism works hop-by-hop, fragmenting the best-effort packets and reassembling these at the next hop. Since fragments do not contain MAC address-headers, forwarding of fragments through bridges is not supported, and preemption may only be activated with bridges supporting the IEEE 802.1Qbu standard.

The IEEE 802.1Qbv (enhancement for scheduled traffic) defines how a set of queues, destined for an output port, can be served by a round-robin mechanism; it allows each of the queues to be served within timeslots, one-by-one in a cycle, scheduling one or more packets in bursts from each of the queues into designated time slots. The duration, and hence, start of the time slots, may vary. Moreover, time-synchronization by, for example, using the IEEE 1588 protocol, is required. The maximum delay on a packet caused by the bridge, is given by the duration of the scheduling cycle.

The IEEE 802.1Qch, cyclic queuing and forwarding (CQF), applies the IEEE 802.1Qbv in combination with IEEE 802.1Qci per-stream filtering and policing (PSFP). The mechanisms in the latter standard enable filtering (i.e. identifying packets in a flow through header inspection) and policing by only accepting packets arriving within a predefined time window on a defined port. It also assigns packets to an output queue. In CQF, two output queues exist for each output port, being alternately served by a cyclic IEEE 802.1Qbv scheduler, which allows input to the first queue while one or more packets in a burst from the second queue are being scheduled, and vice versa. It is all based on synchronizing the queues of all the bridges in the network, as well as synchronizing incoming time windows with the scheduling of the outgoing queues for each individual bridge. As a result, packets are scheduled in groups, receiving a fixed delay of one cycle time for each hop in the network.

Another traffic shaper, not requiring synchronization while still bounding delay, is the IEEE 802.1Qcr asynchronous traffic shaper. The shaper assigns an eligibility-time to all incoming packets. These eligibility-times are applied for selecting packets for scheduling. Packets are dropped if they stay too long within the bridge (i.e. beyond a predefined residence time). This mechanism has the ability to reduce the average delay, but maximum delay will still be higher, or at its best, equal to the synchronous mechanisms operating with asynchronous input streams. This is because by definition, in an asynchronous network the packet arrival pattern may be similar to a worse case pattern in a synchronous network. Hence, the maximum delay for IEEE 802.1Qcr will be equal to or higher than the maximum delay for IEEE 802.1Qbv.

A mechanism not relying on packet preemption, while enabling a mix of deterministic traffic and best-effort traffic in a network, is a time window based priority mechanism described for Integrated Hybrid (hybrid as in packet and circuit) Optical Networks (IHON). The mechanism eliminates PDV on the deterministic traffic by adding a fixed delay corresponding to the MTU of the best effort traffic. Best effort packets are scheduled in between deterministic packets whenever a gap is available that is equal to or larger than the packet waiting in a best-effort queue. Main benefits are elimination of any interference and PDV on the deterministic traffic caused by best effort traffic. Furthermore, different from the packet-fragmenting as in preemption, the mechanism works together with bridges that do not support it, allowing lowered PDV in the network for each node it is applied.

Furthermore, IHON describes an aggregation and scheduling mechanism where PDV from contention is avoided. The mechanism relies on preserving the packet gaps between packets in the individual deterministic packet streams.

Packet streams being aggregated are scheduled into time slots in a cycle synchronized across the network using e.g. a control packet at the start of each time slot. However, the packet streams are allowed to be asynchronous with variable length packets and still transferred with no added PDV. As illustrated in Figure 35, the streams being aggregated are divided up into virtual containers before being scheduled to the output. A fixed delay corresponding to one cycle time is added to each of the packet streams.

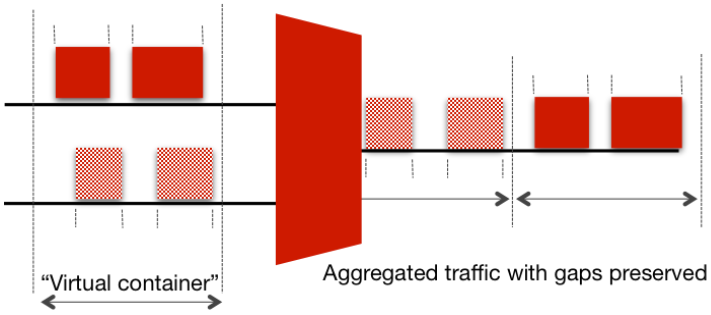


Figure 35: Aggregation of multiple deterministic packets streams into virtual containers while preserving packet gaps.

5.3.2 Bounded delay aggregation and fixed delay forwarding

A bounded delay when aggregating and forwarding packets in asynchronous systems may be achieved using a gap-planning and gap-detection scheme (GPGD). In difference from the IHON scheme aggregating in virtual containers with packet gaps preserved enabling fixed delay, the bounded delay scheme aggregates packets into time slots without preserving the gaps between the packets. The maximum delay is bounded by the duration of an aggregation cycle containing a number of time slots, similar to the method described in IEEE 802.1Qbv. A difference is that while the IEEE 802.1Qbv relies on a synchronous network for preserving the planned gaps, the GPGD scheme uses a GPGD for detecting vacant time slots in an asynchronous network. The gap-detection is similar to the scheme applied in IHON for insertion of Statistically Multiplexed (SM) traffic. For a node with an incoming traffic stream containing bypassing traffic, gaps corresponding to the size of time slots may be detected without PDV impact on the bypassing traffic stream. By allowing traffic to be added only within the duration of a time slot and by planning the maximum number of time slots that can be assigned between an ingress and egress point across the network, a bounded delay on the traffic to be inserted is guaranteed. For example, for a system with a cycle containing 8 time slots, a wavelength across a network from an ingress-node to an egress-node may be reserved for using the GPGD scheme. Along the path, traffic may be added and dropped within a number of time slots assigned to each node. Exactly which time slot and when the specific time slot occurs for each of the nodes are not fixed but varies according to the traffic pattern. However, within the duration of a cycle, a vacant time slot will be found for the node adding traffic, as long as the number of time slots along the path is not oversubscribed. Hence, for this scheme the PDV equals the maximum delay, given by the cycle time.

5.4 RF processing and modelling

5G mobile networks are expected to support higher mobility, higher data rates and lower latency [47]. Recently, we experience the offset of 5G standardization in 3GPP, widely known as New Radio (NR). In the context of NR, the air interface will extend, compared to 4G, to carrier frequencies from 1 GHz up to 100 GHz [50]. Moreover, it will co-exist with the evolved LTE (Release 13 and beyond) [47], and bring together existing and newly emerged technologies.

The solution of a large number of antennas at the BS, i.e. Massive MIMO, has been one of the main candidates. Due to the large multiplexing gain and antenna array gain, high spectrum efficiency can be achieved. Furthermore, high-energy efficiency is provided, because of the concentration of radiated energy.

Another highly researched solution to 5G networks is mmWave. First, the consideration of mmWave frequencies resolve the issue of limited spectrum. Moreover, mmWave, although limited to shorter ranges compared to Sub-6 GHz and the existence of penetration loss, research has shown that mmWave systems can support high data rates, reduce the antenna-size and are ideal for small-cell deployments.

At the same time, vehicular communications (i.e. cars, trains) constitute a great part of research towards 5G networks. Based on the above, for the Rail Vertical we will investigate the following two 5G solutions in the rail environment:

a) Sub-6 GHz LTE Massive MIMO coverage cell,

b) mmWave Access Points (APs) along the trackside.

To investigate the aforementioned scenarios, first the channel will be modelled with the Ray-tracing Tool developed in the University of Bristol. Based on those results, further processing will be performed to evaluate

the spectral efficiency (using Matlab RBIR simulator) and emulating the capacity of the network (using the Anite F8 Channel Emulator). The main characteristics of the Ray-tracing tool are discussed in Section 5.4.1.

5.4.1 Ray-tracing Tool

The mmWave 3D Ray-tracing engine, developed at the University of Bristol, constitutes a powerful tool that allows channel modelling of various environments. In particular, the Ray-tracing simulations can model important parameters like delay and angular spread, delay, phase, time of flight, Angle of Arrival (AoA) and Angle of Departure (AoD) of each ray, amplitude, etc. Adding beamforming is also an important tool that the simulator provides.

The Ray-tracing engine allows the investigation of any environment, as long as the particular map is bought and loaded. Then, for each simulation, the user can define the specific route, the number of APs on the route, the distance between them, the resolution, the transmit power and many other parameters that would simulate in the best possible way a real scenario. As a result, all possible ray paths are identified, at the end of the simulation, and subsequent significant parameters are stored in a series of files.

Both scenarios that will be investigated as 5G solutions to communications in a train environment, for the 5G-PICTURE project, will be based on channel modelling simulation on the Ray-tracing platform.

5.4.1.1 Sub-6 GHz LTE Massive MIMO coverage cell

In this scenario, depicted in Figure 36, 5G connectivity to trains could be achieved by Massive MIMO technology. APs on the train will be served by a Sub-6 GHz LTE Massive MIMO station per cell. High data rates are expected due to the multiplexing gain as well as the array gain that a Massive MIMO system offers. Massive MIMO stations could be connected via CPRI to coordinate connectivity as the train moves from one cell to another, i.e. hand-over.

For this scenario, we will consider a BS at a height of 25 m and APs on the train at 2.5 m high. Considering a 2 km long track, at a frequency of 3.5 GHz, the channels will be modelled with the Ray-tracing simulator, in the University of Bristol (CSN group). Moreover, with the Matlab RBIR simulator, the spectral efficiency will be investigated, in a rail environment, using the ray-tracer acquired channels. Finally, the F8 channel emulator platform will be used to emulate the capacity and compare results with the ones produced by the Matlab RBIR simulator.

5.4.1.2 mmWave Access Points (APs) along the trackside

For the mmWave scenario, depicted in Figure 37, we will consider mmWave APs, at a height of 3 m, placed along the trackside, 400-500 m apart, and around 3-5 m away from the tracks, and mmWave APs on the train, all at a height of 2.5 m. Thus, fronthaul will be based on mmWave technology. We will consider both 26 GHz and 60 GHz frequencies.

We will consider a track 2 km long. Like in the massive MIMO scenario, the channels will be modelled with the ray-tracer simulator. In addition, beamforming will be introduced, selecting beam width in azimuth and elevation. Then, further simulations will be performed using also the RBIR simulator and the F8 channel simulator to fully investigate and evaluate spectral efficiency and capacity of this scenario.

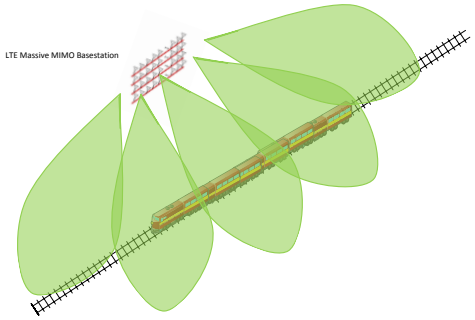


Figure 36. Vertical Rail – Sub-6 GHz LTE Massive MIMO cell.

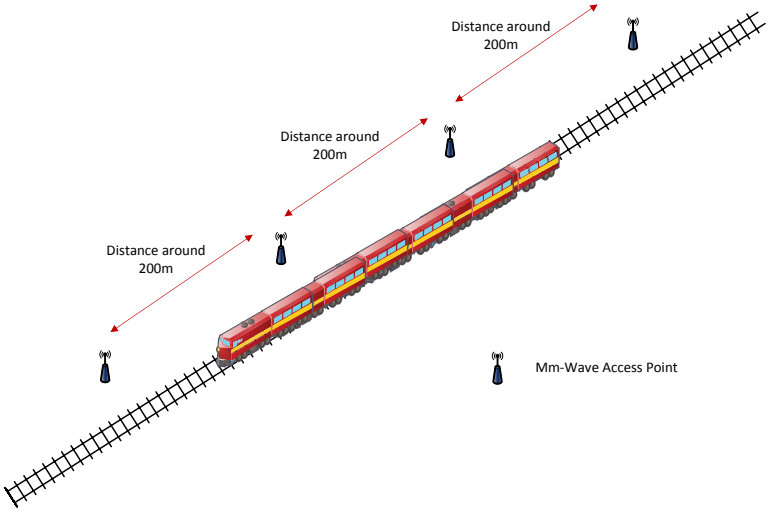


Figure 37. Vertical Rail – mmWave APs along trackside.

Figure 38. Bristol Temple Meads route (1.4 km).

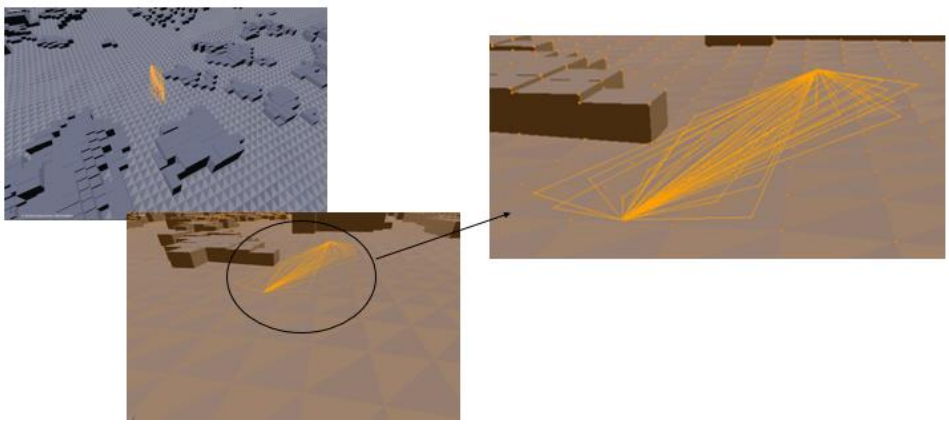


Figure 39. Bristol Temple Meads - Rays for a specific point.

Figure 40. London Paddington route (3.5 km).

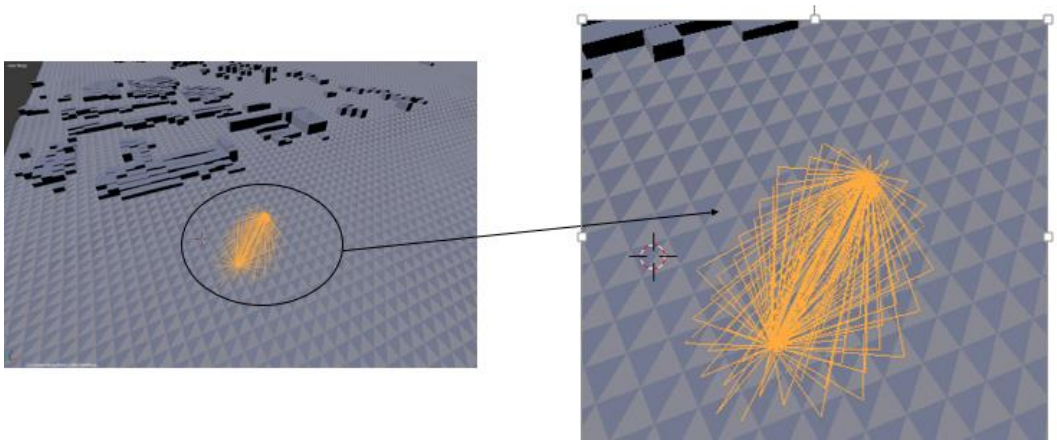


Figure 41. London Paddington – Rays for a specific point.

Finally, results of the Ray-tracing tool can be seen in Figure 38-41. First a 1.4 km route, depicted in Figure 38, at the Temple Meads train station in Bristol, was simulated with only one AP, at a height of 3 m, on the trackside (the beginning of the route) and one AP, at 2.5 m height, on the train. The resolution was 1 m and the transmit power 0 dB. Figure 39 shows all the rays for this scenario for a particular point. Similarly Figure 40-41 depict the route (at Paddington train station in London) for a 3.5 km track and the same parameters as before.

5.5 RF/BB processing

To address the limitations of the D-RAN and C-RAN approaches, 5G-PICTURE proposes a novel architecture exploiting flexible functional splits. The introduction of these splits allows dividing the processing functions between the CU and the remaining baseband processing functions. The flexible functional split will be implemented by using specific programmable network platforms that are configured by means of SDN technologies. In this section will be illustrated the 5G-PICTURE approach for supporting a flexible functional split.

5.5.1 SDN enabled routing and forwarding between RU and BBU

In 5G-PICTURE, the SDN technology can be applied in the FH for routing and forwarding the data stream between RU and BBU. As shown in Figure 42, the RU and BBU pools are connected to the SDN controller. The SDN controller sends a message to the RU and BBU for updating the rule of routing and forwarding the data in FH. For the SDN enabled routing and forwarding, the RU and BBU run applications for communication with the SDN controller through the control channel. A SDN agent running on the top of RU and BBU receives and sends the message from and to the controller to update the rule and to collect information for the configuration of routing and forwarding.

5.5.2 In-Band E-CPRI for enhanced synchronization

CPRI has been introduced to enable the communication between RRUs and BBUs in the FH network. The CPRI is expected to be deployed for the data communication in the FH split into the RRUs and BBUs in 5G network. However, recently eCPRI that is encapsulated CPRI in Ethernet frame is becoming a more attractive solution due to cost effective and greater reconfigurability. Ethernet frame is a common form of packets in network not only for data but also for control channel. The eCPRI can exploit the existing network infrastructure without using a dedicate equipment to support CPRI frame. In general, the CPRI protocol comprises of user data, control and management, and synchronization channels. The eCPRI encapsulates each of the channels into Ethernet format. However, the eCPRI has an issue to synchronize the control and management data with the user data due to transmitting in separated channels between RRUs and BBUs. In 5G-PICTURE, the In-Band eCPRI is proposed to address the issue to provide better synchronization. The In-Band eCPRI can form the Ethernet frame with IP headers. The payload of the In-Band eCPRI contains not only the user-data but also the control information data for RU in the header of the payload. The control data in In-Band eCPRI will be added into the payload if necessary. Therefore, the user-data of In-Band eCPRI can always be processed after applying the control information into RU before processing the user-data.

In 5G-PICTURE, the FPGA platform is used to implement the In-Band eCPRI processing units to improve the synchronization of data and control processes between RRU and BBU. Figure 42 illustrates the processing units of RRU and BBU for the In-Band eCPRI. The Figure shows the transmission process from BBU to RRU.

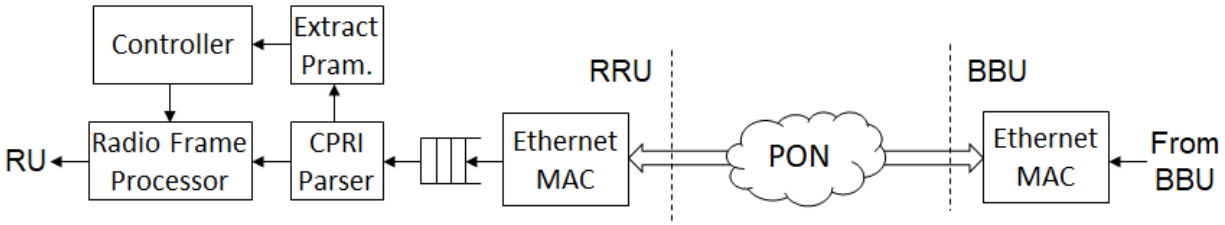


Figure 42. In-Band eCPRI protocol processing unit.

The In-Band eCPRI module consists of eCPRI parser, parameter extractor, controller and radio frame processor. When the In-Band eCPRI packet arrives at RRU, the parser parses the packet to divide into user data and control information. While the user data is buffered in the radio frame process, the extractor pass the parameter into the controller. The controller will update the radio frame processor with the extracted information and trigger to start the radio frame processor for sending the user data to the RU.

5.5.3 C-RAN functional split as programmable network function

As mentioned beforehand, the RAN programmability aims to treat the RAN as disaggregated RAN modules with functional splits in between. A generic RAN module architecture and available modules packaged as an entity is shown in Figure 43. It includes the northbound and southbound interface with the programmable data plane and in-band configuration. Also, several static (configuration file) and dynamic (FlexRAN and Operation/Business Support System (OSS/BSS)) management interfaces are revealed. These interfaces enable the considered software-based RAN module to be programmed over the both control and data plane processing as we mentioned in the Section 1.3. Note that the FlexRAN can delegate the control decision to be made at the RAN runtime over the RAN module as mentioned beforehand.

Based on aforementioned RAN module, we further consider the programmability over the functional split of C-RAN topology. As mentioned in the Deliverable D4.1 [22], such technical component aims to utilize both in-band control (between RU and DU) and out-band control (between DU and CU) to dynamically change the functional split in between. Further, it will under the control of a centralized FlexRAN controller over the involved RAN entities. For instance, the split between RU and DU is in-band re-configured by the DU and such reconfiguration is under the management of a centralized controller in order to properly maintain the RAN service toward corresponding users via handover, data buffering, etc.

However, such functional split dynamicity shall follow the capabilities of both end-point (e.g., CPU, memory) and connected interface (e.g., FH capacity). For instance, the FH throughputs of LTE system in 10 MHz radio bandwidth with two different functional splits are measured in Figure 44 (Split A corresponds to the 3GPP option 8 and Split B corresponds to 3GPP option 7-1). We can see that the FH throughput shall be larger than 500 Mbps when applying Split A option; however, less than 300 Mbps is required for Split B. Such requirement can be reduced via applying the sample compression [28], i.e., a factor of 2 is seen when applying the A-law

compression. Moreover, the required number of CPU cores is also measured based on the Intel i7 Sandy Bridge architecture in 3.2 GHz. The results in [28] shows that 2 CPU cores are required to deploy the RU for 10MHz radio bandwidth in Split A and 3 cores for Split B. To sum up, the split reconfiguration shall be aware of capability among involved entities and connections, and the controller can only program the applicable functional split changes.

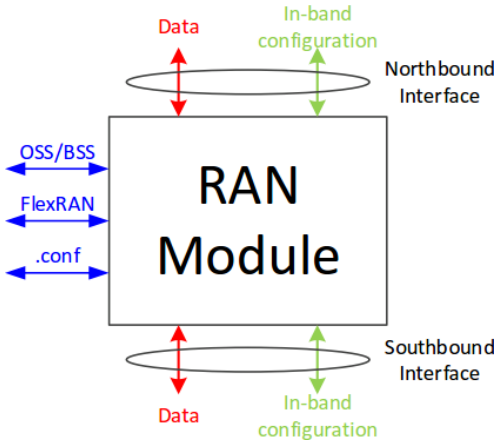


Figure 43: Generic Interface Ports for OAI Entities.

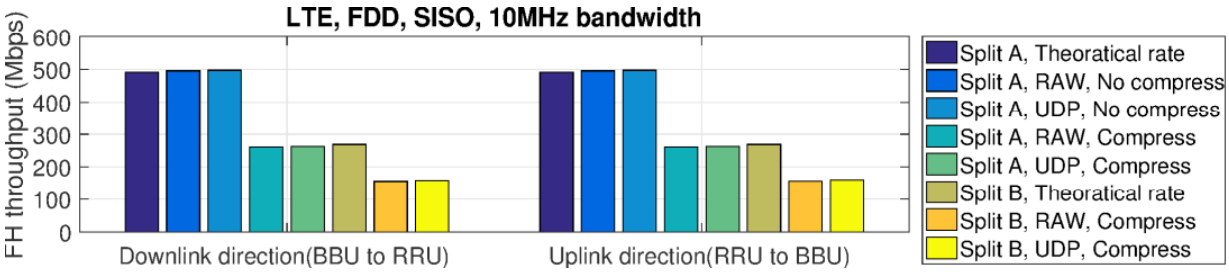


Figure 44: Fronthaul throughput of 10 MHz radio bandwidth of two functional split.

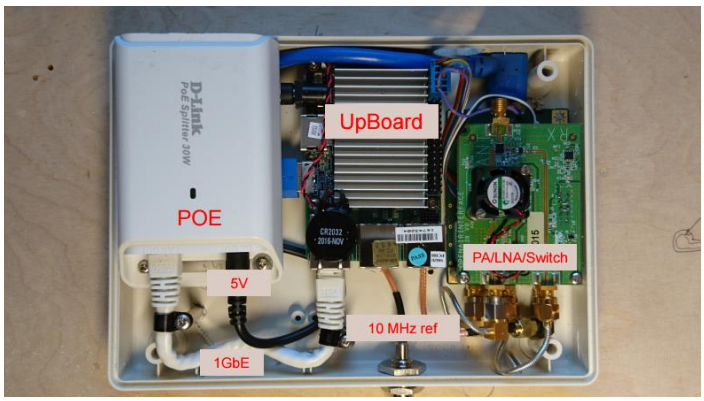


Figure 45: RU prototype.

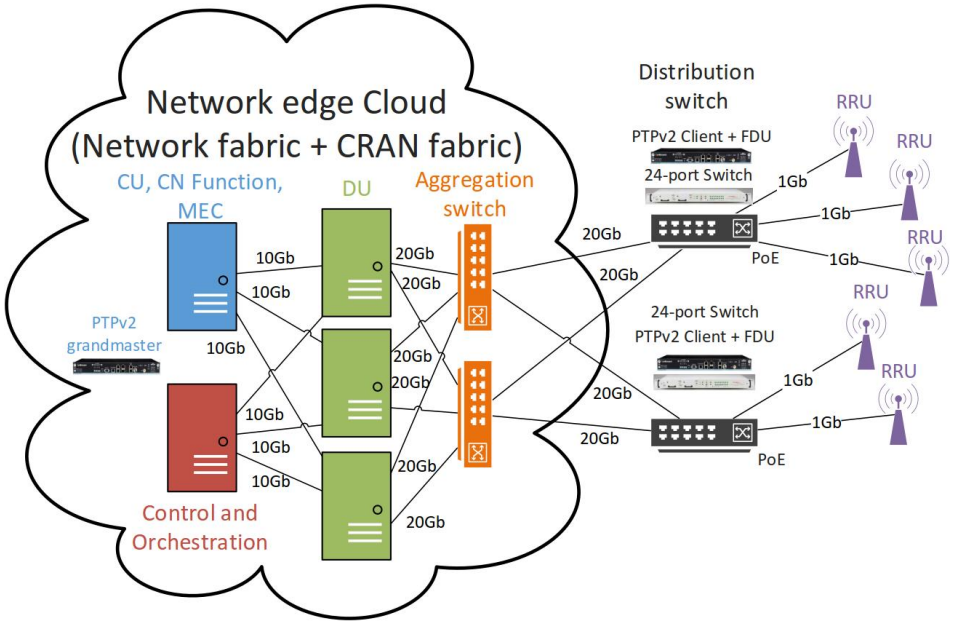


Figure 46: Logical C-RAN deployment example.

Based on the aforementioned measurement statistics, we come up with the RU prototype deployed at the EURECOM in Figure 45. An RU prototype contains the Pico-ITX or other smaller motherboard (e.g. UpBoard with Intel Atom quad-core processor), Power over Ethernet (PoE+) to support power supply, wiring for 1 Gigabit/sec Ethernet, RF front-end components (PA, LNA, Switch), 10 MHz/PPS frequency synchronization cable, baseband-to-RF radio unit (e.g., USRP B200-mini) and RF front-end circuits. The control of changing the functional split is in-band controlled through the Ethernet interface by the corresponding DU as stated in Section 3.5 of deliverable D4.1. Note that as DU and CU are logically centralized, and hence they can be deployed over the GPPs (or some specialized hardware for low-latency processing) among the centralized computing farm.

In Figure 46, we present the logical C-RAN deployment following the aforementioned three-tier disaggregated RAN manner. Each RU is connected with individual 1 Gigabit/sec Ethernet that are aggregated and multiplexed in the distribution switch. These aggregated switches provide the PoE capability for each connected RUs. The 2-tier network edge cloud is composed of network fabric (i.e., core network, MEC entity) and C-RAN fabric (i.e., CU, DU) using densely-deployed commodity servers which execute the OAI RAN software, CN function, control and management applications. Moreover, the DU is connected through 20 Gigabit/second FH network to the aggregation switch and provides the timing reference as the precision time protocol (PTP) grandmaster. Further, the CU is connected with DU through 10 Gb/s link and provides some core network functions and can be jointly deployed with the MEC server to provide dataplane programmability.

5.5.4 DSP and Layer-1 Functions Integrated into Radio Units

The introduction of massive MIMO [42] with several dozens or even hundreds of antenna elements renders current CPRI-based C-RAN architectures infeasible due to the extremely high data rates required on the CPRI fronthaul (see, e.g. [44]). To mitigate this, it necessary to include parts of the digital signal processing (DSP) of the baseband directly into the remotely deployed radio units. Especially an integration of layer-1 functions offers to dramatically reduce the required transport capacity. Effectively, the RRH and the distributed unit (DU) become a single entity, including complete analogue and RF processing, as well as partial baseband functionality. This will be referred to as an Active Antenna Distributed unit (AADU) to differentiate from DUs or RRHs with passive antennas, as currently used. Within 5G-PICTURE, the architecture of such AADUs is investigated and a corresponding hardware platform will be developed. This will be described in the following sections.

5.5.4.1 AADU Architecture

Figure 47 shows the high-level architecture of an AADU. It can be differentiated into several radio sub-units (RSUs) and an interface sub-unit (ISU). The RSUs are responsible for per antenna processing, which can include analogue and RF processing (filtering, amplifiers, ADC/DAC), calibration, and partial digital baseband processing, as well as local power distribution and an interface towards the ISU. The ISU performs all joint processing, including partial baseband processing, control & management, and an interfacing towards the CU.

The focus in 5G-PICTURE will be on how to distribute the baseband processing functionality among CU, ISU and RSU, which will be discussed in detail in the next subsection.

The AADU uses a modular approach, by utilizing several identical RSUs to compose the antenna array. This has several advantages:

- Different antenna configurations/form factors (16x4, 8x8) can be built relatively quickly.
- Analogue components (PAs, LNAs, ADCs/DACs) are close to antenna element location, which avoids long RF routing distances.
- Per-antenna processing can be performed on distributed hardware, reducing processing requirement per RSU.
- The data rate between the RSU and ISU can be reduced, as each link transports data of only a sub-set of the antennas.

5.5.4.2 AADU Functions and Processors

The physical layer of 4G and 5G radio technology is the most computational complex of the overalls tasks. By including parts of it in the AADU, both the computational requirements of the CU and the FH data rates can be dramatically reduced. However, it is still an open question which parts of the PHY should be ideally placed at the AADU. Figure 48 shows the typical processing steps of the downlink of a PHY layer. In addition, the functional splits according to 3GPP terminology are indicated. One additional split is indicated as 7.A. The difference of this split lies in the specific beamforming performed by the AADU. In general, there are two options for beamforming: time-domain based, right before A/D conversion, or frequency-domain based, before resource element mapping.

The TD beamforming has the advantage, that it can be performed after the full 3GPP PHY layer processing and is hence more independent. In addition, in only converts the beam data to per-antenna data right before D/A conversion, hence limiting the interconnect data rate (see also Sec. “Interconnect Architecture”). However, it does not per-subband or per-subcarrier beamforming. Also, channel estimation cannot be performed per antenna, hence requiring beam-sweeping or similar approaches for beam selection.

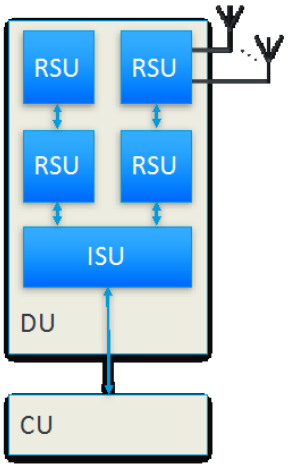


Figure 47: AADU overall architecture.

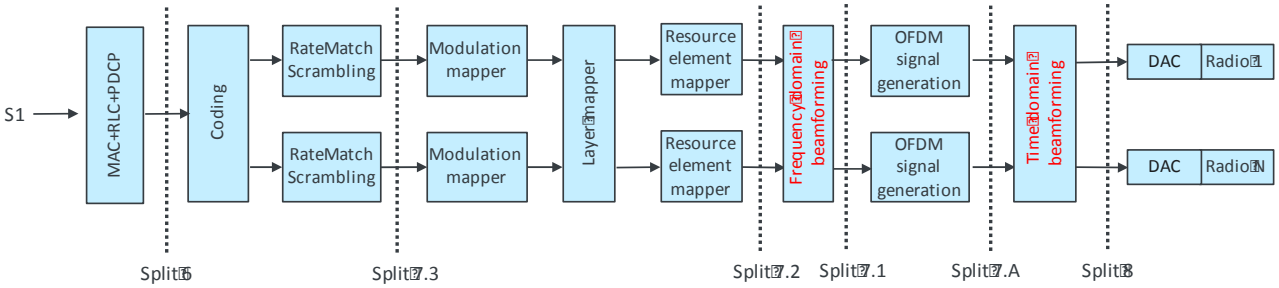


Figure 48: Physical layer processing chain.

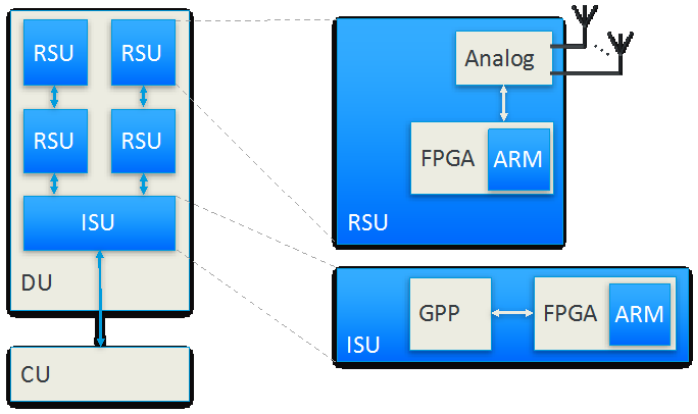


Figure 49: AADU hardware architecture.

FD beamforming in comparison offers full freedom in beamforming and allows for per-antenna channel estimation. However, it needs to be incorporated into the overall 3GPP stack and required the aggregation of per-antenna streams, which increases interconnect data rates.

Both approaches will be considered in the following and an optimal selection will be made within the scope of 5G-PICTURE.

In principle, all of the PHY layer processing functions can be performed on GPP hardware [45]. However, this is currently usually limited to a few spatial layers and a single 20 MHz carrier. With the introduction of up to 32 layers in LTE Rel. 14, carrier aggregation of several 20 MHz carriers or larger carrier bandwidth in 5G NR, PHY processing will be very challenging on GPPs. At the same time, the processing functions on the PHY are usually fixed and require little flexibility or programmability. Accordingly, hardware accelerators such as FPGAs will be used in the AADU, which offer a good compromise between programmability and power efficiency. In addition, ARM-based GPP can be synthesized or hard-coded into FPGAs to allow for a flexible programming for less computational tasks. Depending on the functional split between AADU and CU, the AADU can either include only FPGAs, or additional GPP processors with the FPGAs serving as accelerators. Figure 49 shows the principal hardware components for an AADU with integrated layer-1 DSP functionality.

5.5.4.3 AADU Functional Split

The chosen functional split has a strong impact on the design of an AADU, as it both determines the processing capabilities as well as the interface requirements. In addition to the usually functional split between CU and DU, the proposed AADU architecture incorporates an additional, inter-CU functional split between ISU and RSU. While having no impact on the transport network architecture, it still needs to be considered for the overall design.

Figure 50 shows three different functional split options under consideration for the AADU. IN the following the corresponding characteristics are listed.

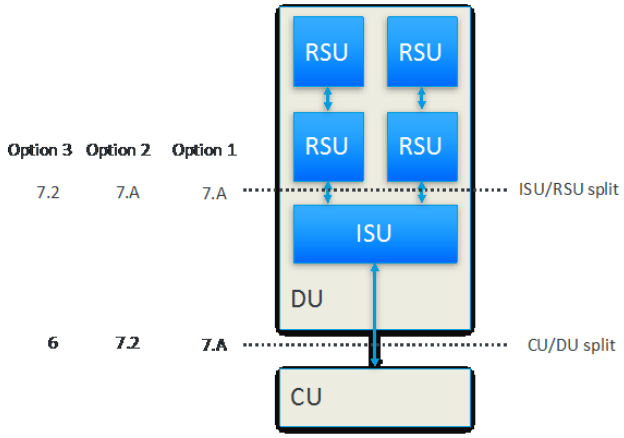


Figure 50: AADU functional split options.

Option 1:

- TD beamforming in RSU, ISU serves only as interconnect, remaining PHY processing on CU.
- High fronthaul and ISU/RUS data rate but reduced compared to per-antenna transport.
- Limited processing capabilities in AADU (TD beamforming only).

Option 2:

- TD beamforming in RSU, partial PHY processing on ISU, remaining PHY processing on CU.
- Higher computational requirements for ISU due to FFT/IFFT.
- Beamforming weights and pilots need to be transferred between CU/DU.
- Further reduced FH data rate.

Option 3:

- Partial PHY processing in RSU, remaining PHY processing on ISU.
- Higher computational requirements for RSU due to FFT/IFFT.
- Possibility to perform FD beamforming.
- Higher computational requirements for ISU (full PHY processing).
- GPP+FPGA required in ISU.
- Low FH data rate.

5.5.4.4 Interconnect architecture

Coupled to the functional split is the interconnect architecture in the AADU, which also has an impact on the interconnect data rates. Three options can be considered, which are depicted in Figure 51: daisy chain, star, or column-wise interconnect. In the daisy chain architecture, data from one RSU is passed to the next and only one RSU is directly connected to the ISU. In this case, the final RSU/ISU interface has to carry the data stream of all RSUs. While this has no impact for split option 1 and 2, where the beamforming is performed at the RSU, it could effectively quadruple the interface data rate for Option 3.

In contrast, in the star architecture, each RSU is directly connected to the ISU. This limits the data rate on each individual interface. However, it has the disadvantage of longer routing lengths, and the ISU still having to transmit and receive the full data rate.

Finally, the column architecture is a compromise of the former two options, combining daisy chaining between different rows of RSUs while using a star architecture for different columns.

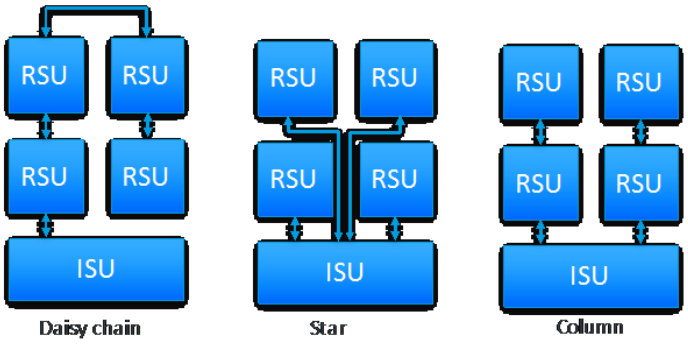


Figure 51: AADU interconnect options.

To compare the different split and interconnect options, Table 3 gives the corresponding required data rates on the two interfaces, the fronthaul from CU to DU, as well as the AADU-internal ISU/RSU data rate.

The assumptions are as follows:

- 64 antennas on 4 RSUs→16 antennas per RSU.
- 8x8 antennas, 4x4 RSUs.
- 2x20 MHz LTE carrier.

- 16 beams.
- 32 bit time domain I/Q samples, 16 bit frequency domain I/Q samples.
- 256 QAM.
- 25% protocol overhead.

Table 3: Data rate for different AADU options.

Data rate in Gb/s	Option 1		Option 2		Option 3	
	RSU/ISU	DU/CU	RSU/ISU	DU/CU	RSU/ISU	DU/CU
Star	39.3	39.3	39.3	10.8	10.8	5.4
Daisy chain					43.0	
Column					21.5	

5.6 MIMO at mmWave

To meet the high demands in terms of data rates of upcoming 5G access networks, the capacity of current wireless networks must increase. There is the aim of 1000x increase in data traffic predicted by 5G till 2020 [52]. MmWave is seen as a key enabling technology for achieving high-data rates and high capacity in 5G. High data rates are achievable due to large amount of unutilized bandwidth available in the mmWave spectrum. On the other side, high free space path losses are coupled with mmWave frequencies. Nevertheless, small wavelengths allows design of large antenna arrays which can be packed in a small form factor. In this way, mmWave systems can achieve sufficient array gain to provide sufficient link budget. Propagation measurements done at 28, 38, 60 and 73 GHz showed the feasibility of mmWave communication in outdoor scenarios [53][54]. Although several frequency bands are being investigated, 60 GHz is of special interest due to its unlicensed nature and the presence of solutions already available on the market. The best known are solutions for local and personal area networks according to standards IEEE 802.11ad/WiGig [55] and IEEE 802.15.3c [56], respectively. Those standards treat only single stream transmission relaying only on RF analogue beamforming, as depicted in Figure 52.

To support very high data rates (tens of Gb/s), it is necessary to have multistream MIMO transmission (spatial multiplexing). An example of conventional MIMO approach used in the Sub-6 GHz band is shown in Figure 53.

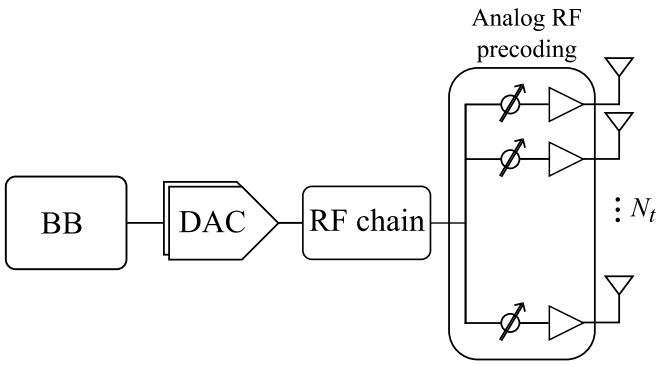


Figure 52: RF analogue beamforming mmWave MIMO system supporting single stream transmission.

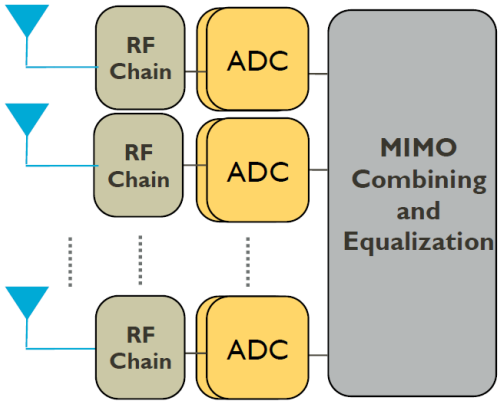


Figure 53: Conventional MIMO where all the signal processing is done in digital domain.

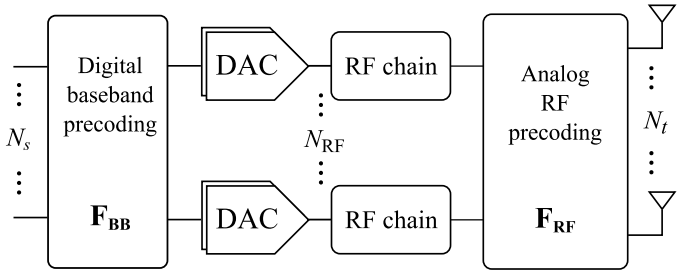


Figure 54: Hybrid precoding transmitter architecture.

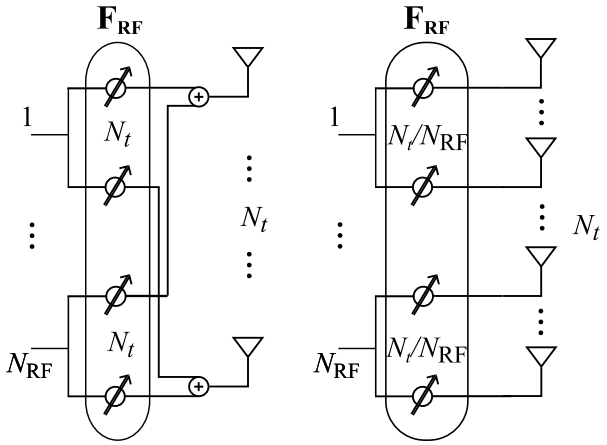


Figure 55: RF Analogue beamforming structures: fully-interconnected structure (left), partially-interconnected structure (right).

In the conventional MIMO approach, all MIMO processing is done in digital domain. Each antenna has dedicated RF chain with Digital-to-Analogue Converters (DACs) or Analogue-to-Digital Converters (ADCs). The use of the traditional MIMO approach in mmWave systems with Large Aperture Arrays (LAAs), is not practical. Such solution would introduce high hardware costs and increase the energy consumption.

For that reason, a hybrid beamforming solution for MIMO at mmWave, shown in Figure 54, is proposed. This solution offloads some of MIMO processing to the analogue domain, and represents a trade-off in power consumption and hardware complexity. In the hybrid beamforming architecture, the sharp beams are formed with an analogue beamforming (phase shifters) stage which compensates for the large path loss at mmWave bands; and the digital beamforming provides the necessary flexibility to perform advanced multi-antenna techniques such as spatial multiplexing and/or multi stream transmission. Depending on the structure of the RF analogue beamforming stage, fully-interconnected and partially-interconnected structures are proposed in the literature, as depicted in Figure 55. In the former, the signal at the output of each RF chain is connected to all

antennas through the network of phase shifters, while, in the latter, each RF chain is connected to an antenna subset [57].

A fully-interconnected structure provides higher beamforming (BF) gain as it utilizes all phase shifters and full antenna array and has more degree of freedom in RF domain and it outperforms the partially-interconnected one. However, it is argued that a fully-interconnected structure is not realizable in practice, being the latter (using subarrays) is much more favourable.

5.6.1 LoS MIMO at mmWave frequencies

In addition to hybrid beamforming, mmWave MIMO architectures such line-of-sight (LOS) MIMO at mmWave frequencies is attracting research interest [58][59]. LOS mmWave MIMO systems are actually interesting for applications such as fixed distance BH links. A generic scheme of LOS MIMO system is shown in Figure 56.

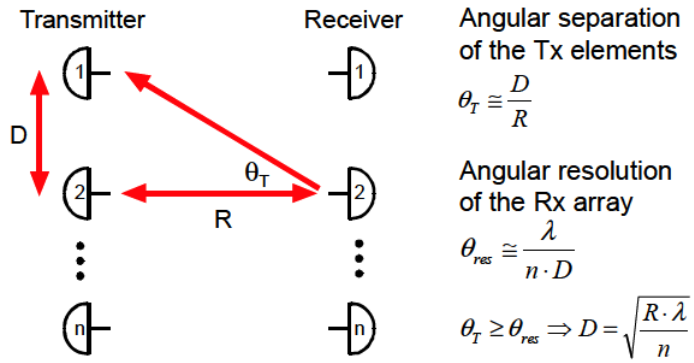


Figure 56: LOS MIMO system [58].

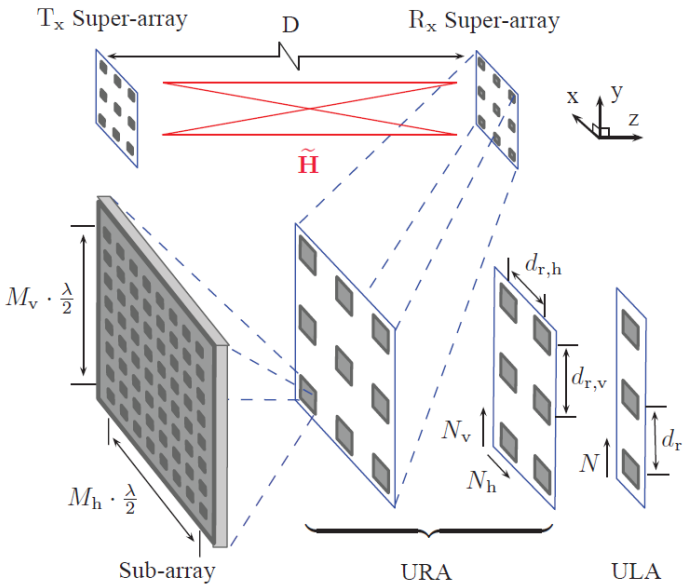


Figure 57: LOS MIMO system with super-arrays [59].

LOS MIMO systems are analysed using the principles of diffractions-limited optics (image theory). Multiple parallel data streams are possible utilizing specific spacing between antenna elements which is correlated with link range. The relationship which relates antenna elements spacing *D* and the distance between transmitter and receiver *R* is as follows

$$D = \sqrt{\frac{R\lambda}{n}},$$

where λ denotes the carrier wavelength (5 mm at 60 GHz) and *n* represents the number of antenna elements at the transmitter and at the receiver (assuming equal number). Following the above equation, a 4×4 (16-

element) square antenna array of dimensions *nD*×*nD* = 3.4×3.4 meters would operate at 1 km link range at 60 GHz. In addition, at 200 m link range, the same antenna array would have the size of *nD*×*nD* = 2×2 meters. Considering single polarization, 2 GHz of channel bandwidth and 2.5 Gb/s data rate per channel or antenna pair (for example, MCS 9 in IEEE802.11ad), such system could support 40 Gb/s aggregate data rate. In principle, this system architecture is scalable to larger antenna arrays supporting super high data rates. In a 60 GHz LOS MIMO system is proposed using the so-called super antenna arrays which combine both beamforming and spatial multiplexing, as depicted in Figure 57. Such systems are capable of achieving hundred Gb/s aggregated data rates at small cell distances.

5.6.2 RF front-ends for MIMO at mmWave

In the 5G-XHaul project we have developed analogue-front end solution with RF beamforming [48][49]. Our solution is flexible and modular relying on two integrated circuit (ICs), beamformer and up-/down conversion modem. This allows different RF beamforming architectures at mmWave, such as massive MIMO arrays or hybrid beamforming MIMO with subarrays. Exemplary solutions are shown in Figure 58. Examples of the AFE boards developed in the 5G-XHaul project are shown in Figure 59, being the RF board shown on the left the one consisting of beamformer and antenna array. This board can be used with different off-the-shelf 60 GHz modems. On the right hand side, the complete 60 GHz solution with integrated antenna, 60 GHz beamformer and up/down converter is shown.

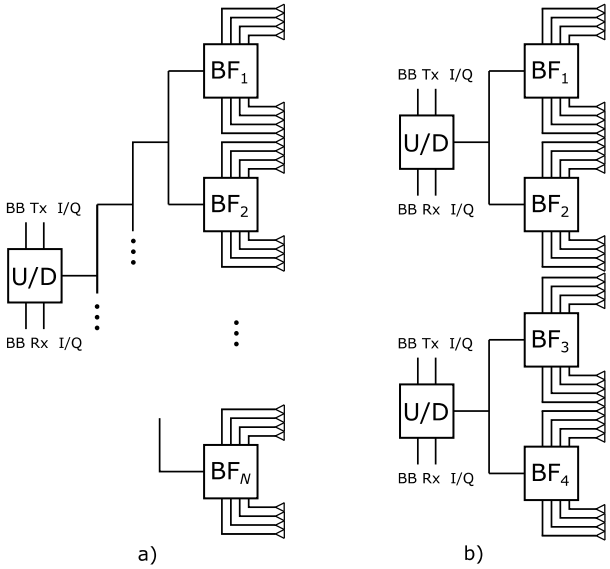


Figure 58: Examples of different mmWave beamforming architectures: a) massive mmWave array, b) hybrid beamforming mmWave MIMO architecture with subarrays.

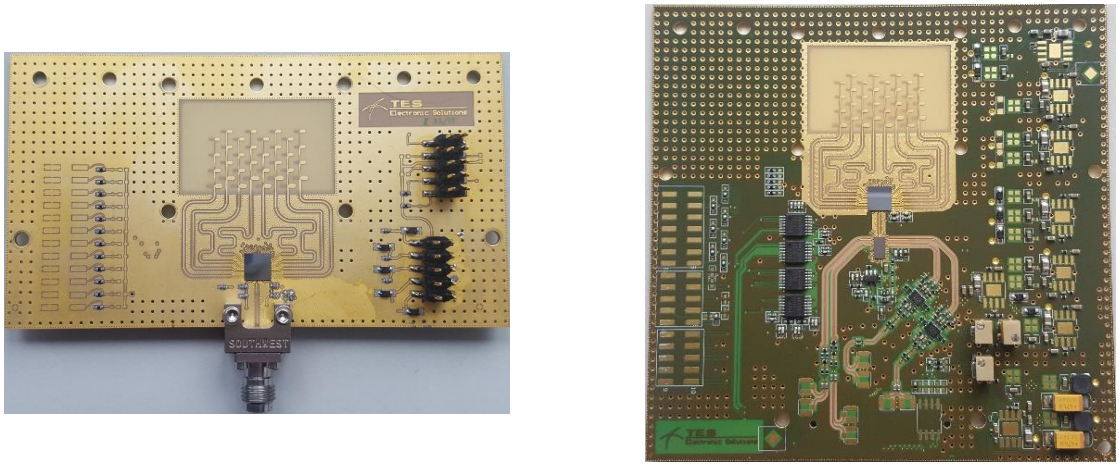


Figure 59: 60 GHz RF front-ends: RF board with beamforming to be used with off-the-shelf up/down converter (left) and complete RF front-end board (right).

The solution shown in Figure 58.a) can be used for designing a single antenna with high gain and narrow beam of the overall super-array.

In 5G-PICTURE there is no plan to design these complex Analogue Front-End (AFE) solutions but to leverage the existing ones (like those in Figure 59) for research on MIMO solutions, i.e. LoS mmWave MIMO or for a subarray-type hybrid beamforming MIMO system. More details on the concept and possible implementation of MIMO using these boards will be included in deliverable D3.2.

5.7 Interfaces - Multi-Protocol / Multi-PHY interfacing functions (MPIs)

Nowadays, current networks are fragmented and interconnected using proprietary interfaces. It is also a fact that current architectures are vendor-specific architectures. These issues raise many inter-operability challenges. In 5G the aim is to relax these constraints towards a harmonized network with open, configurable interfaces, to leverage the benefits of the diverse set of implementations.

The large variety of technologies present in 5G has very different characteristics including rates of operation spanning from few Mbps up to several Gb/s and adopts a wide range of protocols and technology solutions including PCIe 4.0, USB 3.1, 1G/10G/40G/100G Ethernet, SFI/XFI, SATA, Q/SGMII and CPRI. As the different technology domains adopt different protocol implementations and provide very diverse levels of capacity granularity, etc., interfacing is critical.

In 5G-PICTURE there exists a plethora of technologies and implementations, bringing along diverse challenges to the consortium when setting up the interfaces between them. These interfaces and associated functions will enable the required integration across technology domains, providing an integrated transport solution. It is therefore of utmost importance to analyse and define the architecture and requirements of Multi-protocol/Multi-PHY Programmable Network interfaces. This work falls within Task 3.3 in WP3, where the physical interconnections and associated programmability will be brought.

5G-PICTURE will leverage of fast Multi-PHY and Multi-Protocol interfaces based on state of the art FPGA allowing great flexibility to mix and match a variety of protocols and technology solutions. To that end, we pursue the development of high speed low cost energy efficient serializer/deserializer (SerDes) interfaces based on FPGAs. These interfaces will facilitate the mapping of different QoS classes across different domains. The output of these interfaces will be used as input to the Open Packet Processor (OPP), to support stateful packet/flow processing related tasks.

The objectives of the MPIs are:

- to enable the support to heterogeneous networks,
- to enable mapping of traffic across infrastructure domains,
- to enable the selection of the optimal timing paths depending on the synchronization requirements, and
- to enable end-to-end optimisation.

The functionalities being brought by these interfaces to the edge nodes encompass traffic adaptation, protocol mapping, etc.

5.7.1 Initial design for optical edge nodes

One example of the functionality of the MPIs is shown in Figure 60, where HW programmability features at the edge nodes support a variety of multi-protocol/PHY interfaces which allow mapping of very different traffic streams coming from the wireless access domain to optical frames/streams, capitalising on a number of HW programmable building blocks including:

- Open Packet Processor (OPP), and
- sliceable bandwidth variable transceivers (BVTs)

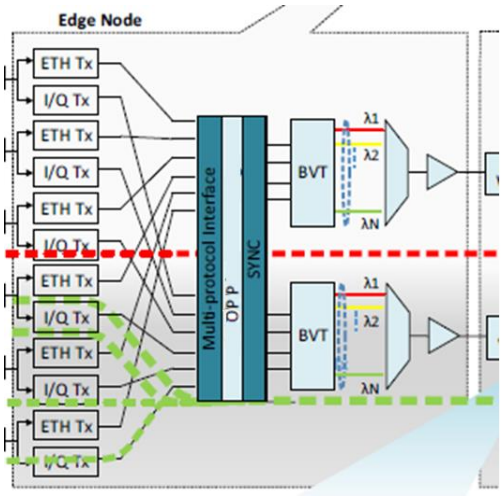


Figure 60: Edge node equipped with HW programmability features.

The MPI takes as input OPP frames/streams coming from network nodes with different PHY/protocol characteristics, convert and forward them to an OPP node. The OPP performs additional encapsulation/decapsulation operations for packet level interfacing, select the per-flow allocations of the incoming streams and mark the packets to send to the BVTs. The BVTs will utilize FPGAs, advanced Bandwidth Variable (BV) optical cross-connects and fast optical switching modules to enforce the allocation of variable size-spectral/time slots, required to support services with continuous channel allocation at various bit rates (i.e. heavy and light CPRI) and services with sub-wavelength time slot allocation (Ethernet flows).

5.7.2 Traffic adaptation at lower layers

In 5G environments supporting a large variety of protocols there is a clear need to support different traffic types, e.g., continuous stream of data (e.g., CPRI flows for fronthaul services) and packet (e.g., Ethernet and IP frames). As an example, streams coming from the wireless access domain will be converted to optical frames/streams (Figure 61).

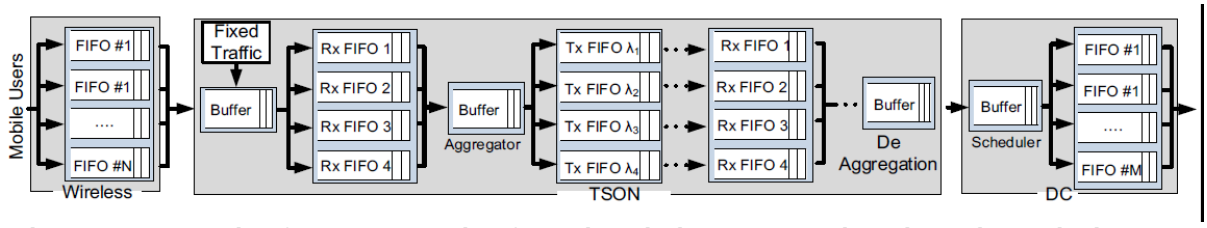


Figure 61: Example of streams coming from the wireless access domain to the optical transport domain.

5.7.3 Synchronization

The 5G-PICTURE architecture encompasses different technologies, which must be integrated across technology domains, e.g. wireless/optical/packet. 5G-PICTURE will investigate novel approaches to deliver high accuracy synchronisation in such heterogeneous environment. It is a task of WP3 to define and develop fast Multi-Link/PHY interfaces to enable this level of integration. The different domains adopt different protocol implementation and provide very diverse levels of requirements. WP4 synchronization functions will make use of the programmable infrastructure building blocks and abstractions made available by WP3. Figure 62 depicts the initial architecture of the expected implementation, where the interfacing boards at the edge, which inherently should offer SyncE and 1588 support, will be able to offer synchronization capabilities that can be shared by the different technology domains attached to it.

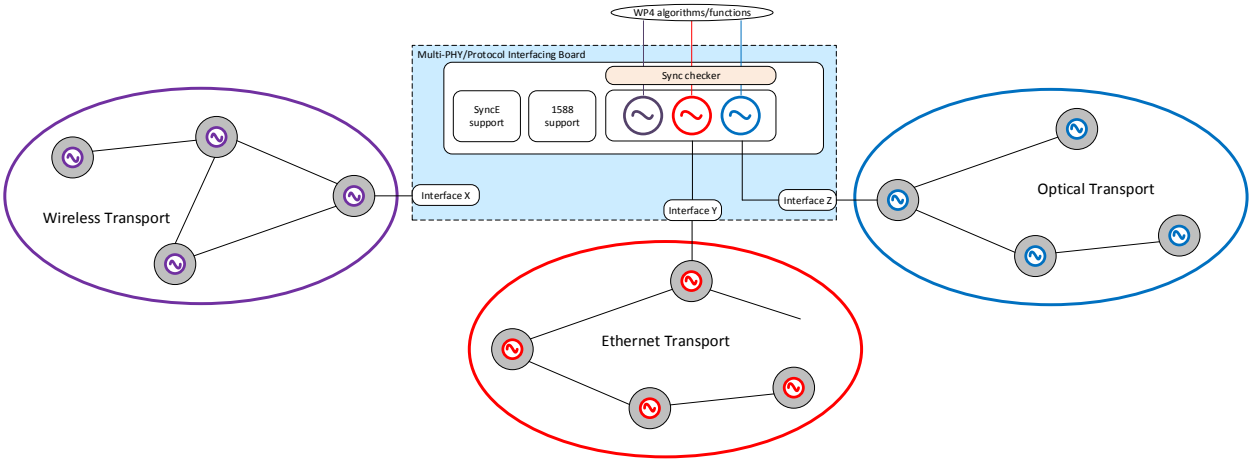


Figure 62: Multi-PHY/Multi-Protocol Interfacing solution enabling synchronization across technology domains.

6 Summary and Conclusions

This document has presented the initial definition of data plane programmability and infrastructure components developed in WP3 of the H2020 5G-PICTURE project, providing a state-of-the-art overview of the programmability of 5G network elements, and describing the development platforms that are used by the project partners to implement the new technologies of WP3.

In particular, the deliverable exposed the various target hardware platforms to develop the various programmable network platforms that will be developed and presented the initial functional definitions of the programmable platforms. The set of methodologies selected to abstract the programmable network platforms have been individuated and discussed in section 4. Finally, section 5 illustrated the different hardware technologies developed in 5G-PICTURE that provide basic building blocks of the 5G network architecture.

The deliverable proposes first specifications of those technologies. These specifications will be revised according to the ongoing work among the three parallel tasks 3.1, 3.2 and 3.3 of WP3. The revised and definitive detailed description of 5G-PICTURE data plane programmability and infrastructure components will be provided in November 2018 with deliverable D3.2.

7 References

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: enabling innovation in campus networks. ACM SIGCOMM Computer Communication Review, vol. 38, no. 2, pp. 69-74, 2008.

[2] M. T. Arashloo, Y. Koral, M. Greenberg, J. Rexford, and D. Walker. SNAP: Stateful Network-Wide Abstractions for Packet Processing. Proc. of the 2016 ACM SIGCOMM conference, pp. 29-43.

[3] Open Networking Foundation. OpenFlow Switch Specification version 1.5.1 (protocol version 0x06). ONF TS-025, Mar. 2015. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.1.pdf

[4] P. Bosshart, et al. "P4: Programming protocol-independent packet processors." ACM SIGCOMM Computer Communication Review 44.3 (2014): 87-95.

[5] R. Ozdag, R. Intel® Ethernet Switch FM6000 Series-Software Defined Networking. Available at http://www.intel.com/content/www/us/en/ethernet-products/switch-silicon/ethernet-switch-fm6000-sdn-paper.html

[6] H. Song. Protocol-oblivious forwarding: Unleash the power of SDN through a future-proof forwarding plane. In Proc. of the 2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, HotSDN '13, 2013, pp. 127-132.

[7] The P4 language Consortium. The P4 Language Specification, version 1.0.2. March 2015. Available at http://p4.org/wp-content/uploads/2015/04/p4-latest.pdf

[8] The P4 language Consortium. The P4_16 Language Specification, version 1.1.0. January 2016. Available at https://p4.org/p4-spec/docs/P4-16-v1.0.0-spec.pdf

[9] S. Yan et al., “Multilayer network analytics with SDN-based monitoring framework”, in IEEE Journal of Optical Communications and Networking, vol. 9, 2017, pp.A271-A279

[10] A. Napoli et al., “Next Generation Elastic Optical Networks: the Vision of the European Research Project IDEALIST”, IEEE Communications Magazine, vol. 53, no. 2, 2015

[11] N. Sambo et al, “Next Generation Sliceable Bandwidth Variable Transponders”, IEEE Communications Magazine, vol. 53, no. 2, pp. 163-171, Mar. 2015

[12] A. Tzanakaki et al., “5G infrastructure supporting end-user and operational services: The 5G-XHaul architectural perspective”, IEEE International Conference on Communications (ICC 2016), May 2016.

[13] M. Yang et al., “OpenRAN: a software-defined ran architecture via virtualization,” in ACM SIGCOMM computer communication review, vol. 43, no. 4. ACM, 2013, pp. 549–550.

[14] I. F. Akyildiz et al., “Softair: A software defined networking architecture for 5g wireless systems,” Computer Networks, vol. 85, pp. 1–18, 2015.

[15] A. Gudipati et al., “SoftRAN: Software defined radio access network,” in Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking. ACM, 2013, pp. 25–30.

[16] T. Chen et al., “SoftMobile: control evolution for future heterogeneous mobile networks,” IEEE Wireless Communications, vol. 21, no. 6, pp. 70–78, 2014.

[17] M. Bansal et al., “Openradio: a programmable wireless dataplane,” in Proceedings of the first workshop on Hot topics in software defined networks. ACM, 2012, pp. 109–114.

[18] W. Wu et al., “Pran: Programmable radio access networks,” in Proceedings of the 13th ACM Workshop on Hot Topics in Networks. ACM, 2014, p. 6.

[19] A. Gudipati et al., “Radiovisor: A slicing plane for radio access networks,” in Proceedings of the third workshop on Hot topics in software defined networking. ACM, 2014, pp. 237–238.

[20] X. Foukas et al., “FlexRAN: A Flexible and Programmable Platform for Software-Defined Radio Access Networks.” in ACM CoNEXT, 2016, pp. 427–441.

[21] A. Checko et al., “Cloud RAN for mobile networks A technology overview,” IEEE Communications Surveys & Tutorials, vol. 17, no. 1, pp. 405–426, 2015.

[22] 5G-PICTURE, Deliverable D4.1: “State of the art and initial function design”, February 2018.

[23] 3GPP TR38.801 V14.0.0, Study on new radio access technology: Radio access architecture and interfaces (Release 14).

[24] N. Nikaein, “Processing radio access network functions in the cloud: Critical issues and modeling,” in Proceedings of the 6th International Workshop on Mobile Cloud Computing and Services, ACM, 2015, pp. 36–43.

[25] N. Yu, et al., “Multi-resource allocation in cloud radio access networks,” in Proceedings of Communications (ICC), 2017 IEEE International Conference on. IEEE, 2017, pp. 1–6.

[26] V. Q. Rodriguez and F. Guillemin, “Towards the deployment of a fully centralized cloud-ran architecture,” in Proceedings of the IEEE International Wireless Communications and Mobile Computing Conference (IWCMC), 2017, pp. 1055–1060.

[27] N. Nikaein and C.-Y. Chang, “Slicing and orchestration in service-oriented RAN architecture,” IEEE Software Defined Networks Newsletter, Dec. 2017.

[28] C.-Y. Chang, et al., “FlexCRAN: A Flexible Functional Split Framework over Ethernet Fronthaul in Cloud-RAN,” in Proceedings of Communications (ICC), 2017 IEEE International Conference on. IEEE, 2017, pp. 1–7.

[29] T. X. Tran, A. Younis and D. Pompili, "Understanding the Computational Requirements of Virtualized Baseband Units Using a Programmable Cloud Radio Access Network Testbed," 2017 IEEE International Conference on Autonomic Computing (ICAC), Columbus, OH, 2017, pp. 221-226.

[30] A. M. Mahmood, A. Al-Yasiri and O. Y. K. Alani, "A New Processing Approach for Reducing Computational Complexity in Cloud-RAN Mobile Networks," in IEEE Access, vol. PP, no. 99, pp. 1-1.

[31] T. Werthmann, H. Grob-Lipski and M. Proebster, “Multiplexing Gains Achieved in Pools of Baseband Computation Units in 4G Cellular Networks”.

[32] C. Desset, et al., “Flexible power modeling of LTE base stations”.

[33] R. Miao, et al. "SilkRoad: Making Stateful Layer-4 Load Balancing Fast and Cheap Using Switching ASICs," in Proc. ACM SIGCOMM, 2017.

[34] Z. Liu, et al. "One sketch to rule them all: Rethinking network flow monitoring with univmon", in Proc. ACM SIGCOMM, 2016.

[35] P. Bosshart, et al. "Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN." ACM SIGCOMM Computer Communication Review. Vol. 43. No. 4. ACM, 2013.

[36] G. Bianchi, et al. "Open Packet Processor: a programmable architecture for wire speed platform-independent stateful in-network processing." arXiv preprint arXiv: 1605.01977 (2016).

[37] S. Pontarelli, et al. "Stateful OpenFlow: Hardware proof of concept," in Proc. of the IEEE 16th International Conference on. High Performance Switching and Routing (HPSR), 2015.

[38] D. Zhou, B. Fan, H. Lim, M. Kaminsky, and D. G. Andersen. “Scalable, High Performance Ethernet Forwarding with CuckooSwitch”. In Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies, ACM CoNEXT ’13, pages 97–108. ACM, 2013.

[39] D. Fotakis, R. Pagh, P. Sanders, and P. Spirakis. Space efficient hash tables with worst case constant access time. Theory of Computing Systems, 38(2):229–248, Feb 2005.

[40] C. Cascone, R. Bifulco, S. Pontarelli, A. Capone, “Relaxing state-access constraints in stateful programmable data planes”, to appear in ACM SIGCOMM CCR 2018.

[41] S. Pontarelli, M. Bonola, and G. Bianchi. "Smashing SDN" built-in" actions: Programmable data plane packet manipulation in hardware." Network Softwarization (NetSoft), 2017 IEEE Conference on. IEEE, 2017.

[42] UE turbo decoding offloading to FPGA, [Online] https://gitlab.eurecom.fr/oai/openairinterface5g/wikis/how-to-offload-turbo-decode-to-fpga

[43] E. G. Larsson, O. Edfors, F. Tufvesson and T. L. Marzetta, "Massive MIMO for next generation wireless systems," *IEEE Communications Magazine*, vol. 52, no. 2, pp. 186-195, February 2014.

[44] J. Bartelt et al. "5G transport network requirements for the next generation fronthaul interface." *EURASIP Journal on Wireless Communications and Networking,* vol. 2017, no. 1, pp. 89, 2017.

[45] N. Nikaein, et al. "OpenAirInterface: an open LTE network in a PC." Proceedings of the 20th annual international conference on Mobile computing and networking. ACM, 2014.

[46] Small Cell Forum, “DOCUMENT 082.09.05, FAPI and nFAPI specifications”, May 2017.

[47] Implementation of Turbo-decoding LTE UE offloading, https://gitlab.eurecom.fr/oai/openairinterface5g/wikis/how-to-offload-turbo-decode-to-fpga

[48] 5G-XHaul deliverable D4.9, “Initial report on mm-Wave circuits and systems for high rate point to multipoint links”, December 2016.

[49] 5G-XHaul deliverable D4.10, “Final report on mm-Wave circuits and systems for high rate point to multipoint links”, March 2018.

[50] S.-Y. Lien, S.-L. Shieh, Y. Huang, B. Su, Y.-L. Hsu, H-Y. Wei, “5G New Radio: Waveform, Frame Structure, Multiple Access, and Initial Access”, IEEE Communications Magazine, vol. 55, no. 6, pp. 64-71, June 2017.

[51] C. Kilinc, J.F. Monserrat, M.C. Filippou, N. Kuruvatti, A.A. Zaidi, I. Da Silva, M. Mezzavilla, “New Radio 5G User Plane Design Alternatives: One 5G Air Interface framework supporting multiple services and bands”, IEEE Globecom Workshop, Washington DC, Feb. 2017.

[52] J. G. Andrews et al. ,”What Will 5G Be?”, IEEE Journal on Selected Areas in Communications, vol. 32, no. 6, pp. 1065-1082, June 2014

[53] T. S. Rappaport et al., ”Millimeter wave mobile communications for 5G cellular: It will work!”, IEEE Access, vol. 1, pp. 335-349, 2013.

[54] S. Rangan, T.S. Rappaport, E. Erkip, “Millimeter-wave Cellular Wireless Networks: Potentials and Challenges.” Proc. IEEE, vol. 102, no. 3, pp. 366-385, Mar. 2014.

[55] IEEE Standard 802.11ad,”Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 3: Enhancements for Very High Throughput in the 60 GHz Band”.

[56] IEEE Standard 802.15.3c, ”Part 15.3: Wireless Medium Access Control and Physical Layer Specifications for High Rate Wireless Personal Area Networks (WPANs)”.

[57] 5G-XHaul deliverable D4.4, ???

[58] C. Sheldon et al., “Spatial multiplexing over a line-of-sight millimeter-wave MIMO link: A two-channel hardware demonstration at 1.2Gbps over 41m range,” in 38th European Microwave Conference, October 2008

[59] 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, pp. 1-6.

[60] 5G-XHaul deliverable D4.2 “Optical Fronthauling Solution”, October 2017.

8 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
BBU	Baseband Unit
BH	Backhaul
BS	Base Station
BVT	Bandwidth Variable Transponder
BWT	Blu Wireless Technology
CAN	Controller Area Network
CAPEX	CAPital EXpenditure
CP	Control Plane
CU	Centralized Unit
C-RAN	Cloud Radio Access Network
DAC	Digital-to-Analogue Converter
DA-RAN	Dis-Aggregated Radio Access Network
DCB	Data Center Bridging
DDOS	Distributed Denial of Service
DL	Download
D-RAN	Distributed Radio Access Network
DU	Distributed Unit
ECN	Explicit Congestion Notification
EDP	European Deployment Plan
EFSM	Extended Finite State Machine
FDD	Frequency division Duplexing
FH	Fronthaul
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
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
ISU	Interface Sub-Unit
ITU	International telecommunication Union
LAA	Large Aperture Array
LTE	Long Term Evolution
LTE-A	Long Term Evolution Advanced
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
MU	Multiple-Unit
NB-IoT	Narrow Band Internet of Things
NFV	Network Function Virtualisation
OAI	OpenAirInterface
OBSAI	Open Base Station Architecture Initiative
OLT	Optical Line Terminal
ONU	Optical Network Unit
OPEX	OPerational EXpenditure
OPP	Open Packet Processor
PDCP	Packet Data Convergence Protocol
PDV	Packet Delay Variation
PHY	Physical
PMOD	Peripheral Module
PMP	Packet Manipulator Processor
PNF	Physical Network Functions
POE	Power over Ethernet
PON	Passive Optical Networks
PPP	Public Private Partnership
PRB	physical resource block
P2MP	Point-to-Multipoint
PTP	Precision Time Protocol
QoS	Quality of Service
RAN	Radio Access Network
RISC	Reduced Instruction Set Computer
RPC	Remote Procedure Calls
RRH	Remote Radio Head

RSRP	Reference Signal Receive Power
RSU	Radio Sub-Unit
RU	Radio Unit
SAI	Switch Abstraction Interface
SBC	Single-Board Computer
SDN	Software Defined Networks
SerDes	Serializer/Deserializer
SIL	server instrumentation library
SINR	signal-to-interference and noise ratio
SON	Self-Organizing Network
SO-RAN	service-oriented RAN
SW	Software
TCP	Transmission Control Protocol
TSN	Time Sensitive Network
TSON	Time Shared Optical Network
TTI	transmission time interval
UDP	User Datagram Protocol
UP	Upload
vBBU	Virtual Baseband Unit
VLIW	Very Long Instruction word
VLAN	Virtual Local Area Network
VNF	Virtual network function
WDM-PON	Wavelength Division Multiplexing Passive Optical Network