How will edge computing shape the 5G deployment? The hardware acceleration use case

PhD Thesis of:
Federico Civerchia

Sant'Anna
School of Advanced Studies – Pisa

Academic Year 2018/2019

International PhD Program in Emerging Digital Technologies

Curriculum: Embedded Systems

How will edge computing shape the 5G deployment? The hardware acceleration use case

Author
Federico Civerchia

Supervisor
Prof. Piero Castoldi

Correlator
Prof. Maxime Pelcat

Abstract

The recent improvements in the information technology will lead to the new era of the communication (5G) where everything will be connected, where smart and connected objects will be a constant presence in our daily life. Thus, stricter requirements in terms of communication bandwidth and latency have to be satisfied to meet the demands of a huge number of connected devices. Instead of the centralized approach, moving the data processing to the edge can improve the performance since it reduces the infrastructure-user round trip time and it saves Cloud bandwidth. However, this decentralization comes at a price though. Moving data computation and communication processing to the network edges improves system scalability and reliability but it requires more local hardware and only a subset of data is analyzed. This means that an edge system does not have global visibility of the information.

This thesis aims at presenting a novel approach to accelerate the 5G infrastructure at the edge. The idea is to exploit hardware acceleration to improve the processing of the protocol stack functionalities and network functions close to the final user. In this way, the bandwidth for the communication between 5G radio infrastructure and the central Cloud can be saved. Moreover, real time application can benefit from the improved computation capabilities by means of hardware offloading.

Considering the latest developments in the embedded systems in terms of computational power and lower hardware cost, we envision that edge computing can be exploited to improve 5G infrastructure. Thus, the edge computing is ready to be deployed in 5G architecture, improving the user experience.

To my grandfather and my family

A mio nonno e alla mia famiglia

Federico

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

In the new era, thought itself will be transmitted by radio.
Guglielmo Marconi, New York Times (11 Oct 1931)

1. From 1G to 5G

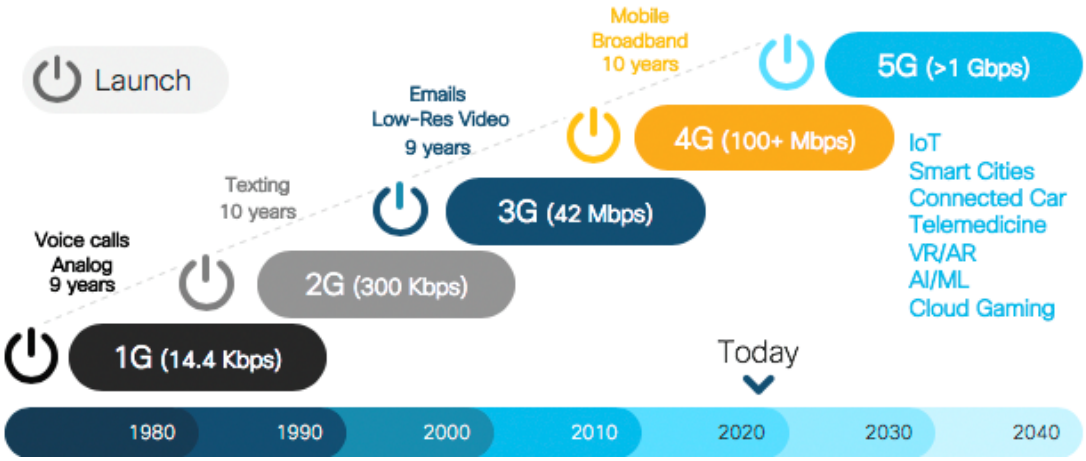


FIGURE 1.1. Evolution of the wireless network: from 1G to 5G [1]

Human technologies have never been so connected in the physical world. With the recent progress of the mobile/fixed communication, it will be possible to connect up to 7×10^{23} devices for each square metre of the hearth. This leads to the new era of the communication where everything will be connected, where smart and connected objects will be a constant presence in our daily life.
Things were significantly different in 1885, when Guglielmo Marconi – the Italian scientist considered the father of the radio communication – proposed the first wireless transmission in the history of science using telegraphs. Marconi could immediately understand the importance of his research, as he wrote in the paper *Wireless telegraphic communication* [**Mar09**]: "with regard to the utility of

[1] Source: Cisco VNI Global Mobile Traffic Forecast, 2017-2022

wireless telegraphy there is no doubt that its use has became a necessity for the safety of shipping, all the principal liners and warships being already equipped, its extension to less important ships being only a matter of time, in view of the assistance it has provided in cases of danger. Its application is also increasing as a means of communicating between outlying islands, and also for the ordinary purposes of telegraphic communication between villages and towns, especially in the colonies and in newly developed countries".

Since then, engineers and scientists were working on an efficient way to communicate by means of Radio Frequency (RF) waves. In particular, during 1970s, the invention of hand-held devices, capable of connecting with each other wirelessly, permitted to standardize the first generation of mobile phone (1G). 1G is based on analog transmission technology designed to provide basic voice service. This first concept of communication considered analog signals without data capabilities and the digital signalling was used to connect radio towers with the infrastructure via Frequency Modulation (FM) at around 150 MHz. Despite this important milestone, 1G technology suffered from a number of drawbacks. The coverage area was poor since the cell size was 2-20 km [**Sch03**, **Sto02**] each and sound quality was low. The few providers of the epoch operated with different frequencies without compatibility between each other thus, there was no roaming support. In addition, there was a lack of security since the analog signal cannot implement advanced encryption methods.

The second generation of mobile phone (2G), launched under the Global System for Mobile (GSM) communication in 1991, brought digitalization with respect to the previous generation. Indeed, it provided a digital system alongside the analog transmission. For the first time, calls were encrypted, the sound was significantly clearer with less noise in background and the cell coverage area increased up to 35 km. Also Short Messages Service (SMS) and Multimedia Messages Service (MMS) services were implemented to offer new functionalities for the end-users. Moreover, the allocation of the spectrum was optimized with modulation techniques that will be also used in the next generations such as Time Division Multiple Access (TDMA), Frequency Division Multiple Access (FDMA) and Code Division Multiple Access (CDMA) [**EVBH08**]. The 2G evolved in 2.5G in 1999, better known as Enhanced Data GSM Evolution (EDGE), which was closer to the next generation (i.e., 3G) in terms of Quality of Service (QoS) and throughput (up to 300 Kbps against 64.4 Kbps of the GSM) [**BMC04**]. Despite the

enhancements with respect to 1G, also 2G had many drawbacks, especially interference issues due to frequency reuse. Also TDMA shown its limitation in case of unfavorable terrains, topographic or electromagnetic conditions that can bring to communication failure [**EUO+18**].

The third generation of mobile phone (3G) starts with the introduction of the Universal Mobile Terrestrial/Telecommunication Systems (UMTS). This communication era provided dedicated networks to improve throughput and reduce latency. Moreover, many services started from this generation such as global roaming and enhanced voice quality. The data transfer capability increased up to 4 times faster than 2G, enabling multimedia services like video streaming and Voice over IP (VoIP). Users could access data from any location in the world as "data packets" since the network evolved from circuit commutation to packet commutation [**Fag14**]. This network switching technique enables the possibility of encapsulate data into a packet that can be routed from a source to a destination. In this scenario, no dedicated circuit is necessary to forward the traffic as considered in the circuit switching paradigm. The most significant drawback of this generation relies on the energy consumption since more power is necessary, with respect to 2G, to sustain the communication between user devices and cells [**HT05**]. In order to enhance data rate in existing 3G networks, another two technology improvements are introduced to network: High Speed Downlink/Uplink Packet access (HSDPA/HSUPA) and High Speed Packet Access plus (HSPA+). These technologies refers to the advanced 3G network (i.e., 3.5G) that can support up to 42 Mbps data rate.

The fourth generation of mobile phone (4G) is the present generation of communication. 4G architecture is design to improve the packet-switched traffic with seamless mobility, QoS and latency/throughput communication. The main components of the architecture are the evolved Node B (eNB) and the Evolved Packet Core (EPC). The first one is dedicated to acquire the user data and process them according to the 4G protocol stack. The EPC is responsible for the communication management and it is composed by four entities: Home Subscriber Server (HSS), Service Gateway (SGW), Packet Data Network Gateway (PDNGW) and Mobility Management Entity (MME). The HSS is a database that contains user-related and subscriber-related information. SWG is the connection point between the radio-side and the EPC while PDNGW enables the communication between EPC and the external IP network. MME handles the

signalling related to mobility and security for network access, interacting with the HSS [**R+13**, **Kha09**]. To achieve the minimum communication latency, advanced techniques are introduced like Multiple Input Multiple Output (MIMO) and OFDM. The main drawback of the system is the interference due to the crosstalk from two different transmitters using the same channel. This problem is a constant presence from 2G, where it represented the main issue for a good communication, to 4G and also 5G. However, many interference coordination schemes and interference-aware receivers have been developed from 2G to 4G to significantly reduce the problem. For instance, we can consider the Network Assisted Interference Cancellation and Suppression (NAICS) that considers advanced receivers capable of detecting interfering transmission, improving the QoS of the overall network [**HT09**, **ADF+13**].

Other wireless technologies are part of 4G era, such as WiFi, Bluetooth and Zigbee. These protocols are thought for both improving end-user communication and introducing Internet of Things (IoT) applications. This represents an important improvement in terms of flexibility and compatibility, considering that 2G does not even support the roaming. This aspect is also underlined by the backward compatibility of the 4G systems with the previous generation of mobile phone.

So far, each generation of mobile technology brought along with it improvements that have changed our daily lives. 1G enabled the first communication, 2G enhanced 1G by means of digitalization and improving sound quality, security and network capacity. 3G introduced the mobile broadband for internet services, while 4G addressed the growing demand of bandwidth dedicated to each user. The fifth generation of mobile phone (5G) should address the limitations of the 4G providing high bandwidth and low latency for a large number of users. Moreover, it should also address a wide range of services, especially in the field of machine-to-machine communication [**Yil16**]. Indeed, the effort that the scientific community is dating towards the deployment of 5G leads to think that the next generation will be disruptive. It will provide extremely low latency/high bandwidth connection for many applications, from HD streaming to autonomous driving. In other words, connections everywhere at anytime with higher capacity, higher data rate, lower end-to-end latency, massive device connectivity, reduced cost and consistent QoS provisioning compared with 4G [**Ben14**].

Generation	Speed	Technology	Time Period	Features
1G	14.4 Kbps	AMPS, NMT, TACS	1970-1980	Voice only, no roaming
2G	64.4 Kbps	TDMA,CDMA	1990-2000	Data + Voice, roaming
2.5G	170-300 Kbps	EDGE, GPRS	2001-2004	Multimedia services and streaming starts
3G	1-3 Mbps	UMTS, CDMA	2004-2005	Multimedia services and streaming support
3.5G	14-42 Mbps	HSDPA, HSUPA, HSPA+	2006-2010	Higher throughput and speeds
4G	100-300 Mbps	OFDM	2010-now	High definition streaming and multimedia
5G	Gigabits	Under study	2020	High speed and throughput for large number of devices

Table 1.1. Summary of the wireless network generations

2. 5G vision and challenges

The increasing number of connected devices brings to a big rise of the traffic volume. 5G has the role to address such enormous traffic, providing low latency/high throughput communication, network scalability and flexibility as well as low system complexity and power consumption [**OBB+14**]. To cope with such requirements, 5G envisions to improve carrier frequencies with larger bandwidth, number of antennas and the network infrastructure. The need to change frequency is due to the scarcity of the free frequencies remained in the bandwidth considered so far. Table 1.2 summarizes the multi-layer frequency approach proposed for 5G [**LTRa+18**].

Frequency Range	Layer	Note
High frequencies Above 6 GHz	Super Data Layer	Extremely high data rates
Medium frequencies 2-6 GHz	Coverage & Capacity Layer	Compromise between capacity and coverage
Low frequencies Below 2 GHz	Coverage Layer	Lower data rates to improve coverage area

Table 1.2. 5G frequencies overview

An increasing number of antennas permits to enhance the network densification which corresponds to more nodes served per unit area. The novelty in this field is to include small cells alongside distributed antenna systems. Pico-cells (i.e., hundreds of meters) and femto-cells (i.e., some meters like home WiFi range) are considered as small cells. Antenna distributed systems are similar to pico-cells from a coverage point of view but the processing is demanded to a central site that is connected to many antennas. In this scenario, the transmission power is lower, reducing the power consumption, with the advantage of reaching more connected devices due to densification [**PCK12**]. Power consumption saving is also achieved by the energy harvesting that can maximize the lifetime of the wireless devices. The idea behind this concept is to transmit data via radio signals that can also recharge the devices. Indeed, in RF energy harvesting, radio signals with frequency range from 3 KHz up to 300 GHz can carry energy in form of electromagnetic radiation [**LWN+14**]. However, the benefits of the increased number of cells pose challenges for the mobility. A moving node of the network has to pass and interact with many cells, thus the passage between cells is frequent and it can cause a lack of synchronization. Indeed, the handoff can be particularly challenging since beams (i.e., messages used for synchronization) have to be aligned to communicate. Actually, such handoff does not exist at layer 3 (i.e., IP layer) in the 5G communication, but many coordination techniques are under study where the user can communicate with many base stations to achieve the optimal synchronization [**ABC+14**].
Regarding the 5G architecture improvements, this thesis focuses at this aspect, thus an exhaustive description of the infrastructure will be provided in the first chapters of this work.

3. Contributions and outline

This thesis aims at presenting a novel approach to accelerate the 5G infrastructure at the edge. The idea is to exploit hardware acceleration to improve the processing of the protocol stack functionalities and network functions close to the final user. In this way, the bandwidth for the communication between 5G radio infrastructure and the central Cloud can be saved. Moreover, real time application can benefit from the improved computation capabilities by means of hardware offloading. In Chapter 2, the overall architecture of the 5G is introduced. When referring to 5G communication, its architecture is fundamental to

clearly describe the possible improvements that can be addressed by means of hardware offloading. In particular, this chapter describes both 5G Radio Access Network (RAN) architecture and the access network for the datacenter. Hardware acceleration can enhance both solutions that are considered as case studies in this work.

Chapter 3 focuses on the reconfigurable computing approach that can be exploited to accelerate the functionalities of 5G protocol stack. Reconfigurable computing is paradigm that considers programmable hardware acceleration of software algorithms. The idea is to enhance the processing with very flexible hardware platform like FPGA with the supervision of the software that has the role to offload the data and control the hardware processing. The approach proposed in this Chapter is based on Open Computing Language (OpenCL) for FPGA hardware platform. This represents the first study to evaluate OpenCL hardware acceleration in the context of a 5G base station.

Chapter 4 leverages on the improvements of the network functions at the datacenter access network. The potential of the Programming Protocol-independent Packet Processor (P4) open source language, recently introduced by the inventors of OpenFlow, is considered to program the behavior of a switch located at the edge of the network access infrastructure. Special effort is dedicated to motivate and apply P4 within a multi-layer edge scenario. Both Traffic Engineering (TE) and cyber security scenarios are considered as network functions to be improved by means of P4 language. The TE use case is tested with a P4 software switch while the cyber security scenario considers an hardware switch, based on FPGA and compatible with P4 programs. Moreover, FPGAs are also used to accelerate processing functions at the edge, exploiting a purely hardware pipeline. In this way, the computation is completely demanded to the hardware without offloading the data from software. An host application is responsible for monitoring the correct operations execution by changing the parameters of the hardware pipeline.

Finally, Chapter 5 concludes this thesis and provides open future perspectives.

CHAPTER 2

Overall 5G architecture description

The fifth generation of mobile communications (5G) brings with it unprecedented challenges that require thinking in new ways to meet the aggressive performance goals. Indeed, it will revolutionize our world with enormous data transfer capacities that will power the future of smart cities, robotics and the next billions connected things. In addition, the 5G infrastructure will provide tailored solutions such as automotive, agriculture, healtcare, etc [**Moh16**]. Thus, it is necessary to improve the networking service for all the stakeholders. Strong industry partners engage in all stages of development, starting with early research, contributing to standards development, developing technology and ultimately deploying networks.

To meet the requirements of the 5G, the network operators have to allocate resources for each kind of service deployed, which reflects to the implementation of orchestrator functions to manage the entire communication. Thus, each service will have a dedicated logical network called *network slice*. This feature represents one of the main novelty of 5G to address all the tasks and dedicate resource for each business. The ubiquitous approach considered for the 5G infrastructure have to operate in an energy efficient way to contain the power consumption of the network [**Kal18**]. In such environment, the following design objectives can be summarized to understand the challenges of this revolution in the wireless communication.

Spectrum availability: this feature is essential to support the huge amount of data demand. Thus, 5G network have to operate considering wider spectrum than the actual 4G technology. Indeed, the frequency range can be divided in three different parts: below 2 GHz for better coverage area, 2-6 GHz for higher throughput and super/extremely high frequency (over 6 GHz, up to 60 GHz) for military and aerospace applications [**WLH+14**].

Efficient data processing: the improvements of the last years about data processing permit to reduce the processing time of the 5G tasks to meet the latency requirement which represents a really strict constraint in the next generation of mobile phone. In particular, the use of different platforms such as General Purpose computing on Graphics Processing Unit (GPGPU) or FPGA, that exploit the massive parallel computing, can reduce significantly the computation time [**CLS+08**].

Addressing air interface variances: the number of wireless protocol is growing to support a large number of application, especially in the IoT environment. This means that interference can happen when enabling all the protocols. In this scenario, the physical layer has to be rethought with different waveforms and numerologies to meet the variances of the air interface. Moreover, an efficient inter-networking between 5G and 4G is essential to maintain the compatibility with the previous technology [**CSSK+15**].

Multi-connectivity: multiple access points can simultaneously configure radio resources to a given user device, introducing link diversity. This permits to improve the reliability and many schemes are under study to improve the number of different technologies connected simultaneously [**RRM+16**].

Convergent fixed-mobile networking: in the new era of communication, the same physical network has to be shared between wired and wireless user connections. Thus, virtual connection may operate in parallel to ensure connectivity among all the customers, exploiting the network slice paradigm. Moreover, a Software Defined Networking (SDN) orchestrator allows to have an ordinate management of the same physical infrastructure [**EGS+15**].

Traffic differentiation: complex algorithms for traffic differentiation will be implemented to meet the stringent constraint regarding the QoS. Separation and prioritization of the resources are the keywords to be taken into account when deploying this feature [**G+17**].

To address the above objectives, an intelligent transport network has to be deployed and enhanced automation capabilities in the operation and management represent a key requirement. Thus, Figure 2.1 shows the complete transport network, from the 5G RAN up to the datacenter access with the management and

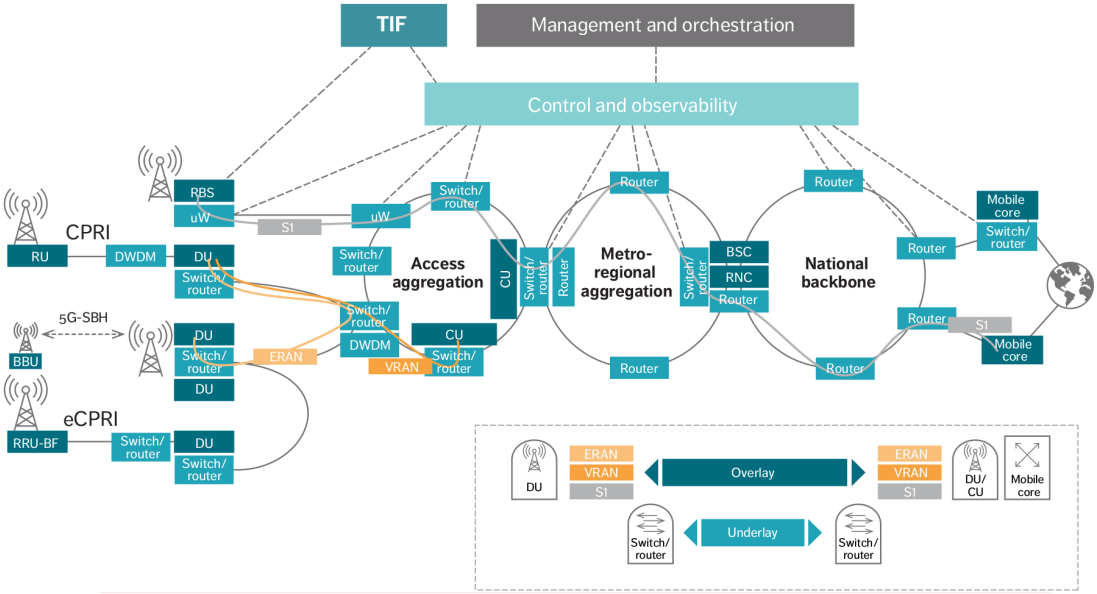


Figure 2.1. 5G transport network architecture

orchestration layer and the Transport Intelligence Functions (TIFs) [**DDGF+18**]. This chapter focuses on the state-of-the-art of such architecture and, in particular, the 5G RAN is described in Section 1 while the datacenter access network is deepened in Section 2.

1. 5G RAN Infrastructure

The 5G infrastructure must provide high level of efficiency, flexibility, and scalability in the RAN. Indeed, applications such as enhanced mobile broadband (eMBB), massive machine type communications (mMTC), and ultra reliable low latency communications (URLLC) [**GT**] demand requirements that the previous generation (4G) mobile communications cannot satisfy. Thus, the RAN has to be re-thought.

In 4G, the eNB represents the hardware and software located at the antenna site, enabling the wireless access of many mobile devices (e.g., smartphones, tablets, PCs, etc.) to the mobile network infrastructure. Here, the protocol stack and signal processing, which handles the data from mobile devices are resolved. This approach has many limitations, in particular for what concerns flexibility and scalability, since the eNB is structured to provide connectivity for a certain number of devices and it is not able to adjust the power consumption (e.g., putting

in standby some functionalities) in case of less users connected. On the contrary, if an area requires a higher number of users to be served, another eNB has to be deployed.

An important novelty of 5G is splitting the eNB into two different entities: the Central Unit (CU) deployed in central locations and the Distributed Unit (DU) deployed near the antenna. This architecture is called Next Generation RAN (NG-RAN) or C-RAN, where the "C" stands for both centralized and cloud. Indeed, CU and DU can be even virtualized and deployed in the cloud [**GKG+19**]. Likewise, the eNB in the 4G infrastructure evolves in next generation eNB (gNB). Centralizing the CU leads to several benefits including economy of scale, reduction of the maintenance for the cell towers, and performance improvement due to better coordination between antennas. Economy of scale refers to the possibility of deploying less hardware devices compared to the actual 4G infrastructure. For instance, a single large router for network access can serve the CU rather than many little routers serving a single eNB. In this way, the CU can be designed to be able to move router ports from under-utilized CUs to over-utilized CUs. The maintenance can be reduced by upgrading the software of centralized CUs. Finally, the performance in terms of lower call drop rates and downlink data rates are 30% improved with the centralized approach [**Per17**].

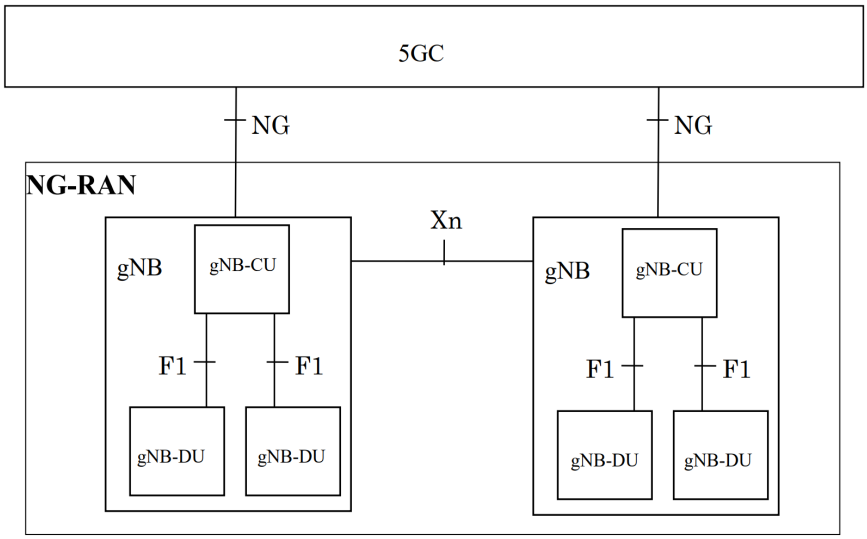


Figure 2.2. 5G architecture

The NG-RAN architecture is depicted in Figure 2.2. Here, many gNBs are connected to the 5G Core (5GC) which manages the User Equipment (UE) communication with the Data Network (DN). The link between the gNBs and 5GC is the NG interface, which corresponds to the *backhaul*. Each gNB consists of a CU, gNB-CU in Figure 2.2, connected to a DU, gNB-DU in Figure 2.2, by means of the F1 interfaces, which is the *midhaul* interface. This interface is the link between DU and CU. Often, based on the DU location, an additional element, including only the Radio Frequency (RF) functions (e.g., filtering, mixing, upconversion/downconversion, Digital to Analog Conversion) called RRU (Remote Radio Unit), is deployed and it is connected to the DU by the, so called, *fronthaul* interface. Many interfaces have been defined to transmit the data in the fronthaul and midhaul links, such as Common Public Radio Interface (CPRI), ethernet-based CPRI (eCPRI) and Next Generation Fronthaul Interface (NGFI). In particular, CPRI, eCPRI are protocols defined for the fronthaul data transport while NGFI is designed for the midhaul communication. CPRI is a radio interface developed by several leading telecom vendors to transport sampled RF data between the DU and the CU. In Figure 2.3, the option 1 CPRI encapsulation is presented as an example to understand the CPRI frame structure. Here, the In-phase/Quadrature (I/Q) samples are encapsulated inside a frame which lasts 260 ns (1/3.84MHz). Each I/Q sample is encoded with 8-bit word and 16 words are transmitted in a single frame (called chip). Thus, the transmission throughput (TT) can be calculated as follow:

$$TT = 1\ chip \times 16\ words \times 8\ bit \times 3,84\ MHz \times \frac{10}{8} = 614.4\ Mbps \qquad (1)$$

Where $\frac{10}{8}$ is the factor due to the 8B/10B line coding for the error correction/detection used in CPRI option 1 [**Spe**].

The utilization of native CPRI in networks, instead of point-to-point connections only, implies the development of dedicated hardware not compatible with current MAC and PHY standard protocols, potentially causing cost inefficiency. Thus, transporting CPRI frame over eCPRI has recently increased interest because of its flexible, cost-effective deployment, easy integration with the current high speed ethernet-based optical networks. Figure 2.4 shows how eCPRI message is mapped into transport network layer payload (e.g. UDP/IP or Ethernet) [**C$^+$17**]. In [**CYRN$^+$16**] the impact of different encapsulation techniques between the DU

and CU functional splits have been described for the use of Ethernet based fronthaul. Moreover, the bandwidth requirements for CPRI are high and the link rates are currently limited without the possibility to scale up. Bandwidth is not the only restrictive aspect, in fact CPRI also requires very strict latency constraints that do not permit to transport data for long distance [**dlOHLA16**].

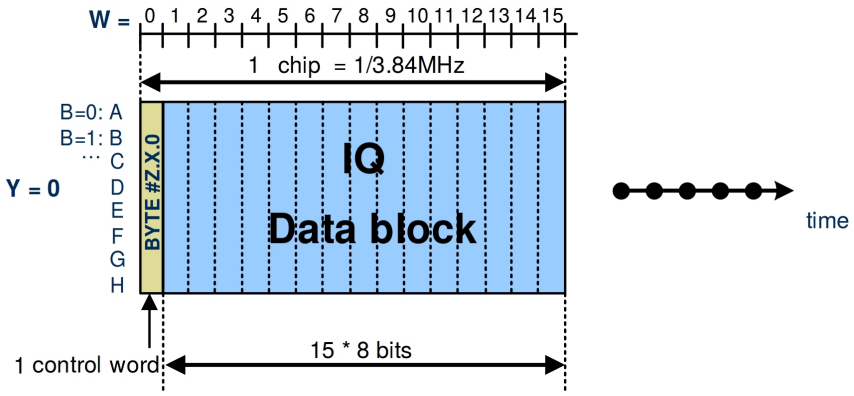


Figure 2.3. CPRI encapsulation

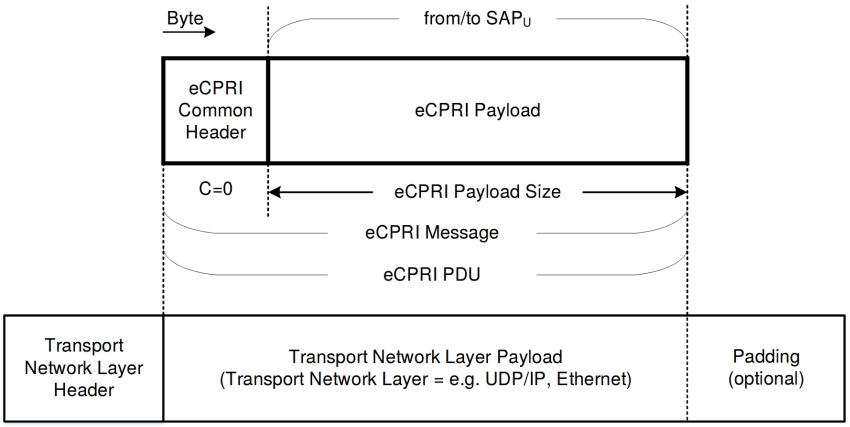


Figure 2.4. eCPRI message format

Regarding the midhaul, as the gNB is splitted into CU and DU, so are the protocol stack functionalities. This is an important novelty of 5G called *functional split* where the functionalities can be splitted according to the scheme in Figure 2.5. The 3GPP association established the requirements, in terms of both latency and bandwidth, reported in Table 2.1 [**3GP17**], for the communication network

Protocol Split Option	Required bandwidth		Max allowed one-way latency
Option 1	DL: 4Gb/s	UL: 3Gb/s	10ms
Option 2	DL: 4016Mb/s	UL:3024 Mb/s	1.5∼10ms
Option 3	lower than option 2 for UL/DL		1.5∼10ms
Option 4	DL:4000Mb/s	UL:3000Mb/s	approximate 100μs
Option 5	DL: 4000Mb/s	UL: 3000 Mb/s	hundreds of microseconds
Option 6	DL: 4133Mb/s	UL:5640 Mb/s	250μs
Option 7-1	DL:10.1∼22.2Gb/s	UL:16.6∼21.6Gb/s	250μs
Option 7-2	DL:37.8∼86.1Gb/s	UL:53.8∼86.1 Gb/s	250μs
Option 7-3	DL:10.1∼22.2Gb/s	UL:53.8∼86.1Gb/s	250μs
Option 8	DL:157.3Gb/s	UL: 157.3Gb/s	250μs

Table 2.1. Bandwidth and latency requirements for each functional split

connecting DU and CU as a function of the considered functional split. As reported in Table 2.1, the requirements are stricter when most of the functions is implemented in the CU.

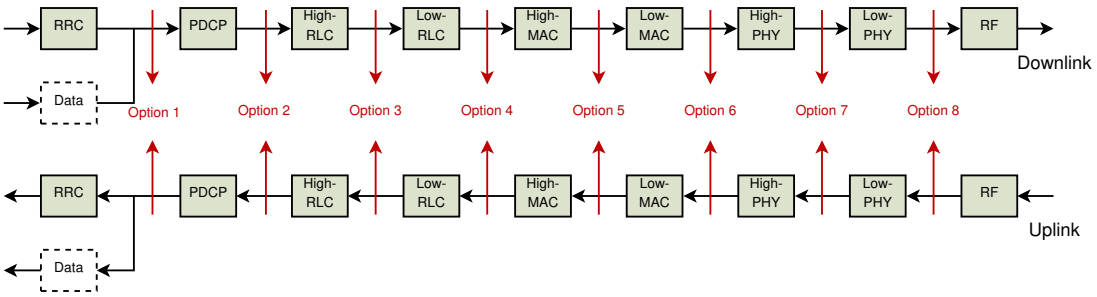


Figure 2.5. 5G functional split

Several other ongoing efforts are investigating on midhaul solution such as the Institute of Electrical and Electronics Engineers (IEEE) 1914 Working Group with the NGFI. In NGFI, the midhaul traffic depends on the functional split implemented and bandwidth and latency constraints may vary according to the functional split used [$\mathbf{Z^{+}15}$]. NGFI has identified both lower and higher possible functional splits. It is claiming to be the protocol that supports key technologies for 5G such as statistical multiplexing and radio interface technological neutrality. The NGFI is based on Radio over Ethernet (RoE) encapsulation, which is summarized in Figure 2.6 and Figure 2.7 where both hierarchy and RoE encapsulation format are presented. The RoE node maps the CPRI ports into Ethernet links exploiting the mapper/e-mapper process blocks. The packet that has to be

sent over Ethernet link is depicted in Figure 2.7. The RoE payload contains a flow of I/Q samples for a single antenna carrier of a group of antenna subcarriers [**19118**].

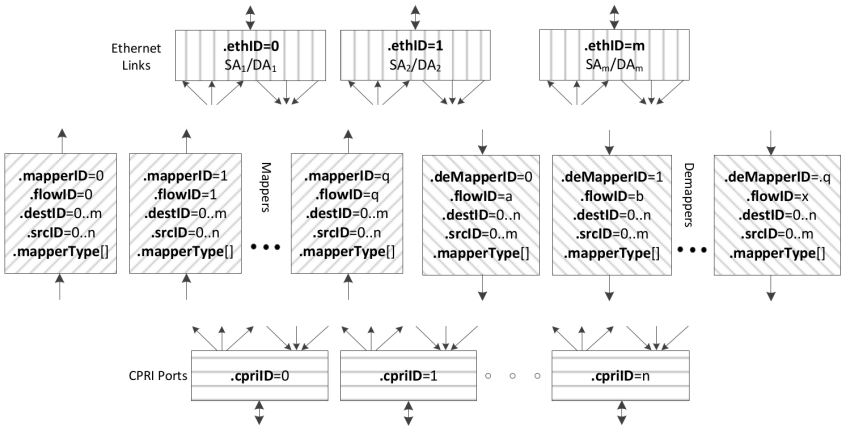


Figure 2.6. RoE node hierarchy

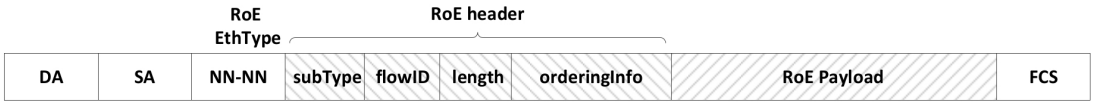
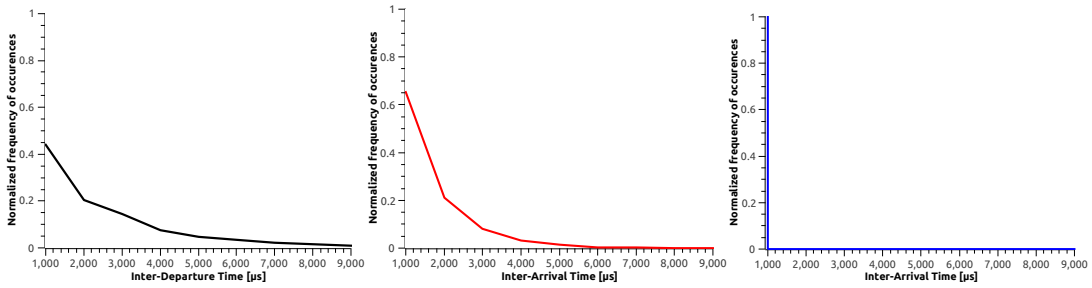


Figure 2.7. RoE message format

An experimental characterization of the midhaul traffic by considering different gNB functional splits (i.e., Option 8, Option 7-1 and Option 2) is detailed below. In particular, the impact of the user traffic, injected on midhaul with different inter-departure time distributions, is evaluated. The utilized mobile network software is OpenAir Interface (OAI) which uses NGFI technique for the midhaul communication [**NMM+14**].

The experiment is coped with sending data between the gNB and the UE. Here, the sender component of D-ITG tool (ITGSend) were running at EPC to generate downstream UDP traffic, and the receiver component of D-ITG tool (ITGRecv) were running at UE to receive UDP traffic [**AGE+04**]. The Wireshark capture was initiated at DU to capture the midhaul traffic for Option 7 functional split, where the Wireshark capture was initiated at CU to capture the midhaul traffic for Option 2 functional split. Figure 2.8 shows the user traffic generated by following negative exponential distribution from gNB to UE. In particular, the



(a) (b) (c)

Figure 2.8. Downstream user traffic generated with exponential Inter-Departure Time between packets: a) source of the traffic generated at EPC b) the fronthaul traffic captured at the CU with Option 2 functional split c) the fronthaul traffic captured at the DU with Option 7 functional split

behaviour of the inter-departure time of the traffic generated at the gNB side is depicted in Figure 2.8a. Then, the traffic is captured with help of the Wireshark at CU with Option 2 functional split and the inter-arrival time of the traffic follows the negative exponential distribution, as shown in Figure 2.8b. However, in Figure 2.8c, it is shown the inter-arrival time of the traffic, related to the lower layer functional split (Option 7-1), is independent on the user traffic. In fact, the distribution no longer follows the negative exponential trend. The experiments for uplink communication and with Option 8 were performed, but not presented because they follow the same behaviour of downlink communication and Option 7-1 respectively. Thus, the midhaul traffic, based on NGFI protocol, is independent on the user traffic profile when the lower functional splits (i.e., Option 8 and Option 7-1) are considered. Whereas, the midhaul traffic is dependent on the user traffic when the higher functional split (i.e., Option 2) is considered.

2. 5G Datacenter access network

Besides the RAN, the datacenter is the other core of 5G transformation. At the beginning of this Chapter we listed the main objectives of the architecture based on the efficiency, scalability, flexibility and intelligence. Thus, a question rises: what kind of datacenter is able to meet the requirements of the 5G? To answer the question, we list the main features to build a datacenter in the 5G era [**Gu18**].

Open: the layers of the datacenter should be open to be accessible from the research community for continuous enhancements. Indeed, the present datacenter infrastructure is based on frameworks like Openstack and database queries are inspired by standards like Hadoop, Spark, MySQL and Redis. These technologies are still under development to offer the best performances in the 5G environment.

Efficient: the latency and bandwidth requirements are 100 times stricter than the last generation of mobile phone. Moreover, with the support of network slicing, many services can be deployed in parallel and most of them are critical in terms of latency and reliability like ultra-HD video streaming or intelligent transport system. For these reasons, hybrid solutions are investigated like System on Chips (SoCs) based on ARM/Intel cores and FPGA, GPGPU and Neural network Processing Unit (NnPU). Reconfigurable computing could also improve the software performances and bottlenecks, exploiting the hardware offloading of the tasks controlled by the software. This solution is also suitable to accelerate the processing of the huge amount of data coming from the IoT environment. Indeed, GPGPU/FPGA clusters can accelerate the computation by means of massive parallel computing.

Flexible: the orchestration of the network slices has to adapt to the changes of the 5G network. Indeed, all the services such as including capacity, service configuration, network elements and service applications should be deployed as quick as possible. Virtual machines to handle the communication can be easily added or removed from the network, thus the flexibility is an important feature to follow the changes.

Distributed: in the 5G environment, most of the services are cloud-based (e.g., IoT core services, big data, deep learning, etc.) but many other functionalities can run in an edge datacenter, reducing the computation at the central location. Thus, ubiquitous computing (i.e., the capacity of the computing to appear everywhere when needed [**Wei91**]) can improve the processing by moving the functionalities at the edge. Such a computing should be controlled by a central intelligence and deployed to support many services like object detection and recognition in surveillance videos, 3D rendering, real-time operations, gaming and much more.

Intelligent: this feature represents one of the main keys for the 5G infrastructure. Indeed, the 5G network manager has to be flexible and to adapt to the network changes. To achieve this goal, a powerful Artificial Intelligence (AI) architecture should be considered along with heterogeneous computing hardware, including GPGPUs, FPGAs and NnPUs, capable of implementing high computational capacity algorithms.

To meet such requirements, a new idea of datacenter is rising based on the Network Function Virtualization (NFV) functionalities that permit to reach a higher levels of flexibility with a rapid deployment. Indeed, many services starts to be deployed with the NFV approach, such as fixed and mobile core network or the 5G RAN as well as the traffic analysis, optimization and security functions [**HPS+15**, **OLAL+17**, **AKL+18**]. Figure 2.1 shows the datacenter access network for the 5G environment. The access aggregation entity corresponds to the edge computing node of the datacenter that is physically close to the users. Its location represents a novelty in the architecture and permits to accelerate the data processing in order to only send metadata to the cloud central node [**Yu16**]. The metro network domain provides connectivity between the edge node and the national backbone, as well as to the global internet from the central cloud node. At the metro network level, the data coming from many edge nodes are collected exploiting the optic fibers based on the Dense Wavelength-Division Multiplexing (DWDM). Such a technology can aggregate up to 100 wavelength channels over the same fiber to accommodate the large aggregated traffic from the access branches. Moreover DWDM permits to reach an incredible high data throughput (around Tb/s) in a single optical fiber [**DGE+15**]. This aspect enables the possibility to easily aggregate all the traffic coming from large variety of services, from entertainment to IoT, 5G and much more.
Finally, the central cloud node located far from the regional and edge nodes, it is the collector of many services like deep processing, networking and other computing resources.
Focusing on the cloud edge node, the NFV approach is more suitable to be deployed at this level of the datacenter architecture since the edge node has to adapt to the services changes based on the users requests. Thus, the flexibility of the NFVs may help to follow the network modifications. Moreover, the deployment of the NFV functionalities in such instance allows to improve resilience, latency and jitter, as well as to reduce the computational load in the metro and

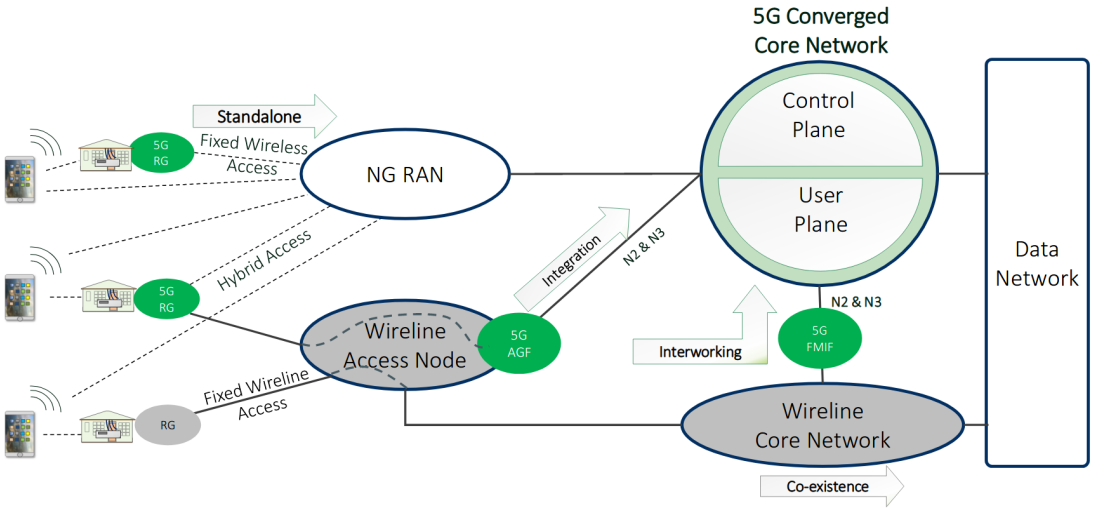


Figure 2.9. Proposed fixed mobile convergence architecture

central nodes. Many researches are still conducted to standardize this new approach for the datacenter edge node, called Central Office Re-architected as a Datacenter (CORD) [**PASA+16**, **ASP15**], where Next Generation Central Office (NGCO) will host the edge node for new mobile network [**BMW18**]. Such technological upgrade coincides with the recent replacement of the copper links with optical fiber, especially in the Last Mile (i.e., the connection between the edge node and the users). As discussed above for the metro nodes, the fiber can be also aggregated into the NGCO that is capable of serving few thousands of connections. Moreover, the NGCO can collect the data from both wired and wireless communication, implementing the 5G and fixed mobile convergence [**BFS+17**]. To achieve the convergence, common policy and user data management between fixed and mobile network had to be enabled. Moreover, NFV approach can improve the convergence by means of resource sharing and avoiding the network duplication due to fixed and mobile networks. The deployment of the convergence at the edge node enables the possibility to achieve the fully seamless datacenter architecture since the data are no longer separately treated based on the communication medium, but they are collected and sent to the cloud central node. Figure 2.9 shows the possible architecture for the convergence [**GBNM+18**].
Many other features can be deployed in the NGCO like services prioritization or dynamic service provisioning but a deepening is necessary for data processing and cyber security. Regarding data processing, many NFVs are dedicated

to data processing at the edge, thus they can be defined as Processing Function Virtualizations (PFVs). The adoption of the programmable hardware offloading may speed up advanced applications and processing functions. For cyber security scenario, one of the main functionalities is the mitigation of the Distributed Denial of Service (DDoS) attacks that can generate excessive bandwidth and it can make the network unstable. The mitigation of these attacks at the edge node has an enormous benefit since the attack is immediately resolved at the edge of the network, without spreading up to the central node. NFV can be deployed to address problems of security mitigation to neutralize possible strikers. As discussed for data processing, such a mitigation could be done with the help of other platforms like Application Specific Integrated Circuits (ASICs), FPGAs or Network Processing Units (NPUs). Indeed, co-design approach can improve data processing and mitigation problems since it combines the benefit of both hardware and software approaches. NFVs are dynamically deployed and they abstract the hardware resources while hardware platforms improves the processing performance in terms of processing time and throughput. Typically, ASIC utilization is expensive to design and it is not reconfigurable, thus its use is targeted for lower lever of the ISO/OSI protocol stack, where the protocols and standards are well defined. However, for application layers that are still under development, the software approach is essential to quickly dispatch improvements. For these reasons, new languages start to have success since they permit the hardware/software communication, enabling the co-design approach. In this scenario, P4 language is capable of closing the gap since it can be compiled for several hardware platform like FPGA or NPU, eliminating the portability issue of the hardware. Moreover, it allows the hardware processing with the support of the software communication. In this scenario, P4 implementation is tailored on security application since it allows to process network data at high throughput with the addition of the software control to enable, disable and update mitigation algorithms when needed. The venue of this co-design languages permits to overcome the threshold of hardware/software application so far. In general, hardware approach like ASIC on NPU is considered for central cloud node (i.e., core) deployment to process huge amount of data dispatched towards the central node. Figure 2.10 shows boundary of the NFV and hardware usage in the datacenter network access [**BMW18**]. However, NFVs are considered to dynamically

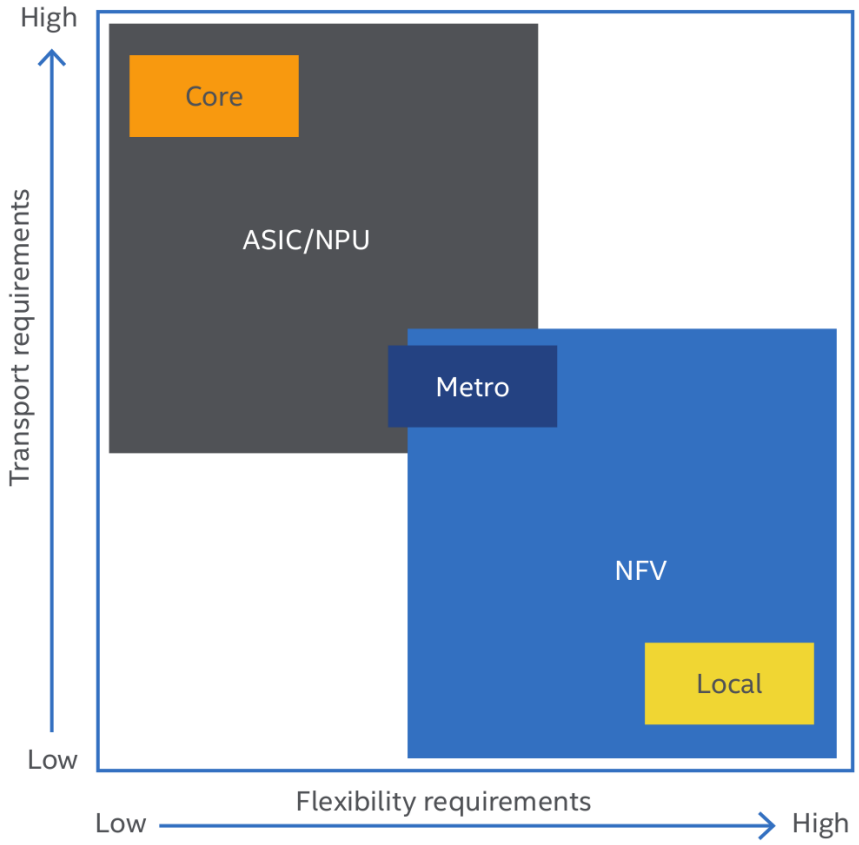


Figure 2.10. Silicon selection based on transport and flexibility requirements

adapt the services close to the end user with unpretentious throughput or processing time. Co-design approach should enable the possibility of using hardware architecture at the edge, improving the processing of the NFV and maintaining software flexibility. In this way, also security issues could be locally addressed, without loading the central cloud node.

CHAPTER 3

Reconfigurable computing for 5G RAN acceleration

In this Chapter, the reconfigurable computing approach is considered to accelerate part of the 5G network functionalities. In particular, it focuses on the acceleration of the 5G OFDM task in the DU by means of OpenCL framework. The OpenCL performance are then compered with the software SIMD and hardware HDL approaches.

1. Acceleration with Reconfigurable Hardware through OpenCL

1.1. OpenCL generalities. OpenCL is a framework for writing programs that can be executed across heterogeneous platforms such as Central Processing Units (CPUs), GPGPUs, Digital Signal Processors (DSPs) and FPGAs. OpenCL specifies a programming language (based on C99) for programming these devices and Application Programming Interface (API) to control the platform and execute programs from the host side (i.e., computational unit on which the host program runs to control the OpenCL device). Originally thought as the intersection between GPGPU and CPU parallel programming, OpenCL is now the best candidate as heterogeneous computing language, especially after the implementation of the FPGA support. It supports different types of parallelism to be suitable for all the hardware platforms. On the other hand, each platform reaches the best performance exploiting a certain type of parallelism. For this reason, the main advantage of OpenCL framework is the possibility to write a single program that can run on heterogeneous platforms seamlessly, even if many optimizations have to be done for each platform to reach the best performance [**SGS10**]. Moreover, it exploits the parallel computing approach which enhances the application performance. Two main parallel programming models are possible with OpenCL: task parallelism and data parallelism. In the first one, the application can be decomposed in several tasks with different computation loads. Each task can be mapped into Processing Elements (PEs), that can run concurrently. This approach is very useful when there is no data dependency between tasks. However,

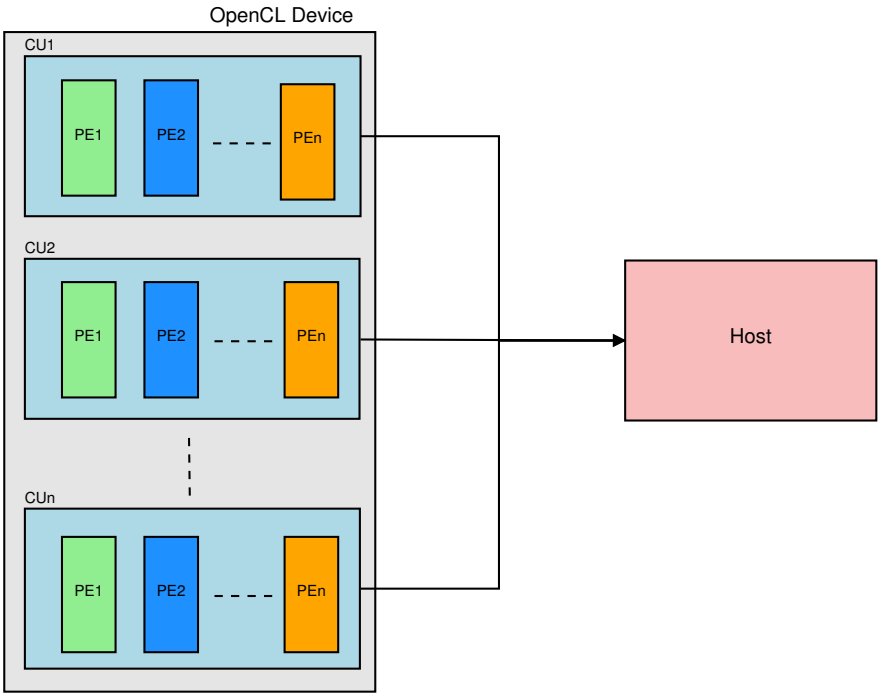


Figure 3.1. OpenCL platform model

it suffers of load balancing drawback since the processing finishes only when the last PE finishes its task. Thus, an unbalanced loading brings to a loss of performance. In data parallelism model, the application can be seen as collection of data elements that can be processed concurrently. This means that a single task can be applied to each data element. This model is possible when a processing on arrays or matrix is needed since each element of the structure can be easily computed by a single task [**MGMG11**].

The OpenCL platform model is shown if Figure 3.1. Here, a single *host* acts as a master capable to interact with many OpenCL *devices* which are the components where a stream of instructions execute. Such a stream of instructions is called *kernel* and it represents the program executed by the OpenCL device. Each device is called Compute Devices (CDs) and it can be a CPU, GPGPU, DSP or FPGA. Each CD can be divided in OpenCL Compute Units (OpenCL CUs) which are further divided into PEs. The PEs represent the smallest OpenCL computation units and all the processing run within the PEs [**GO11**].
The host program and one or more kernels have to be considered to create an

OpenCL application. The host program has the role to initialize and interact with the kernels by using the standard OpenCL API [**HM15**]. It establishes the context for the kernels as well as the command queues to be sent to the device. Moreover, it defines the memory objects which are the buffer to be read/written by the kernels. Each kernel needs its command queue. The host places the commands into the command queue and the commands are then scheduled on the associated device. The commands supported by the OpenCL are: i) kernel execution commands that are necessary to execute a task into a PE of the CD, ii) memory commands that transfer memory objects from/to the device and iii) synchronization commands that refer to constraints on the order of commands execution.
The kernels execute in the CD and they are the processing core of the OpenCL model. Moreover, two types of kernel are defined: pure *OpenCL* and *native* kernels. The first ones are functions written and compiled by the OpenCL framework. The native kernels are functions created outside the OpenCL environment and it is possible to call them by means of function pointers.
Particular attention must be dedicated to the OpenCL memory model since it enables the communication between the host and the device. Figure 3.2 shows the memory regions from the host to the devices. Here, five memory regions are defined [**TS12**]:

- **Host Memory**. This memory region belongs to the host and it cannot be accessible by any other instance of the OpenCL architecture.
- **Global Memory**. This region enables the communication from the host to the device and vice versa. Indeed, it can be accessible by all PEs. In Figure 3.2, a further cache level is depicted. This level may be present depending on the capabilities of the device.
- **Constant memory**. This memory has a read-only access for the PEs. During initialization, the host writes this region which remains the same during the kernel execution.
- **Local Memory**. This region is shared by all the PEs belonging to a certain CU and it is used to allocate variables that are commons for all the PEs. Moreover, if the CD has its own memory, it is deployed as a portion of the CD memory, otherwise it is implemented in the global memory. In this case, the access could be slower, depending on the global memory interface bandwidth.

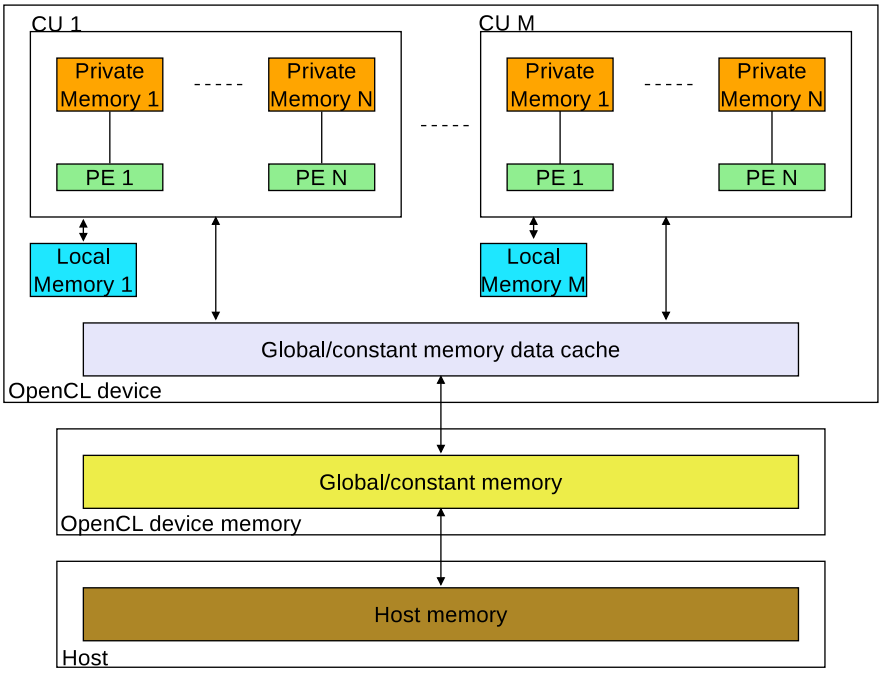


Figure 3.2. OpenCL memory model

- **Private Memory**. This memory region belongs to the single PE. The variables stored are not visible by any other instance of the OpenCL architecture.

1.2. Specific OpenCL constructs and kernels for reconfigurable hardware. In this Section, a focus on the kernels and tools considered to exploit the OpenCL model for FPGA is provided. In particular, it is presented the relation between the two types of kernels that can be deployed in hardware, as well as the SDK (Software Development Kit) used for this work, which is capable to create a hardware application starting from these kernels.
The OpenCL framework for FPGAs allows the designer to simply create an application by means of High-Level Design (HLD) approach. For this work, the Intel FPGA SDK for OpenCL is considered and it supports the OpenCL specification 2.0 [**HM15**]. The Intel FPGA SDK for OpenCL Offline Compiler compiles the kernels to an image file used by the host to program and run the kernel on the FPGA. The model previously described remains the same with an important difference: the FPGA exploits spatial implementation of a program, thus the

instructions are executed when the data are ready. However, in sequential programming (considered for CPU, GPGPU, DSP), the program counter controls the instructions to run. Even if the data are available to be computed, the processing is executed one instruction at a time, following the order of the program counter. This method is time-dependent since the instructions are executed on the hardware across time. Designers have to take into account this crucial difference to create an application FPGA-oriented. Indeed, the way the code is written permits to avoid FPGA area overhead or performance degradation.

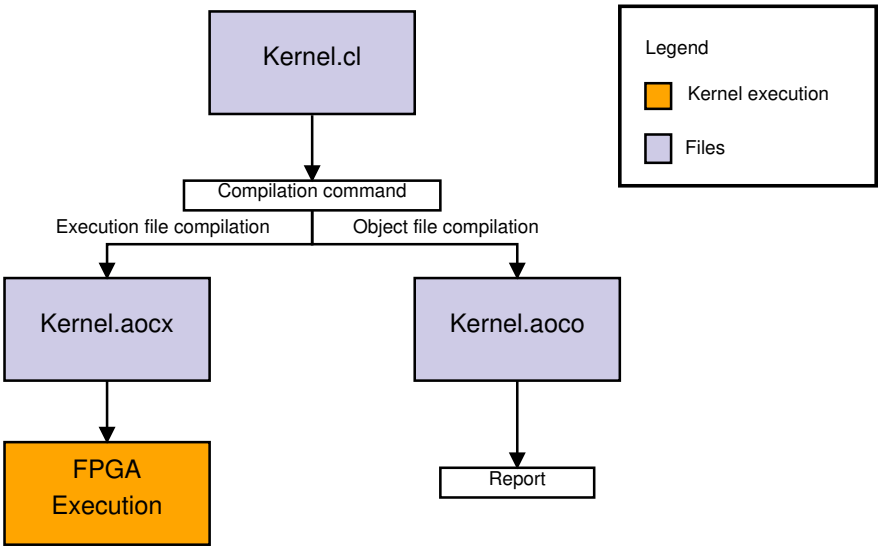


Figure 3.3. Intel FPGA SDK for OpenCL workflow

Figure 3.3 shows the Intel FPGA SDK for OpenCL kernel compilation flow. The Intel FPGA SDK for OpenCL Offline Compiler compiles the kernel and it creates the hardware configuration in a single step (i.e., kernel.aocx). An intermediate step, that creates an OpenCL object file (i.e., kernel.aoco), provides the report on area usage and performance bottlenecks, especially in the memory access.
Two types of kernels are supported by the OpenCL for FPGA framework: NDRange and Single Work-Item (SWI) kernels. A NDRange kernel permits to partition the data among the PEs. In particular, the OpenCL runtime system creates an integer index space (i.e., NDRange) where each instance of the kernel is executed. Each instance is called *work-item* and it is identified by its coordinates in the NDRange. This kernel is suitable when there is no data dependency that makes the partitioning easy. The commands run from the host to create the

collection of the work-items and execute all the work-items in parallel. Work-items are organized in structures called *work-groups* and each work-group has an unique ID, thus the single work-item can be identified through its global ID or by a combination of work-group ID and local ID (assigned to the work-item within the work-group). To understand how the IDs are mapped, we consider a 2D NDRange. A lowercase g letter refers to the global ID of a work-item, thus the coordinates (g_x, g_y) uniquely identify a work-item in the space. Instead, the uppercase G letter is related to the size of the space in each directions. The NDRange space can be written as [0 .. (G_x - 1), 0 .. (G_y - 1)]. Then, we divide the NDRange in work-groups with the same conventions described: W_x-by-W_y represents the NDRange space expressed in the work-group space while the coordinates (w_x, w_y) refer to the work-item inside the work-group. The dimensions L_x and L_y of each work-group can be calculated with the equations Eq.2 and Eq.3:

$$L_x = \frac{G_x}{W_x} \tag{2}$$

$$L_y = \frac{G_y}{W_y} \tag{3}$$

We explained that a work-item is identified by its global ID (g_x, g_y) or by the combination of its local ID (l_x, l_y) and work-group ID (w_x, w_y). Hence, Eq.4 and Eq.5 show the relationship between IDs.

$$g_x = w_x * L_x + l_x \tag{4}$$

$$g_y = w_y * L_y + l_y \tag{5}$$

Figure 3.4 reports an example where global IDs, local IDs, and work-group indices are depicted in a 2D NDRange space [**MGMG11**]. The memory model for the NDRange kernel remains the same depicted in Figure 3.2.

Executing NDRange kernels with size (1,1,1) (or with one work-group that contains one work-item) corresponds to run a SWI kernel. SWI kernels permit to control the pipeline since only one work-item is executed. This approach is better when an application has loops or data dependencies. The NDRange tries to parallelize the computation with several work-items but the performance could be degraded due to a possible stalls for conflicts with data. Unlike NDRange kernels, single work-item kernels follow sequential programming approach, more similar to C programming. Anyway, the data parallelism is applied on SWI kernel type by pipelining the iterations of loops. Indeed, custom pipeline is possible

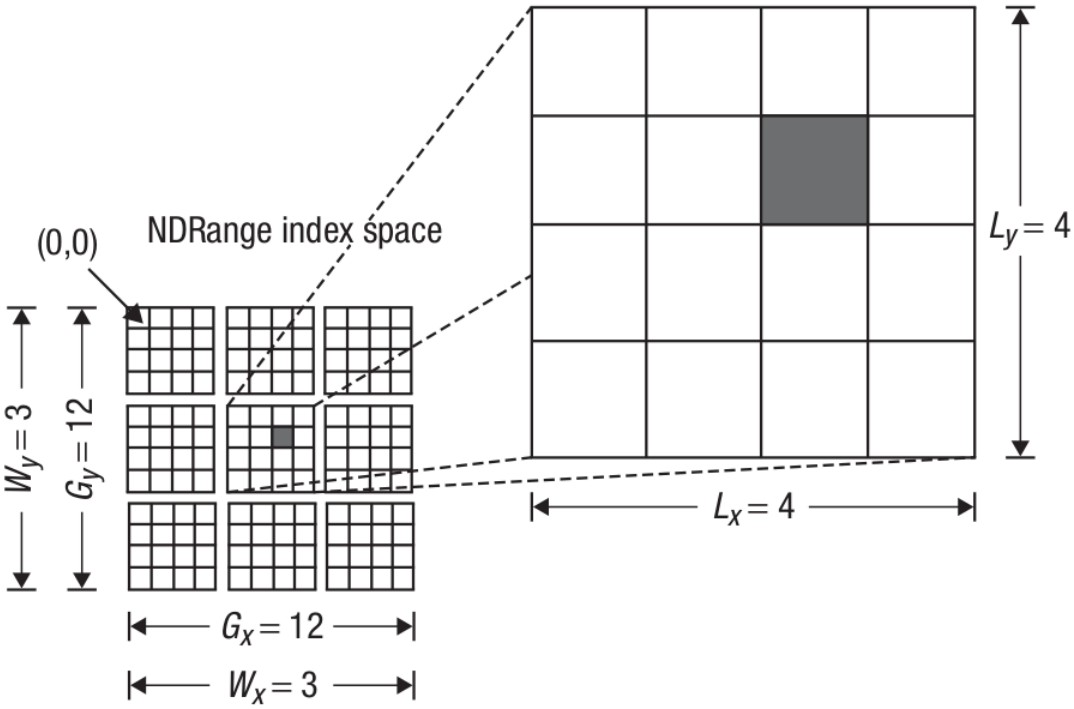


Figure 3.4. 2D example of mapping global IDs, local IDs, and work-group indices in NDRange

and data access patterns can be modified using ad-hoc "pragmas". More details about the improvements of these kernels in terms of area usage and processing time are described in the next section.

2. Insights on OpenCL kernel optimization for FPGA

The main advantage of OpenCL model is certainly the portability, since the same code can be compiled for different architecture. Moreover, its easy integration with the software enables the possibility of the parallel and sequential processing integration. Unfortunately, each architecture requires its optimization since specific instructions have to be executed to achieve better performance. For instance, the SWI kernel can run in the GPGPU but with very low throughput [**WS15**]. Indeed, the GPGPU is equipped with thousands of cores, each capable to run a single work-item, thus a NDRange kernel is more suitable for this platform. Hence, the platform optimization is an important process before compiling an

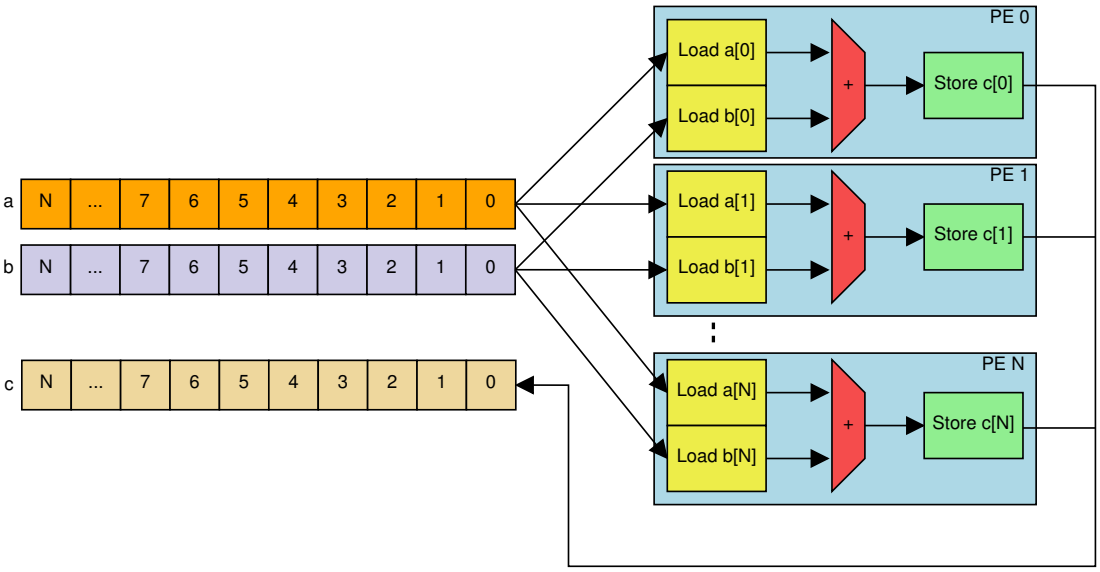


Figure 3.5. Data parallelization of NDRange kernels

OpenCL application and bad optimizations may lead to a platform inefficiency for OpenCL kernels. This section addresses the optimization to be taken into account for OpenCL FPGA SWI and NDRange kernels. A more exhaustive deepening is considered for SWI kernels since the application used as use case is SWI-based.

2.1. OpenCL NDRange kernel optimization. To understand how NDRange kernels are mapped into the FPGAs, we consider the code in Figure 3.6. This kernel permits to do the sum of each element of a with each element of b. OpenCL runtime spreads the sum operations among all the PEs, thus each PE has the role to add one element of a with one element of b, as depicted in Figure 3.5. To avoid area overhead, it is necessary to specify the maximum dimension of the work-group. In this way, the compiler is able to precisely map the kernel to the hardware resources.

Kernel vectorization permits to improve the throughput since multiple work-items can be executed in a single PEs, exploiting the SIMD instructions. N-sized vectorization can increase the throughput up to N times (where N can be 2, 4, 8 or 16). N sum operations can be mapped to a single PE without modifying the kernel structure. Here, each PE can handle N operations and the NDRange dimension can be decreased to save area usage.

Higher throughput can be achieved by means of running multiple compute units.

This method spreads the kernel processing among the compute units and the compiler handles each compute unit as a unique pipeline. Considering T as processing time to complete the kernel execution without multiple compute units and N as the number of compute units, the processing time with multiple compute units is $\frac{T}{N}$. This method increases the area usage as well as the global memory bandwidth which could be higher than the physical interface throughput of the global memory itself (e.g., DDR interface if the global memory is a Random Access Memory (RAM)). In this scenario, the performance could be worse due to a delay of global memory accesses.
Both kernel vectorization and multiple compute units can be instantiated to achieve higher efficiency, but kernel vectorization is preferable over the multiple compute units. Indeed, multiple compute units increase the number of accesses to the global memory while the SIMD vectorization changes the amount of work that a single PE has to handle. Exploiting the SIMD vectorization, no undesired memory accesses are possible and the performance are the same offered by the instantiation of multiple compute units.

```
__kernel void sum (
    __global const float * restrict a,
    __global const float * restrict b,
    __global float * restrict answer
)
{
    size_t gid = get_global_id(0);
    answer[gid] = a[gid] + b[gid];
}
```

Figure 3.6. NDRange kernel to add two vectors

2.2. OpenCL SWI kernel optimization. An OpenCL application based on SWI kernel would be preferable to NDRange kernels in FPGA due to possible conflicts in the memory accesses. Indeed, as described above, NDRange kernels are spreaded among all the PEs that have concurrent accesses to the memory, thus a data dependency can create stalls that decrease the performance. However, SWI kernels permit to control the whole pipeline, as well as the memory accesses. Figure 3.8 shows the same OpenCL code of Figure 3.6 adjusted to be a SWI kernel

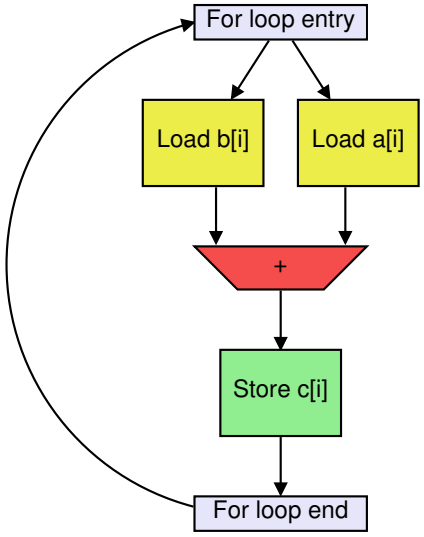


Figure 3.7. SWI kernel pipeline

and, in Figure 3.7, the pipeline of the SWI kernel is depicted. The different approach compared to the NDRange kernel is clear: SWI kernel are based on pipelines that can be controlled and customized while NDRange model is based on massive data parallelization. Both NDRange and SWI kernel include the *restrict* keyword in the kernel pointer arguments. Including this keyword, the compiler is prevented to create unnecessary memory dependencies and it avoids conflicts on memory accesses.

SWI kernel are loop-based, thus it is possible to achieve the best performance by avoiding data dependency on the pipeline and concurrent memory accesses. These conditions are fundamental to optimize a SWI kernel, otherwise the OpenCL compiler needs too much FPGA area to efficiently address data dependency (i.e., implementation of multiple instances of the same variable). A loop inside a SWI kernel can be unrolled, which means that the system tries to run each loop iteration in parallel. Hence, by means of `#pragma unroll N` directive, the compiler unrolls the loop N times. This reflects on a speed up of N times in the execution performance of the loop (without specify the N factor, the compiler automatically unrolls the whole loop). Unrolling read and write loops of global memory access enables to achieve the compile-time *memory coalescing*. This approach is typical when a uniform and regular memory access is needed. In particular, an irregular or not aligned access to the memory tends to decrease the performance in FPGA. The memory transfer between the host and the global memory has

to be aligned to exploit the Direct Memory Access (DMA) acceleration. Unrolling global memory loops permits to have an efficient access, exploiting the whole communication bandwidth of the memory. On the other hand, choosing a big values of N corresponds to have accesses larger than the external memory bandwidth, thus the best unrolled factor is the one that saturates the memory bandwidth.

A significant optimization on area usage is to avoid function calls in the SWI kernel. Indeed, every function call is implemented as a circuit on the FPGA, resulting in a large usage of FPGA resource. Moreover, functions calls prevent the compiler to create a correct report about FPGA area usage that makes the debug harder.

Writing OpenCL code for hardware-oriented programming leads to an improvement in terms of performance and area usage. As matter of example, we can consider a simple if-then-else directive implemented in both software and hardware. In software sequential programming, the if-then-else statement presents an issue in the pipeline. Indeed, there are two possible instructions to be fetched and the CPU has to take a decision based on the condition. Modern CPUs attempt to predict the correct instruction to be executed. Then, the processor fetches the instructions based on such a prediction and, in case of wrong prediction, it discards the partially executed instructions in the pipeline. In this scenario, the pipeline is not stalled, ensuring a smooth execution of the directives. However, in hardware approach (i.e., spatial programming), both "if" and "else" instructions branches are mapped into hardware circuits which lead to a FPGA area overhead since only one circuit is considered at a time. Using conditional statements (i.e., `out = (condition) ? in_1 : in_2`) instead of if-then-else statements can reduce the FPGA resources in many cases. Indeed, the if-then-else construct follows the sequential approach and infers a priority routing network, which needs higher resources. However, conditional statements are mapped as a unique MUX circuit in hardware and they yields better results in terms of both resource saving and performance [**ÖRHT16**].

```
__kernel void sum (
    __global const float * restrict a,
    __global const float * restrict b,
    __global float * restrict answer,
    int N
)
{
    for (int i = 0; i < N; i++)
        answer[i] = a[i] + b[i];
}
```

Figure 3.8. SWI kernel to add two vectors

3. Related works on OpenCL hardware acceleration

To the best of our knowledge, this study is the first that considers OpenCL acceleration in the context of 5G. A number of recent works rely on the OpenCL language for FPGAs to deploy advanced services such as Convolutional Neural Networks (CNNs), Finite Impulsive Response (FIR) filters or image processing accelerations. In [**WTS18**], the authors provide an exhaustive analysis about the Time-Domain Finite Impulsive Responses (TDFIRss) and Frequency-Domain Finite Impulsive Responses (FDFIRss) implemented in the FPGA by means of OpenCL. In particular, they focus on the performance of several FIR implementations on the FPGA as well as performance comparison between FPGA and GPGPU. Moreover, an analysis on the area usage and power consumption is reported.

An optimized implementation of a CNN is proposed in [**AOC+17**], where the authors consider Deep Learning Accelerator (DLA) to overcome the memory bound limit of the CNN deployment into FPGA. They maximize the data reuse and minimize external memory bandwidth. Moreover, Winograd transform is further applied to boost the FPGA performance. The results of such a scenario show that the system is 10x faster of the state-of-the-art implementation and it has comparable performance with GPGPU CNN deployment.

Another approach to improve CNN computation on the FPGA is described in [**ZL17**]. Here, an analytical performance model regarding area usage is presented and applied to a CNN implementation. Authors show a bottleneck on

the on-chip memory bandwidth and provide a 2D interconnection between PEs and local memory to reduce the on-chip memory requirement. Moreover, a 2D dispatcher is further designed to reduce the external memory bandwidth.
Optimization techniques for image processing are presented in [**DSF+17**] where the authors report an optimization review before presenting a spatial-spectral classifier for Hiperspectral Image (HI). This use case is based on the K Nearest Neighbours (KNN) filter deployed into the FPGA by means of OpenCL model. Both SWI and NDRange kernels are tested in terms of area usage and processing time. Moreover, a comparison between the output of KNN filtering and Support Vector Machine (SVM) classification is reported to demonstrate the image processing improvement.
Lastly, the implementation of an efficient 3D Fast Fourier Transform (FFT) is presented in [**SH18**]. The core of the work is to improve the pipelines of the FFT providing a valid alternative to HDL deployment. Moreover, the authors report a performance evaluation between OpenCL, HDL, GPU and CPU. The results show that the OpenCL performance are consistent with the HDL ones with the advantage of using fewer resources than IP core design. A significant improvement, in terms of execution time, is highlighted when the processing is performed via OpenCL and HDL with respect to CPU and GPU platforms performance.

4. Implementation

This section describes the implementation of the 5G DU processing into the hardware by means of OpenCL and the considered experimental setup. This work

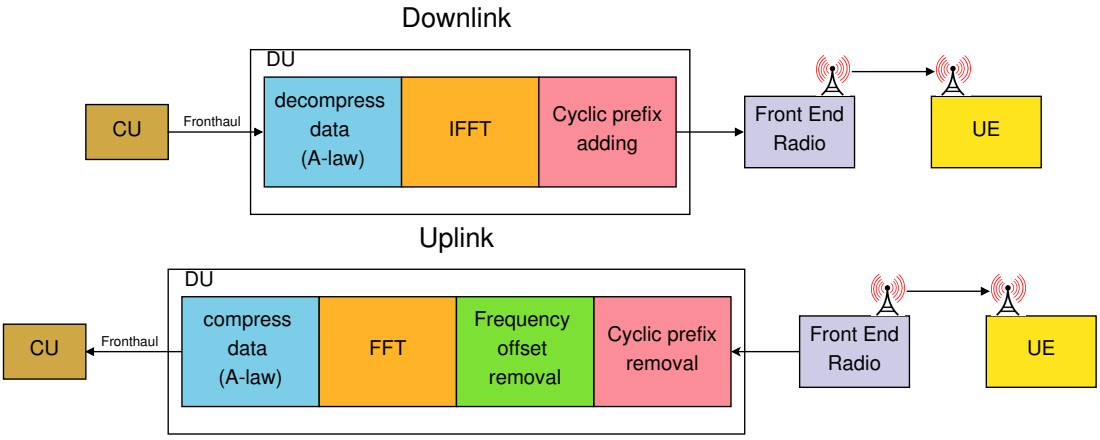


Figure 3.9. DU processing with Option 7-1 functional split

focuses on the Option 7-1 functional split which corresponds to implement the low PHY level functionalities at the DU while the other ones are implemented at the CU. Such functional split is implemented by the OAI experimental framework used to establish a 5G communication [**NMM+14**]. Many other 5G tasks can be also considered as good candidates to be accelerated into the hardware, up to entire layers of the 5G protocol stack. However, offloading a lot of processing into the hardware may result in a huge area required by the FPGA, as well as a costly solution has to be adopted. Thus, only the tasks that require strong computation capabilities with latency critical constraints should be deployed in hardware, exploiting the reconfigurable computing approach where the hardware represents the acceleration part, handled by the software.

Figure 3.9 shows the functionalities implemented in the DU both in downlink and uplink, considering option 7-1 split based on the OAI implementation. In downlink direction, the DU receives compressed the I/Q samples from the midhaul in the frequency domain. The data in the midhaul are compressed to save capacity and the DU applies the decompression algorithm based on the A-law [**Rec88**]. Each I/Q sample consists of 32 bits divided into 16 bits that represent the real part and 16 bits for the complex part. Then, the DU performs the inverse Fast Fourier Transform (iFFT) to convert the samples in the time domain and it adds the cyclic prefix, which is a guard interval to avoid inter-block interference between successive symbols. As last step, the DU sends the data to the radio front end, which has the role to do the Digital to Analog Conversion (DAC) and it handles the communication with the UE, which is a generic end-user device.

In the uplink direction, dual operations than the downlink direction are performed with the addition of the frequency offset removal. Indeed, such an offset is considered in baseband receiver to address the non-idealities problems such as sampling clock offset, IQ imbalance, power amplifier, phase noise and carrier frequency offset non-linearity. The processing described above is a part of the OFDM modulation/demodulation that is considered for 5G communications. In particular, the 5G OFDM processing is considered inside the DU to evaluate both the FPGA area usage and performance time for OpenCL offloading, thus the iFFT and the cyclic prefix addition kernels are deployed in hardware. Regarding the iFFT step, an integer algorithm based on the Cooley-Tukey approach [**CT65**] is implemented. To better understand the algorithm and its implementation, we can focus on the FFT instead of the inverse one since the iFFT can be calculated

by using the forward FFT and considering complex conjugation of the data before and after the forward FFT processing [**DPE88**]. Thus, in the following, we describe the iFFT considering the forward FFT. For the sake of simplicity, the forward FFT is deepened from time to frequency but the same process can be applied from frequency to time to complete the iFFT step. The FFT is based on three steps: i) decomposition of the signal in time, which consists in the division of a time domain signal composed by N points into N time domain signals each composed of a single point; ii) the second step considers the conversion of N time domain signals into N frequency spectra; iii) last step is the recomposition of N frequency spectra into a single frequency spectrum. Figure 3.10 shows how the decomposition works, starting from a 8 point signal. In this scenario, three stages are necessary to complete the decomposition. The first stage breaks 8 points signal into two signals of 4 points each. Then, second stage divides each 4 point signal in 4 signals of 2 points. Finally, the last stage permits to have 8 different point signals by breaking every 2 points signal in two parts. In general, with N points signal, this pattern can continue until there are N signals composed of a single point. Thus, the number of stages can be calculated by $log_2 N$.
The next step is to calculate N frequency spectra of a 1 point N time domain signals. This calculation is very easy since the spectrum of a 1 point time signal is equal to itself, thus no further computation is required to complete the second step. Finally, the third step is to combine N frequency spectra in the reverse order of the decomposition phase for a N points signal in time (first step). To easily understand this step (which is the most important of the FFT procedure), we can consider two time domain signals based on 4 points, *abcd*, *efgh*. To combine them and get an unique 8 point signal, we can add zeros to have 2 signals 8 points each, *a0b0c0d0* and *0e0f0g0h* and if we do the sum we get *abcdefgh*. Adding the zero to a 4 points signal corresponds to a duplication in the frequency domain, thus the final spectrum is the sum of the duplicated spectra *ABCDABCD* and *EFGHEFGH*. Moreover, the signal *abcd* is added with zeros located in the odd places (starting from zero to count) while the signal *efgh* is interleaved with zeros in the even places. In this scenario, the points *e*, *f*, *g*, *h* are also shifted to the right before adding the zeros. In the frequency domain, such a shift corresponds to a multiplication by a sinusoid. Figure 3.11 shows the final synthesis flow diagram to complete the third step, which is so-called *butterfly* due to its winged appearance [**S$^+$97**].

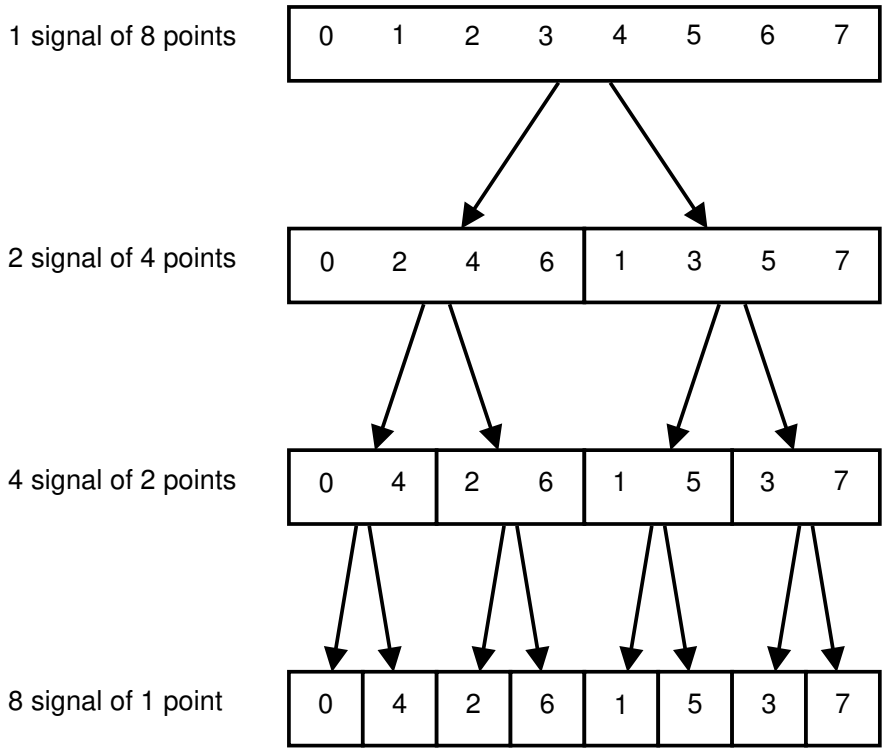


Figure 3.10. FFT decomposition

In the 5G environment, the FFT is used to process the I/Q samples during the OFDM modulation stage in the 5G architecture. The algorithm developed for the OpenCL application receives in input a number of samples equals to the OFDM symbol size. Since the iFFT is considered, all the samples are points of signal in frequency to be converted to a signal in time domain. To complete the OFDM processing at the DU side, the cyclic prefix addition kernel is offloaded to the hardware to prevent interference between OFDM symbols.
Figure 3.12 depicts the block diagram of both HDL and SIMD deployments. Figure 3.12a shows the block diagram for the SIMD approach. The data coming from midhaul is uncompressed and processed by the iFFT task based on SIMD instructions. This work considers the Intel Advanced Vector Extension 2 (AVX2) SIMD instruction set that performs operations on 256-bit vectorized data. Such instruction set achieves better performance than previous SIMD extensions, in particular with floating point calculation and data organization. The iFFT algorithm is computed by using the forward FFT and considering complex conjugation of the data before and after the forward FFT processing. Three SIMD

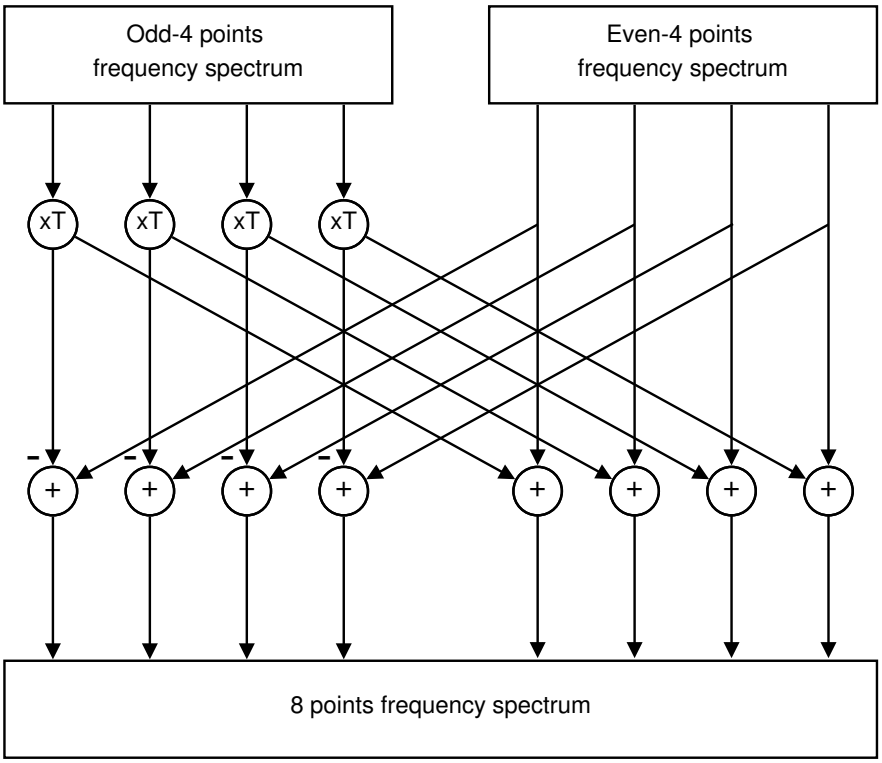
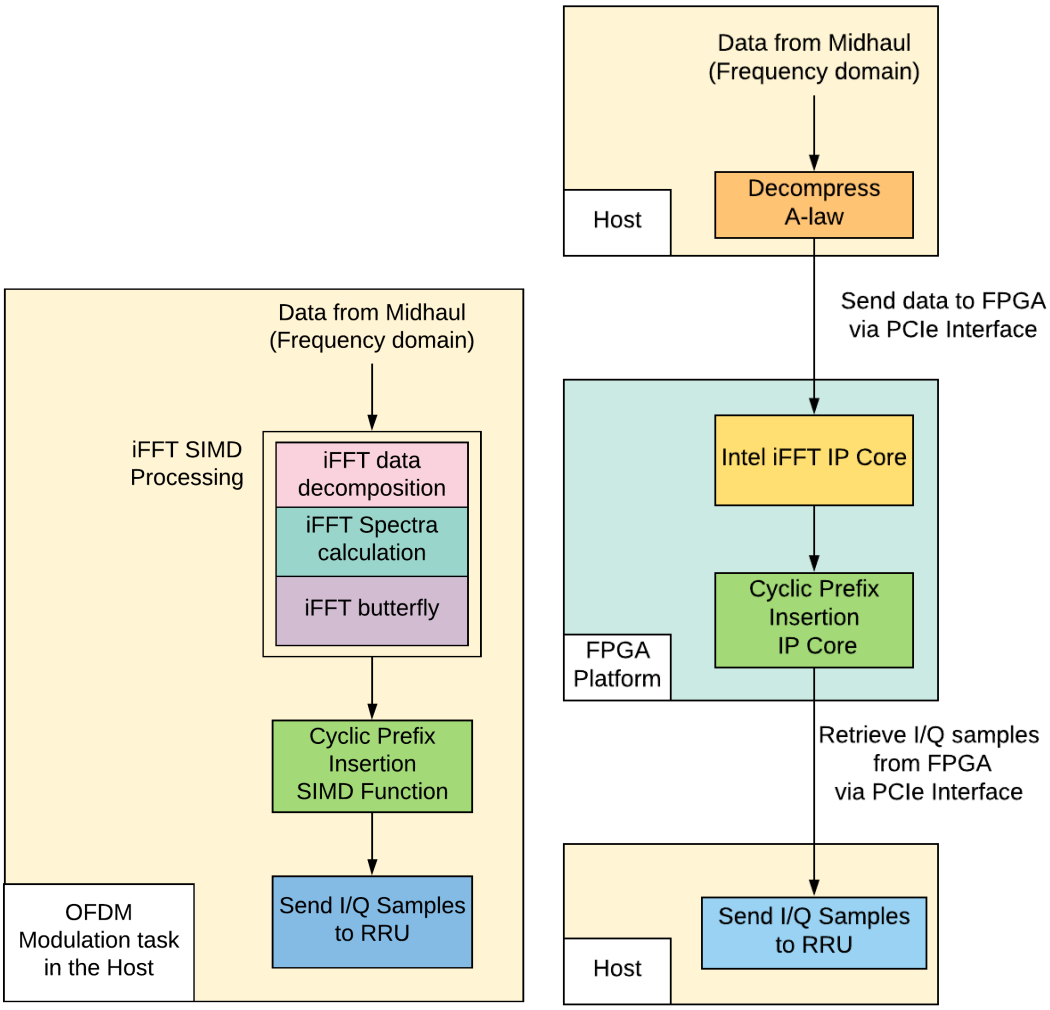


Figure 3.11. FFT butterfly

functions are implemented to address the integer FFT Cooley-Tukey algorithm, as described above: i) decomposition of the signal in frequency, ii) the conversion of N frequency spectra into N time signals and iii) the re-composition of N time signals into a single time signal (i.e., iFFT butterfly). The last step is the most significant one in terms of computation complexity due to totally connected data dependencies to previous steps. Each input sample of the FFT consists on 32 bit integer type where the first 16 bits represent the real part and the second 16 bits the imaginary part. As discussed above, the Intel AVX2 SIMD instructions can process 256 bit at once, thus 8 FFT samples are computed with a single instruction, ensuring a good degree of data parallelism. Since iFFT is considered, all the samples are points of signal in frequency to be converted to a signal in time domain. The output data is computed by the cyclic prefix insertion SIMD function that inserts redundant bytes before iFFT samples in the stream. The number of bytes to be added is summarized in Table 3.1 together with the complete OFDM 5G numerology considered for this work [**MBQB18**,**DPS16**]. Even if the present value considered as maximum symbol size is 4096 [**DGDLR18**],

the results up to 8192 symbol size are also evaluated to understand which are the better approaches to address a possible future implementation of this size, since the 5G PHY layer is still under study.
Figure 3.12b shows the top-level block diagram for the HDL use case. Here, the data in the frequency domain from the midhaul is uncompressed and sent to the FPGA by means of the PCIe interface. Then, the data is processed by the Intel iFFT IP core that returns the samples in the time domain. The streaming iFFT



(a) (b)

Figure 3.12. Architecture descriptions for a) SIMD and b) HDL implemented 5G DU processing

OFDM Symbol Size (N)	Cyclic prefix length (M)	Resource Blocks (RBs)	Channel Bandwidth (MHz)
128	10	6	1.4
256	20	15	3
512	40	25	5
1024	80	50	10
2048	160	100	20
4096	320	275	up to 200
8192	640	550	up to 900

Table 3.1. OFDM numerology

setting allows continuous processing of input data and it provides continuous complex data stream as output, without halting the dataflow. The streaming iFFT is based on quad output iFFT engine which minimizes the transform time. Quad-output refers to the throughput of the internal iFFT butterfly computing [**Int17**]. The engine implementation computes all four radix-4 butterfly complex outputs in a single clock cycle. The output of the iFFT block is sent to the custom IP for cyclic prefix addition. The IP core is synchronized at the output of the iFFT block to add the cyclic prefix bytes without adding clock cycles. The output of the design is a burst of samples in time domain composed by some redundant samples representing the cyclic prefix and the complex samples converted from frequency to time. The number of cyclic prefix bytes depends on the OFDM symbol size considered, as summarized in Table 3.1.
Regarding OpenCL, the 5G OFDM computation inside the DU is considered to evaluate both the FPGA area usage and processing time for OpenCL offloading. Thus, the iFFT and the cyclic prefix addition kernels are deployed in hardware, according to Figure 3.9. Dealing with the iFFT task, an integer algorithm is implemented, always based on the Cooley-Tukey approach. The same steps considered in the SIMD deployment are taken into account. In the OpenCL for FPGA scenario, the implementation of an integer algorithm is preferable to avoid excessive usage of DSP blocks of the FPGA. Indeed the floating point calculations require complex circuits based on adders, subtractors and multipliers that use DSP resources of the FPGA. The algorithm developed for the OpenCL application receives as input a number of samples equals to the OFDM symbol size. To complete the OFDM processing at the DU side, the cyclic prefix addition kernel

is also offloaded to hardware. It prevents interferences between OFDM symbols. Figure 3.13 shows the setup with the internal connection between host, FPGA and FPGA memory. The communication interface between the host and the FPGA is a PCIe that can interact directly with the global memory by means of a memory hardware controller. The PCIe bandwidth is around 4 GB/s. The maximum symbol size considered for this work is 8192 and each samples is 32 bit based. Thus, the maximum bandwidth for the OFDM used to transfer the data from the host to the FPGA is 262144 bits/s, significantly below the maximum PCIe bandwidth. The RAM of the board represents the global memory of the OpenCL environment and it can be accessible from the kernel pipeline, exploiting the global memory interconnect to interleave data accesses and hide latencies. The intermediate variables necessary for the kernel computation can be stored in both the RAM blocks (local memory) and the FPGA registers (private memory). In case of using RAM blocks, a local memory interconnect is necessary to access the memory itself.
Table 3.2 summarizes the characteristics of the Terasic DE5-Net board considered for this work as OpenCL device. Moreover, the host is based on the the processor Intel Xeon W-2133 equipped with 12 cores with 3.60GHz clock frequency.
The host is capable of both compiling and executing the kernel. Once the FPGA is programmed, the host can start the kernel and read/write the global memory

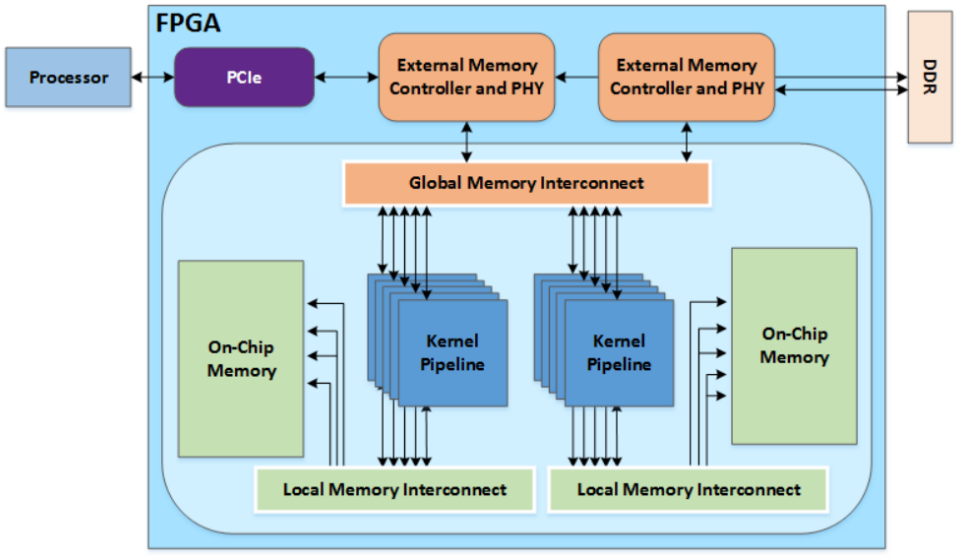


Figure 3.13. System setup

Board	Terasic DE5-Net
FPGA	Intel Stratix V GX
Resources	∼622000 LEs 256 DSPs
RAM	2x 4GB DDR3 @933 MHz each
Interface to Host	PCI Express (PCIe) x8 lanes

Table 3.2. Terasic DE5-Net board specifications

exploiting the OpenCL API.
The kernel considered for this work is SWI based and it is optimized to reach higher global memory throughput and to reduce memory latency transfer. Closing the gap between kernel frequency and memory controller frequency permits to achieve the saturation bandwidth, which is an important parameter to obtain the above optimizations. In particular, memory controller usually runs at $\frac{1}{8}$ of the clock of the external memory for DDR3 memory types. Considering the bus width of 64 bits for DDR3 memory, the memory bandwidth is saturated if the kernel runs at the same operating frequency of the memory controller ($\frac{1}{8}$ of memory frequency). Thus, the saturation bandwidth SB can be calculated with Eq.6, where BW is the bus width, N_{ddr} is the number of DDR banks, MF is the memory frequency and KF is the kernel frequency.

$$SB = BW \times N_{ddr} \times \frac{MF}{KF} = 64\ bit \times 2 \times 8 = 1024\ bits/clock\ cycle \qquad (6)$$

This value is the maximum SB achievable and it can be obtained under the following conditions: i) the kernel has to only have one read and one write memory access. Then, ii) no stalling shall occur, iii) the memory accesses have to be word aligned and iv) balanced readings and writings have to be done (the same number of bytes have to be read and written from/to the memory). Otherwise, if one or more conditions are not met, a significant overhead of computing cycle occurs and the execution time performance is degraded. These conditions permit to increase the OpenCL kernel frequency up to $\frac{1}{8}$ of the clock of the external memory. As described in Table 3.2, the Terasic DE5-Net board is based on two banks of DDR3 memory, 933 MHz each, with double data-rate. This means that we can consider 1866 MHz (i.e., 2×933 MHz) as external memory clock and the FPGA memory controller is $\frac{1}{8}$ of this value (i.e., ∼233 MHz). Hence, the

saturation bandwidth is achieved if the kernel frequency is equal to 233 MHz (at least, as close as possible). In this work, all the OpenCL kernels developed for the evaluation reach this working frequency, ensuring the maximum global memory throughput.

The kernels developed considers the variables storage inside the FPGA registers to reach lower processing time. If we consider on-chip memory (or block RAM) to store the kernel variables necessaries for the processing, several accesses are required to the block RAMs, reducing the processing time. For instance, unrolling loops with large unroll factor can cause memory conflicts on the on-chip memory, if a precise pattern for the accesses is not well-defined. The OpenCL compiler tries to optimize the accesses but area usage and latency increases due to the concurrence.

However, storing the kernel variables as registers permits to achieve higher performance since no memory access is required and no conflicts are possible for concurrent access. As matter of example, we can consider to store an array of 64 elements 16-bit each. The compiler considers 64 registers of 16-bit width. In this scenario, no conflicts are possible since every element of the array has its own register. This methods requires higher FPGA resources, in particular Adaptive Look Up Tabless (ALUTss) and Flip Flopss (FFss). The compiler could increase the size of registers in case of data dependency. Too many variables stored as registers can rapidly increase the overall logic utilization and the OpenCL design does not fit into the FPGA.

Many other optimizations are carried out to improve the kernel performance in terms of both improving processing time and reducing FPGA area usage. In particular, to reduce area usage, the function calls are avoided since every function is implemented as a circuit on the FPGA, resulting in excessive use of the resources. However, for the processing time, all the kernel loops are unrolled. This means that all the loops can be executed in one clock cycle but each loop is deployed in the FPGA as a single hardware circuit, resulting in a significant area usage.

Regarding the FPGA programming method, the Stratix V FPGA is based on the PCIe interface for the host communication. In our scenario, where the PCIe is the interface between host and device, three different methods are considered to program the FPGA. The first one consists of using the USB-JTAG interface to download the bitstream to the FPGA. This is an old approach and rarely used when the PCIe interface is present since it is slow and prone to errors.

The second method is Configuration via Protocol (CvP) and it is the fastest method since it exploits the PCIe, but it also needs the third method to be accomplished. The OpenCL framework for Intel FPGA is based on the Partial Reconfiguration (PR) and its architecture is composed by two different designs: *periphery* and *core*. The periphery is a design where all peripheral interfaces are instantiated and it represents the structure capable of accepting the core design of OpenCL. Thus, the core design is the OpenCL hardware design that can be reconfigured for different OpenCL application, while the periphery design remains the same. Moreover, before reprogramming the core design, the OpenCL programmer checks if the periphery design is currently programmed in the FPGA and, only if it is the same of the compiled configuration, the core design can be replaced. In this scenario, CvP is used to replace the FPGA kernel core design in the FPGA by means of the PCIe interface. Although CvP is faster, it is not suitable for periphery design programming that needs the third approach which exploits the flash memory. Indeed, both periphery and core design are programmed in FPGA flash and this method is used if the periphery design has changed or not already loaded. In this case, the FPGA is programmed with a flash loader that permits the direct communication between the host and the flash. Once the flash is programmed, both designs are loaded and the FPGA is ready to interact with the host. Programming the FPGA via CvP before programming the flash may not have effect on the FPGA since the periphery design has to be loaded to guarantee the communication via PCIe interface.
A further deepening is necessary on the core design to be programmed via CvP. Once the periphery design is ready, it is imported in all the core designs update revision compilations. Then, the compilation of the core design is proceeded and the system automatically creates a "kernel system" that contains IPs and a wrapper logic capable of interacting with the periphery design for external interfaces access.

5. Performance evaluation and results

To evaluate the offloading of 5G computation to reconfigurable hardware, three different performance parameters are considered: FPGA area usage, overall execution time and computational load. All the parameters refer to the implementation of the OFDM downlink processing for Option 7-1 functional split in the

DU part of the 5G architecture, as described in Figure 3.9. The first parameter permits to evaluate the FPGA resources needed to offload the computation. Moreover, the resource utilization for both HDL and OpenCL methods is reported to have a comparison between these two different approaches.
The second parameter is based on both processing time and memory transfer time performance. In particular, memory transfer time is evaluated for the both OpenCL and HDL approaches and it is defined as the time necessary to send the data from the host to the device and vice versa. However, the pure processing time analysis is done by comparing the OpenCL platform with HDL and software SIMD implementations. Moreover, a model is developed to predict the pure processing time values for those OFDM symbol sizes that not fit into the FPGA (for OpenCL case). Likewise, SIMD model is necessary to estimate the processing time for bigger OFDM simbol size since the present implementation of 5G OFDM considers symbol sizes up to 4096, as discussed above. The SIMD approach is already implemented in the OAI 5G framework, thus it can be considered as a benchmark to evaluate the processing time performance. Moreover, it has the best software performance for the iFFT computation [**Rod02**], which represents the most significant part of the OFDM processing. Both area and overall execution time parameters refer to different OFDM symbol sizes already implemented in the 5G architecture. As described above, the present value considered as maximum symbol size is 4096, but also the results at 8192 symbol size are evaluated to understand which are the better approaches to address a possible future implementation of this size, since the 5G PHY layer is still under study.
Finally, the computation load parameter is referred to the CPU processing load for both SIMD execution and OpenCL hardware offloading. In this case, the goal is to show if the OpenCL approach permits to save the CPU processing.

5.1. Resource usage evaluation. To evaluate the area usage of the FPGA, the resource utilization for both HDL and OpenCL approaches is reported. Area usage is reported in Table 3.3 where the results are presented for different OFDM symbol sizes. The OpenCL values refer to the resource occupation, given by the report generated during the OpenCL kernel compilation. Here, it is evident the resource usage differences, which reflect the different model application of HDL and HLD. Indeed, the FPGA resources used for the OpenCL implementation are much higher than the HDL cases. The reason lies in the abstraction layer

introduced by the OpenCL. However, the OpenCL resources could be lower since all the loops are fully unrolled to address the OFDM computation. This OpenCL programming method permits to improve the processing time to meet the 5G requirements, at the cost of significant increase of the area usage. In this scenario, the maximum OpenCL hardware acceleration is implemented for a 512 symbol size OFDM while, for higher symbol size, the OFDM does not fit.

OFDM Symbol Size	Logic Util (ALMs)		Registers		Total Memory bits		DSPs	
	HDL	OpenCL	HDL	OpenCL	HDL	OpenCL	HDL	OpenCL
128	2%	29%	1%	13%	<1%	7%	2%	24%
256	3%	51%	1%	25%	<1%	7%	2%	48%
512	5%	95%	2%	49%	<1%	8%	2%	86%
1024	9%	No FIT	3%	No FIT	<1%	No FIT	2%	No FIT
2048	16%	No FIT	4%	No FIT	1%	No FIT	5%	No FIT
4096	31%	No FIT	8%	No FIT	2%	No FIT	5%	No FIT
8192	60%	No FIT	15%	No FIT	3%	No FIT	5%	No FIT

Table 3.3. OpenCL and HDL FPGA resource usage

5.2. Overall execution time performance evaluation. To evaluate the overall execution time considered for the reconfigurable computing model, the pure processing time and memory transfer time between the host and the OpenCL device are considered. Regarding the memory transfer latency in the OpenCL scenario, around 233 µs are measured for all the OFDM symbol sizes, considering a CPU core dedicated to run the OpenCL kernel. This time represents a bottleneck due to the data transfer and synchronization between the host and the device memory as well as the readings and writings from/to the global memory (which is the DDR3 RAM for the considered board) and the FPGA. The transfer time in case of OFDM hardware offloading exploiting a single CPU core shared with other concurrent tasks (dedicated to both opeartive system and user applications) is around 468 µs for all the OFDM symbol sizes. Such a value is around the double of 233 µs, measured in case of ideal condition of CPU dedicated core, which demonstrates that this latency is due to software synchronization with the PCIe interface since the pure kernel processing time remains the same. The memory transfer time is calculated by subtracting the execution time of the kernel under exam with the execution time of a no operation loopback kernel (the data are sent from the host to the kernel and immediately forwarded by the kernel

to the host). Moreover, more readings/writings from/to the global memory can significantly increase the overall kernel execution time. For these reasons, in a possible implementation of the OpenCL approach for 5G, the PCIe interface between the host and the device represents a bottleneck that has to be avoided. A possible solution is to exploit the *auto-run* kernels and the OpenCL channels. The auto-run kernels permit to execute the processing in hardware without the interaction with the global memory. Indeed, the host starts the auto-run kernel which can process the data acquired by means of the I/O OpenCL channels. For instance, in the 5G context, such a kernel can gather the data to be directly computed exploiting the Ethernet or optic fiber hardware channels. Moreover, if more kernels are instantiated, the OpenCL channels enable the data transfer between the kernels without global memory interactions that increase the overall processing time.

Regarding the pure computation time without considering memory transfers, Figure 3.14 and Figure 3.15 show *performance ratio* trend for the SIMD, OpenCL and the processing time performance for the SIMD, OpenCL and HDL. Both plots are depicted with OFDM symbol size (from 128 up to 8192) as x axis. Here, the processing time curves are based on experimental results and estimated, except for the HDL case. As previously discussed, the PHY layer of 5G is still under study, thus the SIMD values are calculated according to the present symbol size. However, according to Table 3.3, OpenCL kernels does not fit into the hardware for OFDM symbol size bigger than 512. This means that for both OpenCL and SIMD is necessary a model to estimate the processing time up to 8192 OFDM symbol size to understand the behaviour of these deployments.

The model to evaluate the latency is based on the performance ratio calculated by taking into account the complexity of the OFDM processing. On downlink side, the DU have to compute the iFFT and the cyclic prefix addition to complete the OFDM. Since the iFFT is based on the Cooley-Tukey algorithm, the complexity can be calculated by $N \times \log N$ where N is the OFDM symbol size considered, while the complexity of the cyclic prefix addition is M where M is the number of samples to be added as cyclic prefix.

Table 3.4 shows the complexity of the both iFFT and cyclic prefix addition. The number of samples considered for the cyclic prefix are the same presented in [**3GP18**], while for 8192 OFDM symbol size a possible value has to be estimated. Since the complexity of the cyclic prefix is much smaller than the iFFT

OFDM Symbol Size (N)	Cyclic prefix length (M)	iFFT complexity 0(N*logN)	Cyclic prefix addition complexity O(M)
128	10	269.72	10
256	20	616.51	20
512	40	1387.15	40
1024	80	3082.55	80
2048	160	6781.60	160
4096	320	14796.23	320
8192	640	69049.06	640

Table 3.4. DU OFDM computation complexity for Option 7-1 split

complexity, the overall complexity can be considered equal to the iFFT one. The performance ratio is calculated dividing the measured processing time by the complexity. The trends are obtained by the interpolation of the performance ratio values calculated for different OFDM symbol sizes. To get the curves for the performance ratio, the Matlab curve fitting tool is used, choosing the equation that maximizes the *R factor*. Such a factor is defined as the square of the correlation between the response values and the predicted response values. R factor range is between 0 and 1. A value close to 1 indicates that a greater proportion of variance is accounted for by the model.
The OpenCL trend follows the exponential law while the SIMD trend can be considered as a second order hyperbola (Eq. 7).

$$\text{OpenCL} => ax^b + c \quad \text{and} \quad \text{SIMD} => a + \frac{b}{x} + \frac{c}{x^2} \tag{7}$$

Such models permit to estimate the processing time for both SIMD and OpenCL and they are depicted in Figure 3.14 in logarithmic scale, where it is evident the differences between the two trends. Indeed, OpenCL has a better processing attitude when the data size increases due to its pipelined approach. Moreover, the OpenCL performance are not expected to significantly increase since a fully parallelized kernel is considered. This approach results in a more resource usage, as shown in Table 3.3, but the performance considerably improve. However, the SIMD computation is lower for smaller data processing that reflects into an important distance between the two curve for OFDM symbol size less than

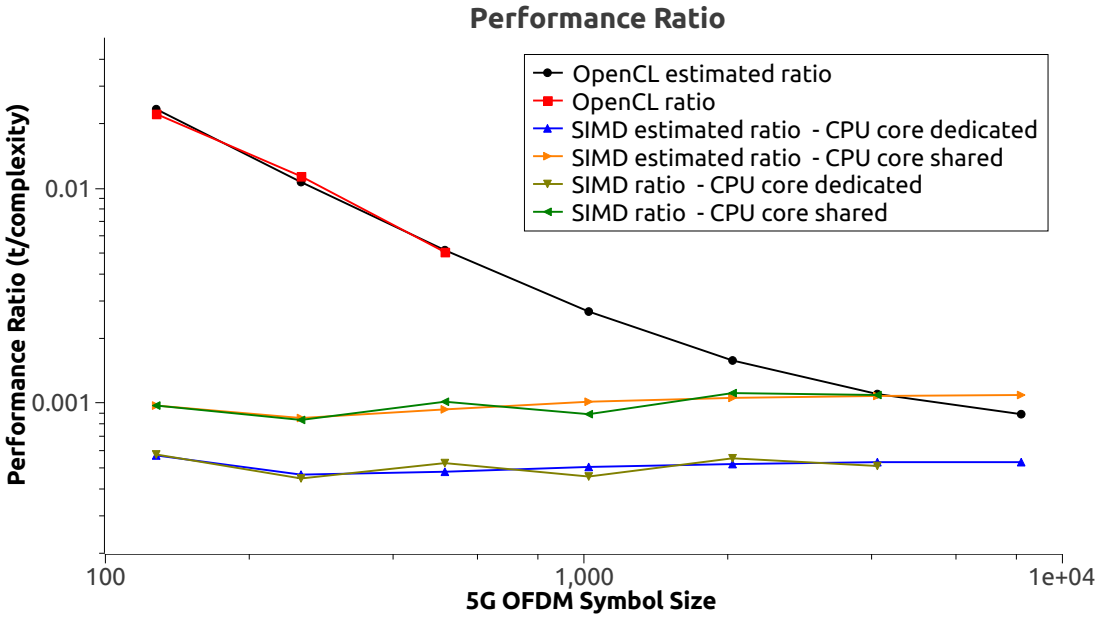


Figure 3.14. Performance ratio trend obtained by dividing performance time measured by OFDM complexity

2048. The SIMD model follows the sequential computing approach typical of the software that increases computation for large data batch. Performance ratio estimated values are calculated starting from the measured processing time, which permits to build part of the plot. In particular, the OpenCL ratio trend is also based on 16 and 64 symbol sizes, to have more points for a better performance ratio estimation, even if these are not possible OFDM symbol sizes.
Regarding the processing time, depicted in Figure 3.15 in logarithmic scale, the SIMD implementation has linear performance which reflects into a parabolic behaviour in normal scale. Moreover, for SIMD evaluation, two different models are depicted, which correspond to the computation of the OFDM task exploiting a CPU core dedicated and a CPU core shared with concurrent tasks. The performance time trend for SIMD considering a core dedicated is the best one after 2048 symbol size but it represents an ideal situation that is difficult to meet in the 5G. Indeed, one of the main novelty of 5G is the virtualization that require multiple threads to be executed in parallel, thus, for this analysis, the OFDM SIMD processing along with other parallel tasks is considered. Furthermore, the SIMD approach cannot be parallelized by means of multi-threading since SIMD and multi-threading are "orthogonal" solutions that consider a different level of

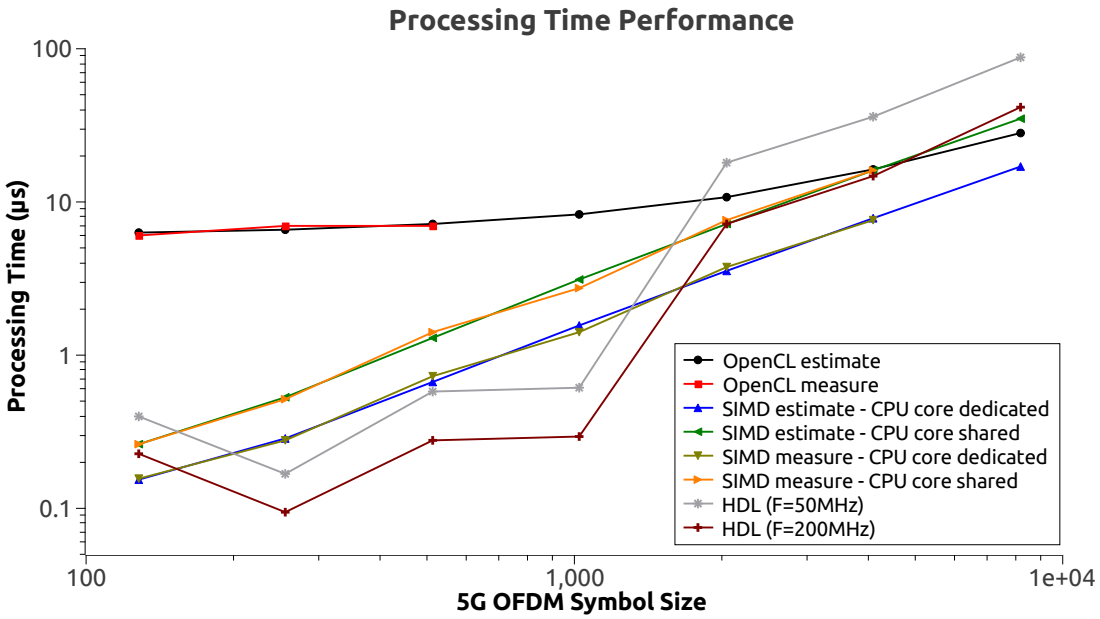


Figure 3.15. OpenCL, SIMD, HDL measured and estimated processing time performance

parallelism. Indeed, SIMD solution is a data parallelism where multiple data can be processed in parallel with a single instruction. As a matter of example we can consider a 256-bit SIMD vector that contains 8 OFDM samples 32 bit each, thus we can process 8 samples with a single instruction. However, the multithreading approach is task parallelism based that is taken into account when a large-scale parallelism is considered. This model tends to distribute the computation among multiple threads and not to multiple data. In this work data parallelism for both software is considered (i.e., SIMD) and hardware (i.e., SWI OpenCL kernel that considers fully unrolled loops), thus the comparison is based on the same type of parallelism. However, another approach based on both method is possible, the so-called Single Instruction Multiple Thread (SIMT), that shares the tasks among multiple cores and each task is executed exploiting the data parallelism (by means of SIMD instructions). This approach is common in GPGPU architecture or Xeon Phi processor, which is a series of X86 manycore CPU designed and made by Intel. Anyway, the OpenCL model also permits to distribute the computation among multiple threads, as discussed in the first section of this Chapter. Indeed, OpenCL is also suitable for GPGPUs to accelerate the application among multiple threads. In this scenario, the best performance can be

reached by considering NDRange kernels that automatically divides the application computation between many tasks to be separately executed in the GPGPU cores. In case of FPGA architectures, both task and data parallelism can be considered since concurrent tasks or multiple data can be anyway accelerated by means of dedicated hardware circuits.
About OpenCL curves, they show an almost constant behaviour (negative exponential in linear scale) and the performance are worse up to 2048 symbol size with respect to SIMD. However, the OpenCL trend (calculated from the performance ratio values) increases less rapidly than SIMD and they have the same performance at 4096 symbol size, while OpenCL could be a better approach after this value. As discussed at the beginning of this Section, running OpenCL kernel in parallel with other software tasks does not affect the hardware processing time but only the data transfer latency. Thus, the OpenCL processing time is the same in case of both CPU core resources shared and dedicated.
The same parameter considered for the OpenCL can be taken into account for HDL deployment. Indeed, the overall execution time can be divided in the throughput latency (i.e., memory transfer time) and the calculation latency (i.e., processing time). The first one represents the latency due to the data transfer towards the OFDM processing hardware IP and it corresponds to N clock cycles, where N is the OFDM symbol size. The calculation latency represents the clock cycles needed to do the processing. Figure 3.15 also shows HDL curves that only takes into account the calculation latency, which is the delay introduced by the OFDM computation task. Calculation latency is depicted in time domain after multiplying the clock cycles needed for the computation by the inverse of the FPGA operating frequency. Here, 50 MHz and 200 MHz working frequencies are imposed for the evaluation of the HDL curves: 50 MHz constraint as working frequency is respected while the 200 MHz is not met for OFDM symbol size bigger than 1024. Table 3.5 shows the frequencies reported by the Intel Quartus Timing Analyzer according to the Slow 85°C Timing model which provides timing delays for the FPGA operating under the conditions of i) slowest silicon for the specific speed grade, ii) low voltage and iii) 85°C junction temperature. Such conditions represent a possible worst-case for the device. Indeed, since each FPGA, which is cut from a silicon wafer, has delay differences, the timing analyzer takes into account the worst performance of a specified speed grade. Low

voltage decreases the transistor switching speed by decreasing electron mobility through the transistors. High temperature also affects transistor speed by changing the characteristics of the silicon material, leakage current, and electron mobility [**MW10**].

OFDM Symbol Size	FPGA Fmax (MHz)	
	50 MHz Constrain	200 MHz Constrain
128	131	231
256	126	222
512	100	209
1024	95	198
2048	78	194
4096	75	183
8192	71	150

Table 3.5. FPGA Timing analyzer frequency report for each OFDM symbol size

In Figure 3.15, it should be noticed that the HDL curves have better performance of both OpenCL and SIMD for OFDM symbol size less than or equal to 1024. After this values, the HDL computation increased and the SIMD processing time is the best of the three approaches up to 2048 OFDM symbol size. Starting from 8192 OFDM symbol size, both OpenCL and SIMD approaches could have less processing time.
Finally, a consideration about the technology is necessary. The Intel Xeon W-2133 used for the SIMD processing is produced with a 14 nm production technology while the Intel Stratix V FPGA is based on the 28 nm technology. This means that 14 nm FPGA (e.g., Intel Stratix 10 series) can achieve better performance in terms of both area usage (in particular for OpenCL approach) and working frequency that can significantly reduce the curves reported for the hardware scenario.

5.3. Computational load evaluation. For the last evaluation, the performance parameter considered is the CPU load. Such a parameter can be further characterized by three CPU states during the computation: *idle* state, *system*

processing state, and *user* state. The first one refers to the time that the CPU is idle and the system does not have outstanding I/O requests. The system processing state represents the CPU utilization that occurred to execute task at the system level (kernel). Moreover, this state refers to the interaction of the system with the external physical peripherals. The last state is the user state that stands for the CPU utilization while executing user level applications.
All the tests are performed considering a single core of the Intel Xeon processor. In particular, one core is unhooked from the operative system computation and exclusively dedicated to the processing of the OFDM algorithm.

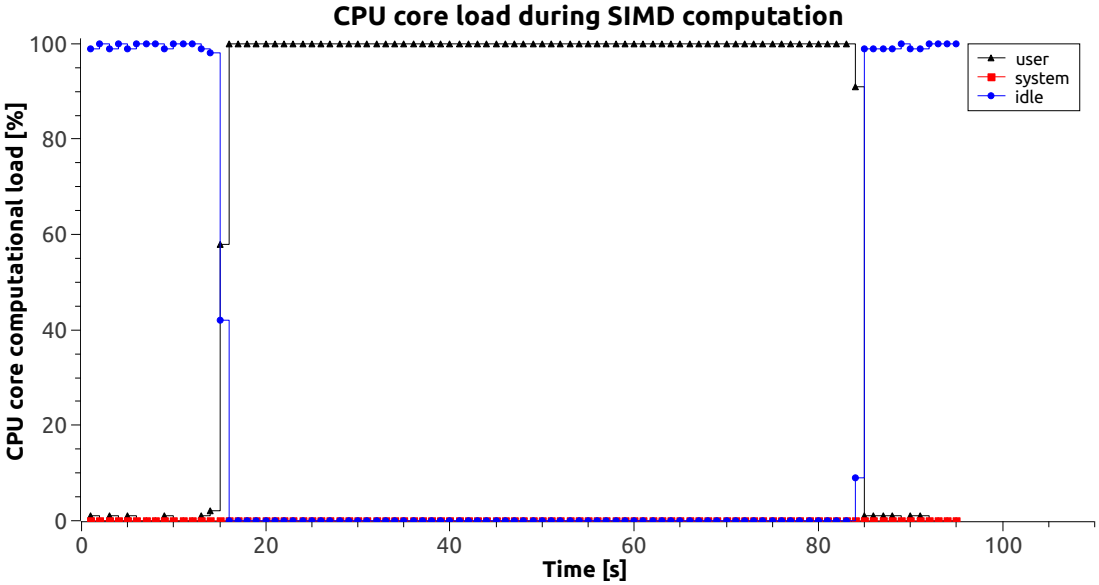


Figure 3.16. CPU core computation load for OFDM execution via SIMD (purely software)

Figures 3.16 and 3.17 report the results of the CPU load during computation of the OFDM via SIMD (purely software) and via OpenCL respectively, while Figure 3.18 shows the performance of the CPU during a switch from software to the hardware processing, triggered by a manager. Such a manager is developed to control the CPU state in terms of processing load. Thus, the hardware offloading is enabled when the CPU computation is heavy for long time. The logic of the manager considers a simple algorithm based on a threshold on the CPU load. The hardware acceleration is triggered once this threshold is overcame after one minute. However, more efficient algorithms that can take into account the average energy consumed by the CPU during the processing phase can be developed.

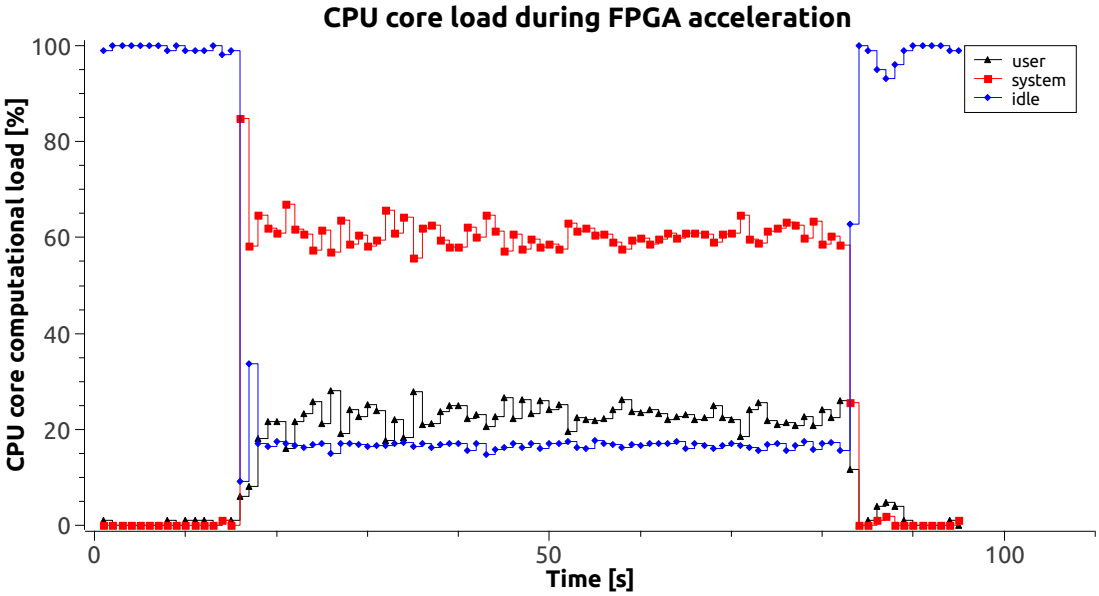


Figure 3.17. CPU core computation load for OFDM execution via OpenCL offload

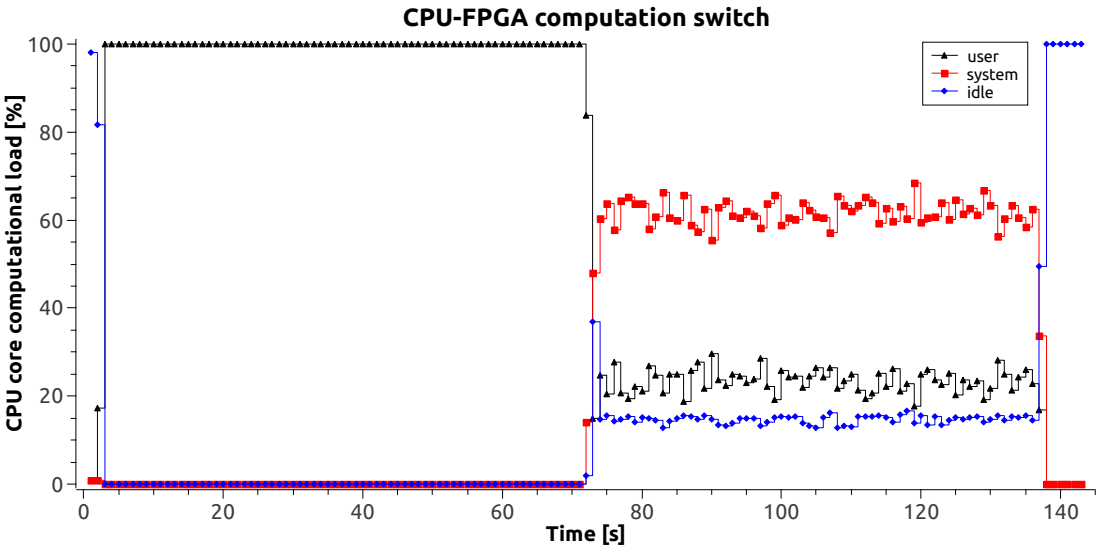


Figure 3.18. CPU core computation load during the switch from software to the hardware

Figure 3.16 shows the CPU load when the processing is purely software. Here, the CPU core is only in idle state at the beginning and at the end of the test. The computation load required by the SIMD is constant at 100% for the entire

duration of the OFDM execution. Moreover, the core is always in user state since no interactions with the external peripheral are required.
Figure 3.17 depicts the percentage of the CPU core during the OpenCL offload. In this scenario, the computation state is divided between the user state and the system state. Indeed, the user state is due to the host OpenCL application running while the system state reflects the PCIe interactions of the host with the global memory. The user application requires around 25% of the core load while the interaction with the PCIe interface is around 60%. The idle state is around 15% which is not found in the pure software use case.
Figure 3.18 show the transaction between hardware to software. Such a transaction is triggered by the manager after around 60 seconds of the CPU core utilization at 100%. Here, the differences between the two approaches are highlighted and the OpenCL approach reduces the computation load since the core remains in idle state for the 15% of the time. This means that the computational power of the CPU can be reduced by means of hardware offloading and part of the computation load can be reassigned to other tasks.
Finally, a consideration for the OFDM symbol size is necessary. This parameter evaluation is carried out with 256 and 512 OFDM symbol sizes and, in both scenarios, the results for SIMD and OpenCL are the same. Considering SIMD approach, OFDM computation always requires 100% of CPU processing. However, OpenCL approach requires the same CPU computation since all the processing is carried out in the hardware. Indeed, on the software side, the OpenCL application, that controls the FPGA execution, needs the same percentage of CPU resources in terms of both user state (i.e., OpenCL API application) and system state (i.e., interaction with PCIe) for all the OFDM symbol sizes.

6. Design productivity analysis with OpenCL acceleration

In this Section, the evaluation of the design productivity of deploying OpenCL kernel is addressed. According to [**PBMB16**], the performance of a High-Level Synthesis (HLS) approach is a trade-off between a quality system optimization (e.g., resource usage, memory accesses, etc.) and design efficiency. In particular, design efficiency can be expressed with several indicators like complexity of the design, the cost of the new code, the "developer friendliness" of programming language, the designer experience and the testability of results. These indicators

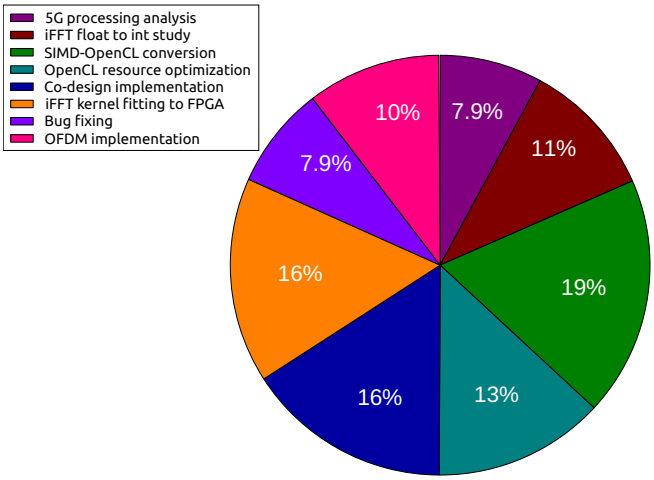


Figure 3.19. Design efficiency and implementation chart

can be also used "as is" for the HLD method considered in this work and they reflect into system integration time and design time. The first one concerns the verification and the validation of the the design and it depends on more electronic aspects like analog-to-digital and digital-to-analog conversions or electrostatic discharge that can damage the system. However, the design time refers to all the tasks necessary to create an application up to the deployment phase. This second parameter depends on the knowledge of the designer about both the system where the application has to be developed and the programming language and tools used to create the application. In this work, the time considered refers to a single developer with experience on the embedded systems but novice in the OpenCL programming model.

Figure 3.19 reports the percentage of overall time considered (3 months) to finish a precise task in the application development. The work starts with analysis of the 5G DU processing under the option 7-1 functional split. In particular, the most significant contribution, in terms of complexity, is represented by the iFFT Cooley-Tukey algorithm. Thus, some examples of its implementation in both float and integer representation are considered before starting the implementation (this task requires 7.9% of the development time). Since many OpenCL iFFT float implementations are available off-the-shelf, the second step considers the adaptation of the float version in the integer version of the iFFT. This step permits to understand if the off-the-shelf iFFT can be suitable for the OFDM

stage in the 5G DU processing. It requires 11% of the time.
Unfortunately, such a conversion needs a lot of work due to the conversion of each float operation into an integer one and the computation of all the twiddle factors. Indeed, twiddle factors are calculated at runtime in the float version by means of sine and cosine functions. In the integer iFFT considered in the 5G architecture, twiddle factors have to be constant values to avoid FPGA resource overhead or they can be computed at runtime after an offline conversion of the functions sine and cosine into integer lookup tables. Thus, a better way to address the integer iFFT design is to consider an OpenCL kernel created from the SIMD implementation of the integer iFFT. The SIMD version of iFFT is already deployed in the 5G communication framework. The replacement of such a iFFT in the 5G application can be easier thanks to the excellent integration of the OpenCL with the software. The SIMD-iFFT conversion to an OpenCL kernel requires 19% of the time.
The next two steps are related to the optimization of the iFFT kernel created in terms of area resources. Initially, it does not fit into the hardware since many optimizations have to be performed. The first step refers to an optimization of the loops inside the kernel and it takes 13% of the overall time. However, 16% of the time is required to move the iFFT from pure OpenCL kernel to a co-design approach. The co-design is necessary to achieve both the best performance in terms of processing time and a lower area usage. All the loops inside the OpenCL kernel are unrolled but this optimization requires many FPGA resources that leads to not fit the kernel in the hardware. Hence, a co-design method can help to reduce the area usage. In particular, the first step of the iFFT requires data reordering task that can be efficiently implemented by means of SIMD approach, without increasing the FPGA resources.
The next two steps refers to the compilation of the kernel into the FPGA and some bug fixing. Despite these steps may seem faster than the rest of the tasks, they take 23.9% of the overall time due to the Intel FPGA Quartus tool compilation time. Kernel compilation requires from some minutes up to several hours for OpenCL execution file generation, thus, this time is significant it should be taken into account by the designers before starting an OpenCL application.
The last step considers the development of the cyclic prefix addition kernel to be combined with the iFFT kernel to complete the OFDM modulation at the 5G DU side. This task takes 10% of the overall time.

Finally, several difficulties have been found in the conversion from SIMD to OpenCL and in addressing FPGA OpenCL improvement in terms of both area usage and execution time.

7. Conclusion

The goal of this work was to evaluate the viability of OpenCL for implementing some functionalities of a virtualized next generation eNB. To perform such evaluation, the implementation of the OFDM in the 5G DU with Option 7-1 functional split has been considered in hardware, software and co-design approach exploiting OpenCL model. The performance evaluation showed that OpenCL implementation is useful when virtualized functions shall be dynamically deployed in hardware or software for orchestration purposes. However, OpenCL shows some drawbacks with respect to the other considered implementations. Thus, it requires better optimizations and further study. In particular, OFDM kernels with OFDM symbol size up to 512 can be deployed in hardware by means of OpenCL, but bigger FPGAs or more pipelined loop have to be considered over this threshold. Furthermore, the pure kernel processing time is compatible with 5G latency performance but memory transfers between host and accelerator are the bottlenecks of the system when using PCIe. Around 233 µs have been measured, considering a CPU core dedicated to the processing. Likewise, around 468 µs have been measured in case of CPU core shared with other concurrent tasks. This behaviour is due to data synchronization from the host to FPGA in both directions, almost regardless of message size. For these reasons, the OpenCL kernels should be developed as auto-run to avoid the interaction with global memory and the PCIe data transfer, otherwise the overall execution time is not compatible with the 5G processing.

Only considering the pure processing time, OpenCL is more suitable for large data batch processing since results have shown that SIMD and HDL approaches have better performance up to 4096 OFDM symbol size. A model for SIMD and OpenCL performance curves has been evaluated to predict the processing time for bigger OFDM symbol size. Such models have shown that over 4096 OFDM symbol size the OpenCL implementation starts to have less performance time than both SIMD and HDL approaches. Thus, in case of a future implementation of a 8192 OFDM symbol size in the 5G PHY layer (the present implementation reaches 4096), the OpenCL model could be considered to implement the DU

processing. Moreover, the results are strictly related to the devices and the technologies considered, thus the hardware performance for both OpenCL and HDL can significantly increase with the latest FPGA technology.
Regarding the computational load, results have shown that the SIMD execution (purely software) require 100% of the computational load throughout the duration of the OFDM execution. However, with OpenCL offloading, the CPU core computation is limited at 85% and, for the 15% of the OFDM execution time, the core is in idle state. The evidences between the two approaches are also highlighted by the performance measured with the manager developed for the dynamic hardware offloading. These scenarios have been used to demonstrate that the computational power of the CPU shall be reduced and part of the processing can be reassigned for other tasks to deploy more services.

CHAPTER 4

Network functions acceleration at the edge

In this Chapter, two scenarios are considered to accelerate network functions at the edge of the datacenter architecture. The first one addresses traffic engineering and cyber security improvements in an edge node (i.e., NGCO of the network infrastructure). The second one relies on the hardware offloading of the processing functions by means of a hardware pipelined service chain. This case also targets the NGCO as deployment point in the infrastructure.

1. Edge node enabling traffic engineering and cyber security

The future convergence of next-generation wireless, wired and IT infrastructure is paving the way to the deployment of enhanced edge nodes [**vLYJ+17**, **KLL+17**]. In particular, edge packet-over-optical aggregation nodes are emerging to interface heterogenous local segments such as 5G RAN fronthauling, IoT gateways and IT Cloud/Fog clusters to the Metro-core transport network. These innovative technologies and platforms are expected to provide extreme QoS differentiation and high throughput at the same time, exploiting the SDN control plane and the orchestration of services and network resources across different segments, from the access to the optical backbone [**LJU+17**, **SPC+13**]. However, several issues may affect the utilization of current SDN solutions at the edge where a plethora of traffic flows with high throughput and dynamic profiles will need advanced TE-based treatment, due to stringent QoS constraints. Moreover, besides control and QoS, edge resources require to be secured against online cyber security attacks.

To address TE, packet-layer substantial innovations such as Segment Routing (SR) [**SPG+16**, **GSP+17b**] also enabling SDN/NFV Service Chaining [**Pao18**] and Network Slicing [**VGIZ+18**] have been proposed to automatically steer per-service traffic slices onto Elastic Optical Network (EON) lightpaths in the context of a multi-layer network employing efficient cross-stratum reoptimization and maintenance [**YZZ+15**, **PCCC17**]. However, such procedures rely on flows

with bandwidth reserved in the control plane only. In the data plane, bursty and unpredictable behavior of traffic forwarded by the edge node may be subject to different time-dependant profiles and statistical variations, that may induce bottlenecks, congestion, queue delay, thus affecting QoS (e.g., ultra-low latency requirements). Indeed, current SDN implementations do not support such required stateful-driven forwarding at wire speed directly at the nodes, whereas the controller is typically involved to react upon critical events. However, this may pose serious scalability issues at the controller and may noticeably delay reaction enforcement at the data plane, thus leading to serious forwarding inefficiencies (e.g., unexpected latency and jitter increase). In addition, at the data plane level, dedicated fixed-function hardware is not the best solution to address SDN flexibility and configurability requirements.

To address cyber security, Network Operation Centers (NOCs)/Security Operation Center (SOC) defend information systems at edge nodes using dedicated tools and systems like Intrusion Detection Systems, firewalls, Security Information and Event Management tools, enhanced with blockchain-based trustworthiness mechanisms as recently proposed for next generation 5G fronthaul [**YWZ+18**]. Usually, detection of attacks or resource bottlenecks is performed without automated tools and is based on systems logs, statistics and experience of operators. In addition, mitigating the attack in decentralized IT (e.g., fog resources) and edge nodes is not trivial and may require the reconfiguration of many network devices at once. Currently, traditional switching solutions are not able to deal with DDoS attacks and implement simple access/blacklist solutions, such as Border Gateway Protocol (BGP) flow spec traffic redirection [**MSR+09**]. However, they did not prove to be adequate against cyber attacks at network edges due to lack of effective context-based security analytics. In the standard SDN context, such attacks may be handled by the centralized controller. However, the default SDN behavior to redirect unknown packets to the controller may be extremely dangerous for the controller itself, being potentially exposed to attack floodings, out-of-service events and information update inconsistencies, especially in the case of stateful applications [**FGG+15**, **Sil18**].

The P4 technology has been recently introduced to enable advanced and configurable packet processing functionalities of network devices, supporting protocol and target platform independence [**BDG+14**, **P4**]. P4 is a high-level, platform-agnostic language for programming the data plane of SDN network devices. P4

allows to define customized and sophisticated switch pipelines, packet forwarding policies and actions, producing portable implementations over different hardware targets (e.g., network interface cards, FPGAs, software switches and hardware ASICs). Specifically, the availability of stateful programming objects, such as counters, meters and registers enables finite state machines and conditional behavior implementation directly in the hardware. The novelty and the potentials introduced by P4 has gained significant attention by many system vendors and the P4 consortium already involves more than 50 industrial partners. So far, the P4 community has been extremely active in developing and improving the P4 compiler itself and the P4 Behavioral Model software switch (i.e., BMV2). However, limited effort has been reported in the scientific literature to show potentially disruptive innovations and advantages of the P4 technology, especially in the context of multi-layer packet over optical networks.
This work proposes the adoption of the P4 technology in a SDN multi-layer packet over optical network to enable advanced data plane programmability. In particular, the work proposes an innovative edge node architecture including a P4 switch with native support of deep packet inspection. The P4-enabled node exploits direct stateful processing at wire speed, not demanded - as in OpenFlow systems - to the SDN Controller. Then, it proposes dynamic P4-based TE solutions for multi-layer scenario, such as traffic offloading and dynamic optical bypass. In addition, augmented firewalling capabilities are envisioned proposing a P4 DDoS mitigation proof-of-concept to protect internal edge resources without the need of dedicated firewall hardware. Finally, the evaluation include the P4 code proposals, along with their enforcement in a multi-layer edge node over two different platforms: the reference P4 software switch, namely BMV2 [**bmv17**], and the FPGA technology employing 10 GE optical interfaces at full rate [**WSD+17**]. Experimental results report the P4 impact in terms of latency, its scalability in terms of number of sustainable flow entries and its effectiveness showing online TE enforcement and fast detection against cyber-attacks.
With respect to our preliminary study [**PCC18**], this work introduces the following novel contribution:

(1) A presentation of the P4 language, along with related works, and its potentials in the context of multi-layer networks;
(2) A detailed architecture proposal of the P4-enabled packet over optical edge node;

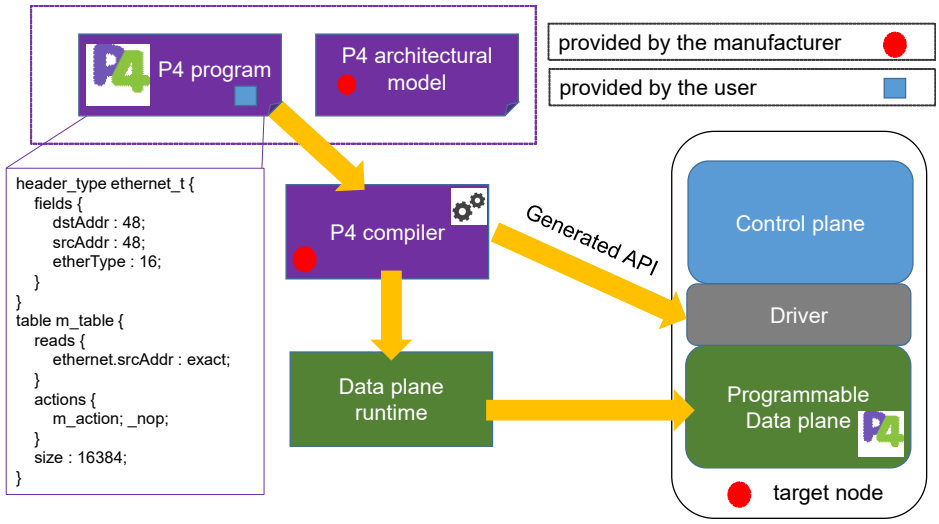


Figure 4.1. Workflow of P4 language compiler and API over programmable devices

(3) Extended experimental results including P4 over FPGA framework exploiting 10 Gigabit Ethernet optical interfaces at full line rate.

1.1. The P4 language. The P4 is a high-level programming language explicitly devoted to design the SDN data plane of packet processors [**P4**]. P4 has been conceived in the SDN paradigm, since some P4 proponents are the inventors of the OpenFlow protocol [**NPL$^+$15**]. According to the P4 proponents, such language aims at becoming the abstract programming language of a general purpose networking chip performing dedicated packet forwarding scheme based on the SDN paradigm. Following the concept of network disaggregation, P4 introduces open source programmability of network data plane, enabling own-made development of new proprietary protocols or headers, advanced forwarding and congestion control strategies, ad-hoc monitoring and telemetry functions without the need of costly dedicated proprietary devices or time consuming hardware firmware upgrades.

Compared to the state-of-the-art of packet processing systems, based on microcode, P4 provides the configurable building blocks of an abstract network node, ranging from Layer0 up to Layer7 functions: parsers (including non-standard headers), metadata (i.e., data that can be internally associated to a packet for processing, for example its input port), conditional controls, tables, along with a primitive set of actions. In addition, following the abstract forwarding model

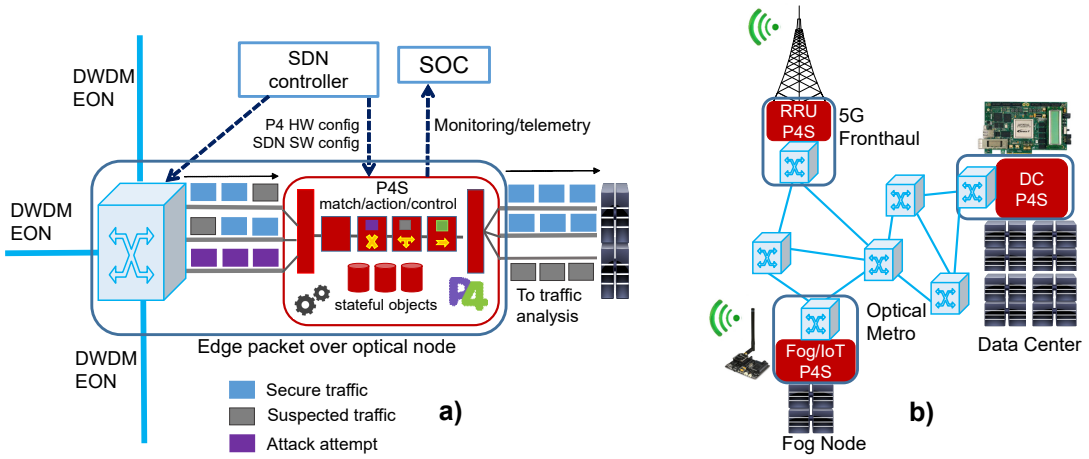


Figure 4.2. Packet-over-optical P4-based edge node: a) internal functional architecture and SDN control/monitoring; b) deployment and applicability scenario of P4 nodes acting as advanced forwarding devices and online distributed security barrier at the edge of the optical metro network

imposed by the language, packet-forwarding policies, algorithms and per-packet custom actions can be implemented producing portable implementations over different hardware targets (e.g., network interface cards, FPGAs, software switches, bare metal switches and hardware ASICs).
The P4 language operates within a high-level view of the general macro-blocks of the switch, called *abstract forwarding model*. The Protocol Independent Switch Architecture (PISA) is the current model and represents one of the most significant results of P4 research team driving the development of the P4 language itself. A comparison evaluation between the P4 implementation and the state-of-the-art of fixed-function switch hardware has shown that packet processing speeds are the same with almost no additional cost or power [**BDG+14**].
The P4 abstract forwarding model is composed by the following blocks:

(1) a programmable Parser block, responsible of identifying the stack of allowed protocols and fields defined by the program;
(2) a programmable Ingress pipeline, made of a set of match+actions tables, responsible of conditional packet processing and field update, egress port and queue selection;

(3) a programmable Egress pipeline, used for per-instance header modifications after egress port selection.

Figure 4.1 shows a simple example excerpt of a P4 program defining the Ethernet header and a table matching the Ethernet source address. Moreover, it shows the workflow of P4 programs compilation and hardware enforcement. The P4 program, written by the user utilizing the architectural model of the physical target, is compiled providing:

(1) a front-end representation (typically a JSON file) used to drive a back-end target-specific compiler for runtime data plane enforcement;
(2) an auto-generated runtime API to control the driver between control and data plane (i.e., to directly populate tables with flow entries following the P4 namespaces).

The P4 language defines a set of stateful objects that can be used to implement finite state machines and complex state-based decisions. Stateful elements store variables beyond the processing lifetime of a single packet, that may be read or updated depending on specified control conditions. In particular, two stateful constructs are available: *tables* and *extern objects*. Tables are read-only for the data plane, but their entries can be modified by the control plane. Extern objects have state that can be read and written by both the control and the data plane. In particular, among extern objects, registers (storing values), counters (storing incremental occurrences) and meters (storing rate values) may be instantiated. These stateful elements and their size are explicitly declared in the P4 code and allocated during the instantiation phase. This way, P4 can be used to dedicate pre-planned and dynamic countable hardware resources to specific functions and processes. Moreover, P4 enables to instruct stateful switches data plane with advanced functionalities with respect to standard OpenFlow switches, for instance implementing user-defined protocols or finite state machines.
Finally, a P4 program may be designed in a modular fashion with a baseline code structure and a set of extendible code pieces. As an example, P4 codes implementing basic router and switch functionalities have been proposed allowing incremental functions and support of protocols. In general, parsers, actions and even tables may be reused in a P4 code (e.g., merging two P4 programs implementing different TE techniques), thus enabling a number of alternative or parallel functionalities within the same switch. However, stateful objects require per-service instantiation, otherwise the monitored state may become inconsistent.

1.1.1. *P4 for reconfigurable hardware.* The P4 language is a flexible language that can be compiled for several architectures. Recently, some frameworks have been developed to support the deployment of P4 programs into programmable hardware targets [**IBMZ19**, **WSD+17**]. In this work, P4→NetFPGA workflow is considered, which uses the Xilinx P4-SDNet tools to directly compile the P4 programs for FPGA target. The idea behind this framework is to provide an affordable platform that allows data-plane programming in hardware at line rate. The architecture for P4→NetFPGA framework is based on a three stage pipeline switch: single parser, match-action pipeline, and deparser. This schema, shown in Figure 4.3, allows to implement many networking protocols/algorithms in flexible manner. The metadata bus contains the information about the packet length and the physical source and destination port where each packet respectively comes and goes to. The user metadata bus is used to exchange custom metadata inside the pipeline and the digest metadata bus is exploited to share the data with the host controller. The workflow generates both a C and Python API, built on top of P4-SDNet tool, to manipulate tables and read/write stateful memory on the switch. In addition to the API, the workflow also generates an interactive Command Line Interface (CLI) that allows the P4 users to interact with the switch in real time, as well as to query various compile time information about the switch.

1.1.2. *Related Work on P4.* A number of recent work rely on the P4 language to deploy advanced services such as telemetry, protocol implementation and stateless firewalling.

In-band telemetry applications of P4 resort to the collection of online network state parameters to be processed by the management plane [**HH17**]. The idea behind in-band telemetry is to collect flow-based direct measurements in the data plane (e.g., latency, queue transit time) using packet manipulation of non-standard headers (e.g., storing a timestamp value) that an external knowledge plan is able to process, also resorting to Artificial Intelligence and Machine Learning techniques. This allows to derive finer statistics for feedback-based automatic SDN intervention procedures.

An example of protocol implementation at run-time using P4 has been presented by the work in [**BHM17**]. The Bit Index Explicit Replication (BIER) is a novel SDN-oriented protocol proposed for multicast routing that requires a dedicated bit-indexed header encoding the multicast tree links selection [**GSP+17a**]. P4

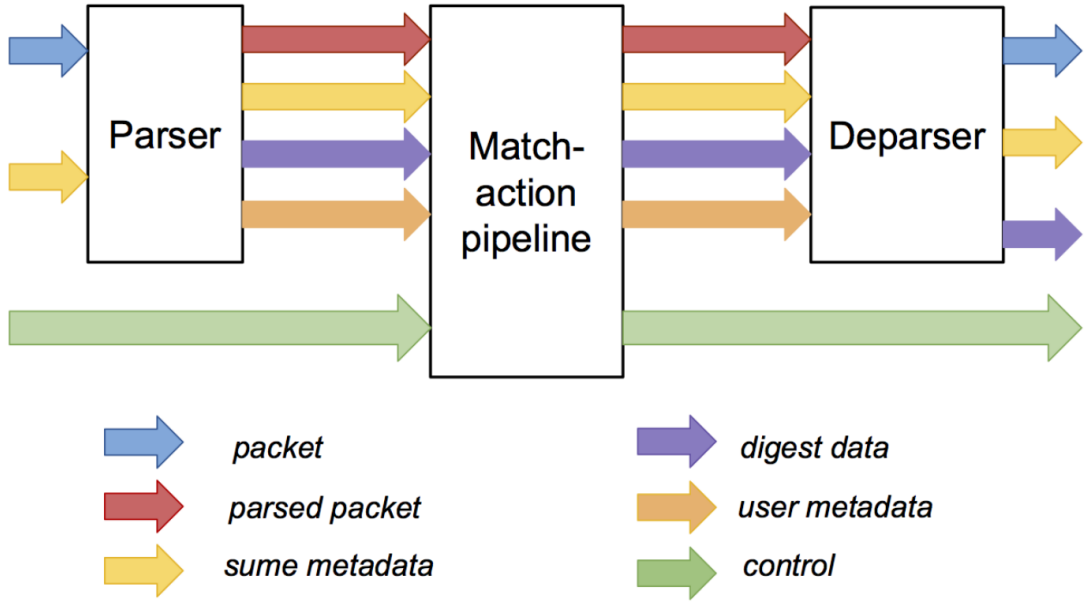


Figure 4.3. Block diagram of the P4 switch architecture used within the P4→NetFPGA workflow

allows a switch pipeline description implementing the header building and its encoding/decoding procedure.

Applications of P4 have been presented in the context of SDN security, however mainly limited to stateless firewall configurations including port/protocol filtering, blacklist and rate limiter [**VK16**]. Stateless P4-based header and packet header manipulation achieving mitigation techniques such as anti-spoofing mechanisms have been explored and analyzed [**ABBS17**]. All such approaches deal with active stateless processing of the header, without introducing simple finite state machines and history-based processing.

The most important work on FPGA and P4 is presented in [**WSD+17**], in which authors developed an open source compiler and runtime for P4 over FPGA and evaluated in deep detail the performances of some P4 programs in terms of hardware resource occupancy and latency compared with fixed function ASIC. Proposed P4 programs relies on L2/L3 forwarding and complex protocol implementations in the context of market data and distributed computation agreement.

Finally, stateful SDN data plane programmability by means of alternative strategies and tools besides P4 have been proposed and discussed concerning efficiency and security [**DCA+17**], among which one of the most interesting and significant

proposes the extension of the OpenFlow protocol supporting finite state machine abstraction compatible with table-oriented API [**BBCC14**, **CPSC15**].
This work focuses on the application of P4 stateful capabilities exploiting advanced TE and cyber security in multi-layer edge nodes with hardware acceleration.

1.2. P4 in multi-layer edge nodes. This section introduces the application of the P4 language to a multi-layer packet-switched (e.g., IP/MPLS or Carrier Ethernet) over optical-switched (e.g., DWDM or EON based) edge node with advanced and programmable SDN forwarding plane. While requested data is becoming closer to the user (i.e. fog nodes), attached edge nodes connected to the metro or metro/core network need a more refined treatment of selected class of traffic requiring QoS and TE (e.g., strict latency requirements), subject to profile statistical modifications or high burstiness behavior. The architecture of the edge packet-over-optical node encompassing P4 programmable data plane is depicted in Figure 4.2-a. The optical part comprises a SDN-controlled ROADM (e.g., disaggregated whitebox) [**VSC$^+$18**, **SIZG$^+$18**] with its tributary cards attached to a P4 switch (P4S), representing the key packet-switching element of the edge node. A number of P4S optical interfaces are connected to the ROADM cards, while the remaining interfaces connect local or internal resources. The P4S is hardware-programmed by the P4 language and is handled by a SDN controller/orchestrator, responsible for table entries population and service deployment. Multiple functions may be programmed at the same device, including TE/QoS features (e.g., latency-aware forwarding, dynamic offloading or bypass) and security applications (e.g., block, mitigation, telemetry and anomalies reports to SOC, suspected traffic deviation). Reporting data, statistics, alarms, telemetry functions to a Monitoring Handler/SOC are also programmable inside the P4S, enable possible integrated multi-layer proactive monitoring infrastructure. For example, P4 in-band telemetry [**HH17**] combined with optical layer advanced monitoring realized in the context of disaggregated networks [**PSCC18**] may be integrated in a joint multi-layer telemetry system.
Different potential P4-based edge node deployment scenarios are envisioned, as illustrated in Figure 4.2-b. In particular, such extended node may be placed as Data Center (DC) gateway, edge node device (e.g., fog node or IoT gateway), 5G fronthaul node (e.g., extended Remote Radio Unit - RRU supporting Fiber to the Antenna technology). For example, in the 5G Fronthaul scenario, specific P4

switches may be employed to perform online traffic telemetry and implement precise latency-assured traffic forwarding (e.g., dynamically manipulating the IP/MPLS QoS service flag thus driving stateful scheduling with priority). In addition, such node may be employed even at intermediate network nodes (e.g, aggregation metro node). In this case, traffic engineering solutions may be enabled not only at the edge but also in the metro network. In the case of cyber security solutions, this choice may avoid that security threats reach the edge and frees resources to better focus on more sophisticated attacks. The P4 solution allows a unique SDN edge switch deployment with advanced TE and security functions, detailed in the next sections, avoiding processing burden at the controller and additional specialized TE and firewall hardware inside cloud/fog node.

In multi-layer optical networks, TE techniques such as optical bypass are possible through instantiation of optical paths and steering of traffic based on policies that can be configured by a SDN controller using specific flow entries. However, such TE techniques and traffic steering configurations are typically enforced with static or stateless match conditions. When traffic conditions change such policies are typically modified by the controller. However, this requires the deployment of complex monitoring and telemetry techniques at the controller, possibly incurring in severe scalability issues in the case of high volume traffic, e.g., as in the case of DC gateway. The availability of a programmable data plane enables the deployment of advanced traffic engineering solutions (e.g., stateful TE) at SDN devices without requiring the intervention of a controller. In addition, the presence of a stateful SDN device capable of providing complex monitoring/telemetry information and alerts may drastically help the SDN controller for both scalability (i.e., reduced number of monitoring polling messages) and operation performance (i.e., accurate and fast detection of anomalies) and new type of networking statistics such as min/average/max latency spent in queue). Telemetry data may be elaborated by an external telemetry collector interacting with the SDN controller, allowing the latter to immediately react without being overwhelmed by excessive monitoring messages.

Several use cases may strongly motivate the adoption of a stateful P4 switch in the architecture of an edge node:

- Advanced Traffic Engineering (e.g., dynamic traffic-based routing, massive load balancing in segment routing applications);
- Cyber Security mitigation and intrusion detection;

- QoS precise forwarding (e.g., ultra-low latency for 5G, lossless Ethernet, automatic packet reordering);
- Advanced monitoring solutions (e.g., active probe generation for fast failure detection or forecasting, in-band telemetry);
- Packet header customization and manipulation for online service differentiation (e.g., mice and elephant flows header differentiation in DC scenarios).

In the next sections, advanced traffic engineering and cyber security mitigation use cases in the context of a multi-layer edge node will be presented targeting the P4 technology as candidate solution.

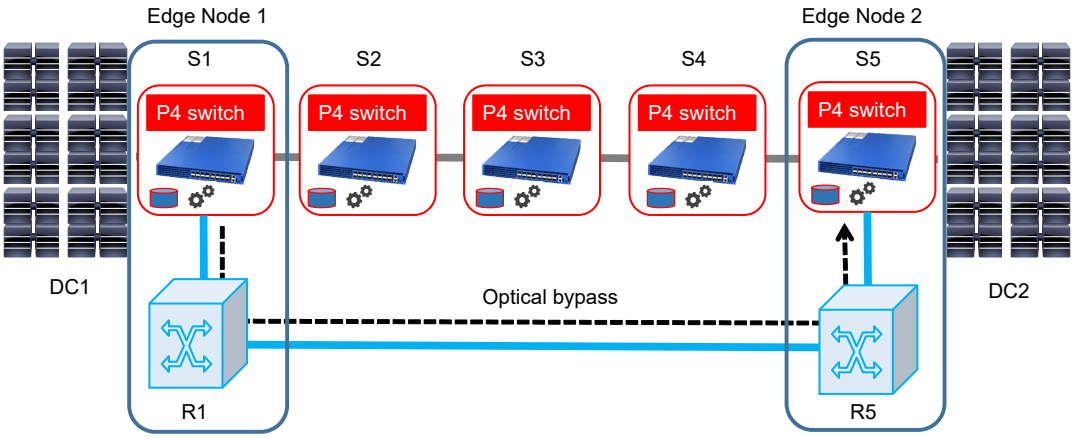


Figure 4.4. Advanced traffic engineering use case: data center gateways equipped with P4-based edge node performing dynamic TE

1.3. Stateful traffic engineering with P4. Figure 4.4 shows an inter-data center connectivity use case, where DC1 and DC2 are connected by a packet-switched path and by an additional packet-over-optical path exploiting optical bypass. Traffic originated by DC1 and destined to DC2 follows the flow rules installed in the S1 switch. Stateless flow rules may be enforced to a standard SDN switch (e.g., OpenFlow switch), based on specific packet attributes. As an example, latency-sensitive traffic (e.g., matched on protocol/application type) may be steered to the optical bypass. However, forwarding dictated by dynamic traffic conditions and profiles (i.e., stateful TE) is not feasible using a standard OpenFlow switch, without the active involvement of the controller. Programmable

```
[PARSERS]
.....
header_type meta_t {
    fields {
        meter_tag : 32;
    }
}
metadata meta_t meta;
meter my_meter {
    type: packets;
    static: m_table;
    instance_count: 16384;
}
action m_action(meter_idx) {
    execute_meter(my_meter, meter_idx, meta.meter_tag);
}
action steer_port(steerport) {
    modify_field(standard_metadata.egress_spec, steerport);
}
table m_table {
    reads {
        ethernet.srcAddr : exact;
    }
    actions {
        m_action; _nop;
    }
    size : 16384;
}
table m_filter {
    reads {
        meta.meter_tag : exact;
    }
    actions {
        steer_port;_drop; _nop;
    }
    size: 16;
}
control ingress {
    apply(m_table);
    apply(m_filter);
}
.....
[EGRESS]
```

Metadata

Meter definition

Actions

Flow tables

Pipeline control

Figure 4.5. Dynamic traffic offloading P4 code based on meters and token-bucket

data plane enables stateful TE. Two examples are provided in the following subsections: dynamic traffic offloading and dynamic optical bypass based on stateful traffic conditions.

1.3.1. *TE: Traffic offloading.* In the first TE use case, traffic offloading is implemented. An incoming traffic rate threshold *TH* is considered. The objective is to dynamically reroute just the portion of traffic beyond the rate threshold along an alternative path, i.e. implementing traffic shaping and limiter and avoiding congestion. To enforce such use case, a P4 program is exploited to build the forwarding plane of the switch. The program first defines the required packet headers (i.e., Ethernet and IP headers), then relies on a pipeline of two Ingress flow tables and on a stateful structure provided by P4, the *Meter*. As defined in [**HG99**], Meters are three state markers (i.e., output states are red, yellow and green) based on the definition of two rates, the Committed Information Rate (CIR) and the Peak Information Rate (PIR). Marker is set to red if rate exceeds the PIR, to green if rate is below the CIR, to yellow if rate is between CIR and PIR.

The traffic offloading P4 code key sections are reported in Figure 4.5. First, packets are parsed to check the protocol stack. In this case Ethernet frames are checked and Ethernet fields saved by the parser (not shown in the figure). Then packets enter the ingress section, designed by the *control* tag, where the tables pipeline is defined. Two tables are defined: the *m_table* and the *m_filter*. In the *m_table*, packets are inspected by their source MAC address (i.e., *ethernet.srcAddr*) and, in case of match, the *m_action* is executed. In the P4 program, an array of meters is declared and instantiated (*my_meter*). The *m_action* triggers the related meter identified by the *meter_idx* index and saves the result in a packet metadata field. Metadata fields are defined to carry out specific information related to the packet. Standard metadata are already defined in P4, such as the packet output interface (i.e., *standard_metadata.egress_spec*). The program defines an additional metadata providing the meter execution result (i.e., *meter_tag*). Then, the packet is passed to a second flow table (*m_filter*) which applies a token-bucket behavior according to its current Meter value, selecting between either the default or the alternative output port (i.e., through the *steer_port* action). In both cases, forwarding rules are dynamically applied according to actual traffic conditions, with no Controller intervention. Indeed, P4 dictates the general switch behavior abstracting from the entries of the flow

tables. This means that the behavior of the switch can be applied to different matching conditions just updating the flow entries of the P4 switch, without the need of reprogramming it. For example, the *steer_port* action identifies the output port of the packet using the *steerport* parameter. Thus, in the case of topology changes (e.g., an additional ROADM is added with the possibility of implementing two rates meter), the update of the output port for yellow and red traffic may be simply re-adapted by modifying at runtime the flow entry of table *m_filter*. In addition, the meter *TH* threshold is configured as configuration entry of the meter at runtime, therefore it can be tuned by network administrator or by the controller during the switch functioning. It is worth to note that the whole P4 program, including parsers, meters, actions, tables and pipeline control sections, is less than 100 lines of P4 code.

1.3.2. *TE: Optical bypass.* In the second TE use case, the edge node implements dynamic optical bypass. An incoming traffic rate threshold *TH* is considered. When DC1-DC2 traffic rate remains below threshold *TH*, traffic is forwarded along the P4 switches chain in the packet-switched layer. Conversely, when traffic exceeds the threshold, all the matching packets are automatically steered to the optical node R1 which injects traffic along the pre-established optical path between R1 and R2. To implement such use case, a P4 program builds the internal structure and describes the forwarding workflow of the edge node. The key sections of the P4 program are illustrated in Figure 4.6. The program defines a metadata *flowlet_meta_t* with two fields to store the timestamps of the input interface of the switch related to the packet itself and the previous one. In addition, an array of registers *flowlet_reg* are defined and instantiated. The control relies on a pipeline of two flow tables of type Ingress (*m_flowlet* and *m_bypass*), however, in this case, the pipeline is not static but subject to conditions. The first table is always executed to match the considered traffic flow according to configured parsing conditions (e.g., source/dest MAC/IP etc, in this case source MAC address). In addition, when matching conditions apply, the *flowlet_action* action is performed. Four commands are executed in the action: the timestamp of the previous matched packet is read from the register and saved in the packet metadata (*previous_ts* field); the current packet timestamp is saved in the metadata (*current_ts* field); the current timestamp is written and stored in the register; the default output interface is selected (e.g., port towards S2). Therefore, the P4 stateful register is used to store in a vector previously collected

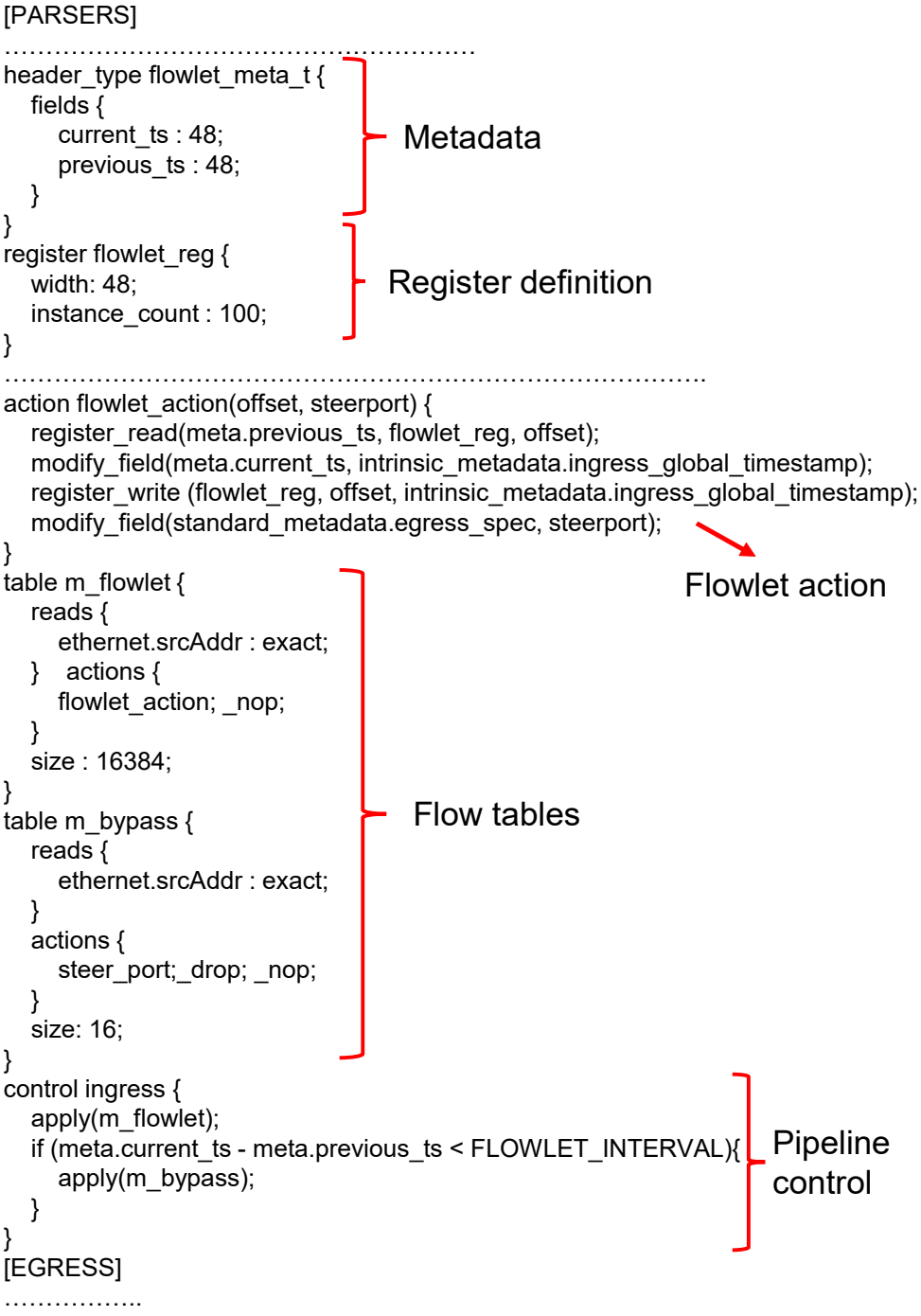


Figure 4.6. Dynamic optical bypass P4 code based on registers and flowlet switching

timestamps. This way, frame rate and traffic profile (e.g., flowlet) can be assessed driving to specific forwarding conditions. This is obtained by defining a pipeline control behavior subject to internal matching conditions. In particular, a P4 Control condition is set to either exploit the second ingress flow table *m_bypass* in the case of flowlet match (i.e., the interarrival time between matching packets is below *TH*, the constant *FLOWLET_INTERVAL* defined in the code), which applies the *steer_port* action, changing the output port value. This activates optical bypass forwarding of matched frames towards R1, or maintaining the default output port towards S2. Also in this case the whole P4 program is less than 100 lines of code.
Note that the optical bypass outgoing port is selected by the parameter of the *steer_port* action. Such value is stored as flow entry of the *m_bypass* table and can be modified at any time (e.g., by the P4 switch command line interface or by the SDN controller). This means that optical bypass selection itself may be adapted to traffic conditions or network status. For example, the SDN controller may decide to steer traffic to another available optical bypass, thus just requiring a single entry update at the P4 switch. Moreover, thanks to P4 stateful objects and internal telemetry, based on steered traffic statistics, the P4 switch might trigger, through the SDN controller, a lightpath adaptation request to the optical control plane (e.g., elastic operations to an active stateful Path Computation Element in the case of additional bandwidth requirement [**PCF+15**] or predictive analytics on traffic flows [**MGP+17**], physical parameter adaptation in the case of poor QoS [**MPS+11**]).

1.4. Cyber security mitigation with P4. The same network scenario shown in Figure 4.4 and the node architecture of Figure 4.2-a are also exploited to show how the P4 technology can be efficiently used to react against cyber-attacks. As an example of cyber threat, a DDoS attack exploiting address/port scan is considered. All possible TCP/UDP ports of a target IP destinations are attacked from multiple infected IP source nodes. Such type of attack can not be simply blocked by access/blacklists, since this could affect also legitimate remote connections. That is, such type of attacks can not be blocked by using traditional OpenFlow switches where just basic stateless permit/deny flow matches are practically available.
Conversely, the stateful nature of P4 provides innovative solutions to directly address such critical threats within the network nodes, by detecting attacks by

means of deep packet inspection and packet sequence correlation at runtime. The designed P4 program, besides including TE solutions, also relies on P4 Registers to store header information (e.g., IP dest and TCP/UDP port along with the related timestamp) for a number N of previously received packets. Then, P4 Control conditions can be configured to analyze the retrieved data and identify possible ongoing port scan attacks. This way, packets normally directed to the default output port can be temporarily blocked, successfully dropping such attempts for a configurable amount of time, or redirecting suspected traffic to dedicated stateful firewalls thus implementing attack mitigation. Such functions are directly implemented at the switch, as before, without involving the SDN controller with excessive amount of packets, which typically happens in DDoS attacks impacting controller's stability and functioning.

1.4.1. *P4 application: DDoS.* The proposed P4 switch edge node is implemented as proof of concept against the TCP SYN flood attacks [**MR04**], enforced on both software switches and programmable hardware functionalities of FPGA. Figure 4.7 depicts the related P4 workflow. The parser section of the program defines the rules to parse incoming packet. All ingress frames received at a given interface, coming from external hosts are first parsed to detect the protocol stack. In particular, Ethernet framing, (optional) IP parsing and (optional) TCP parsing are performed in cascade. After this step, forwarding is applied by checking a Forwarding Table (defined in the P4 program) and the physical egress port is assigned, based on the destination address of the current packet. Based on these fields, the switch detects whether a packet comes from a suspicious host or it belongs to a suspected traffic profile (IP match table may be populated by a centralized security controller). Then, the program enters the control section and checks if the packet is a valid TCP SYN and, if not, forwards it. Otherwise, it parses the TCP port to detect anomalous behaviour. To check anomaly, stateful evaluation of the session is performed. In particular, if TCP destination port is incremented compared to the previous TCP port of the same session (i.e., the basic TCP SYN flood mechanism) a stateful object is accordingly updated to store and keep updated the session state.

In this case, two register variables are allocated per IP Match Table entry. The first variable stores the TCP port received by the last packet belonging to the session, whereas the second variable stores the number of attempts matching the TCP SYN Flood basic behavior. The two registers are continuously updated upon

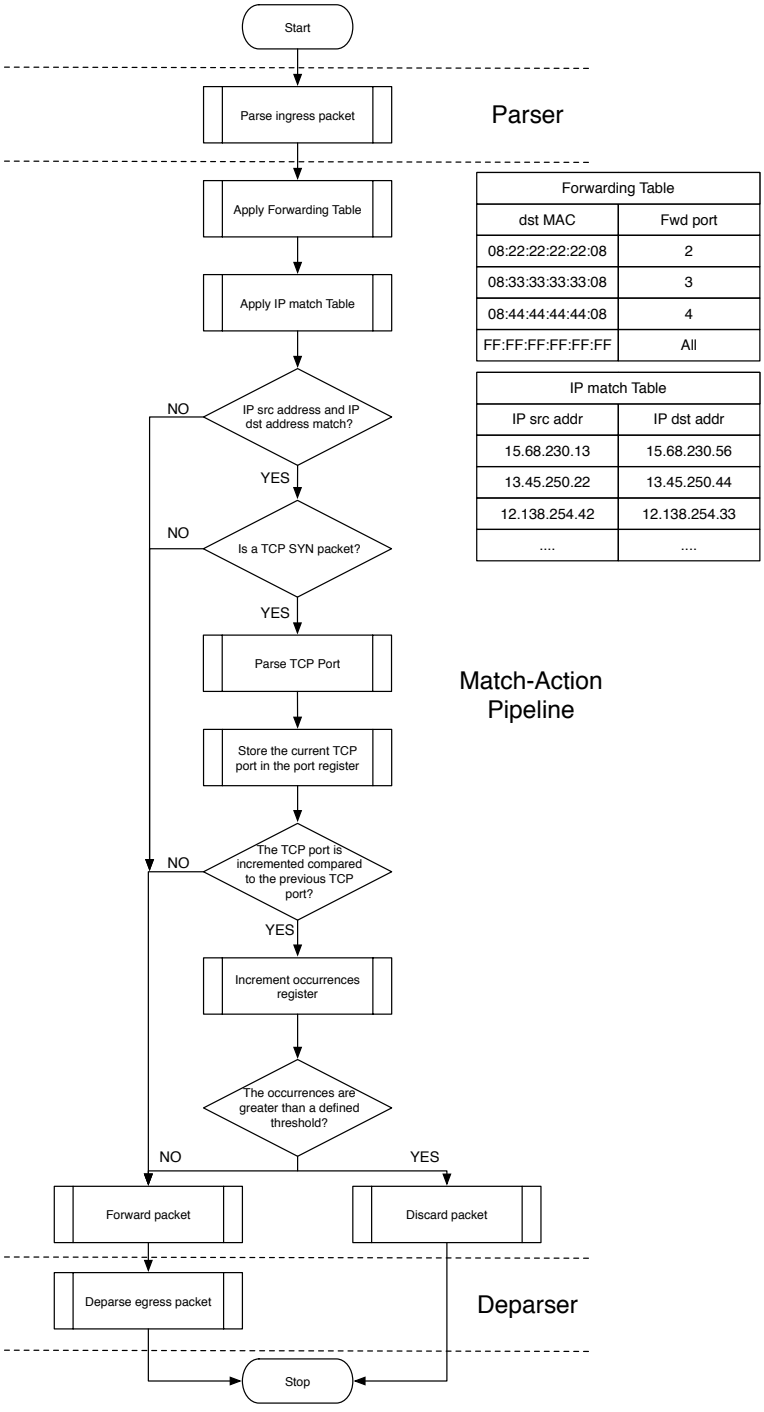


Figure 4.7. P4 program workflow targeting mitigation on TCP SYN flood attacks

each new packet processing. If the number of detected attempts (i.e., the second register) becomes greater than a pre-defined threshold (e.g., 10 attempts), the packet is discarded. Otherwise, registers are updated and the packet is forwarded following the standard P4 egress section.

1.5. Experimental Evaluation. The proposed P4 programs enforcing TE and cyber security have been implemented and experimentally evaluated in a multi-layer network testbed reproducing the network depicted in Figure 4.4. The P4 switch has been evaluated in two different hardware versions, showing the P4 capability to be target-independent:

(1) the reference software switch used in the P4 framework, namely the Behavioral Model version2 (BMV2) implemented in a Linux PC;
(2) the NetFPGA-Sume board, a FPGA dedicated to networking applications.

1.5.1. *Behavioral Model Version 2 soft switch.* In this experimental evaluation section, the edge multi-layer node includes a P4 switch realized with a BMV2 software switch connected to a Reconfigurable Add-Drop Multiplexer (ROADM). In particular the optical bypass is implemented through 100G commercial muxponder, handled by a local agent connected to a optical-layer SDN controller running NETCONF as southbound API [**SIZG+18**]. Five servers (CPU @3.40GHz, 4 GB RAM, Ubuntu 14.04 kernel 4.4.0-31-generic), equipped with multiple 1 and 10 Gigabit Ethernet interfaces, are operated with the BMV2 software switch and configured with the three P4 version 14 programs presented in the previous sections. Traffic is generated and received by two Linux PC servers running Python-based traffic generators and receivers based on the *scapy* library.
Figure 4.8 shows the BMV2 P4 S1 switch behavior (see Figure 4.4) when the P4 program of Sec. 1.3.1 is applied for traffic offload. Traffic is received by the P4 switch interface connected to the generator. In this case the *m_table* flow-entry matching the traffic subject to meter measurement is the source MAC address of the packet generator, thus the meter is applied to all the generated traffic. Note that with the same program it would be possible to meter and tune forwarding for specific traffic flows. When aggregated traffic rate exceeds *TH* (set to a value of 300 packets/s), the second *m_filter* flow table applies the configured P4 rule, correctly identifying for only forwarding towards the alternative port the portion of traffic exceeding *TH*. In particular, the flow entries configured steering traffic

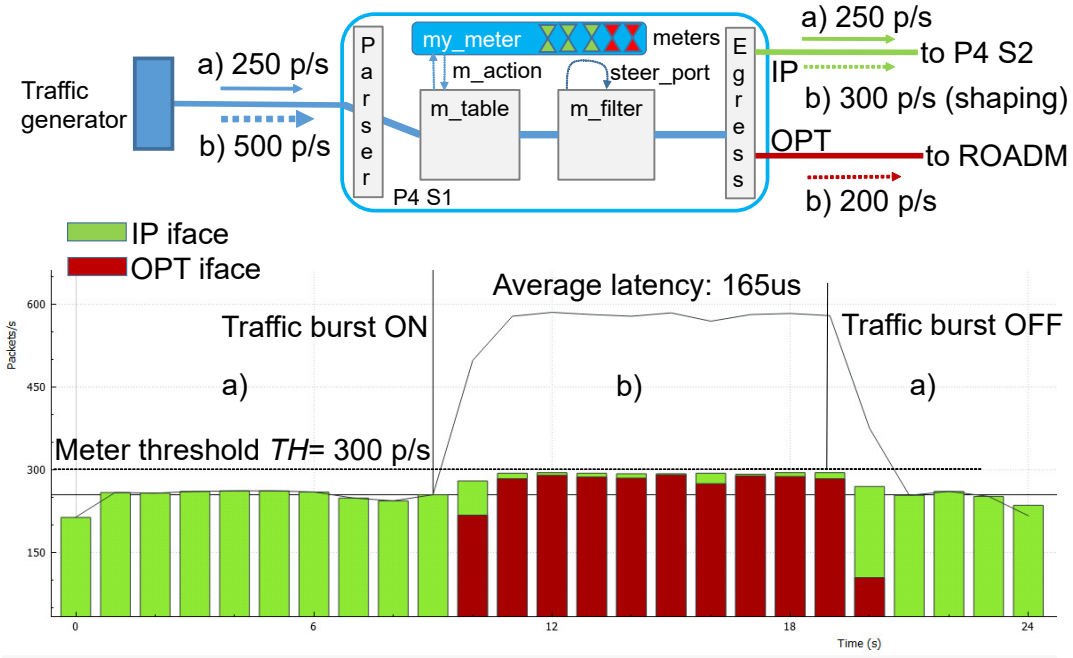


Figure 4.8. BMV2 results: TE traffic offload behavior (packets/s versus experiment time [s]).

to the *IP iface* port (i.e., the port connected to switch S2, in the packet-switched layer) if meter result is set to 0 (i.e., traffic is lower the threshold, see Figure 4.8 case *a*), or steering traffic to the *OPT iface* optical port (i.e., the port connected to the muxponder) if meter is not 0 (see Figure 4.8 case *b*, during the traffic burst injection). Results show that upon a traffic burst of 600 packets/s, the switch applies a shaper on the IP port limiting the output to the threshold, while exceeding traffic is automatically redirected to the optical domain. Such solution allows to keep controlled the profile and the burstiness of the packet switched layer, avoiding possible congestion. With this program, the measured latency of the P4 BMV2 switch is 165us. The meter rate TH does not influence latency results, since the number of operations inside the switch and the amount of instantiated resources is the same for any constant value.

Figure 4.9 shows the same P4 switch S1 behavior when the P4 program of Sec. 1.3.2 is applied for optical bypass. First, a traffic flow at rate below *TH* is considered (see Figure 4.9 case *a*). The first flow table used for matching purposes (i.e., table *m_flowlet*) identifies the metadata timestamp to be stored in the P4 stateful register. The P4 Control condition is not met and traffic is

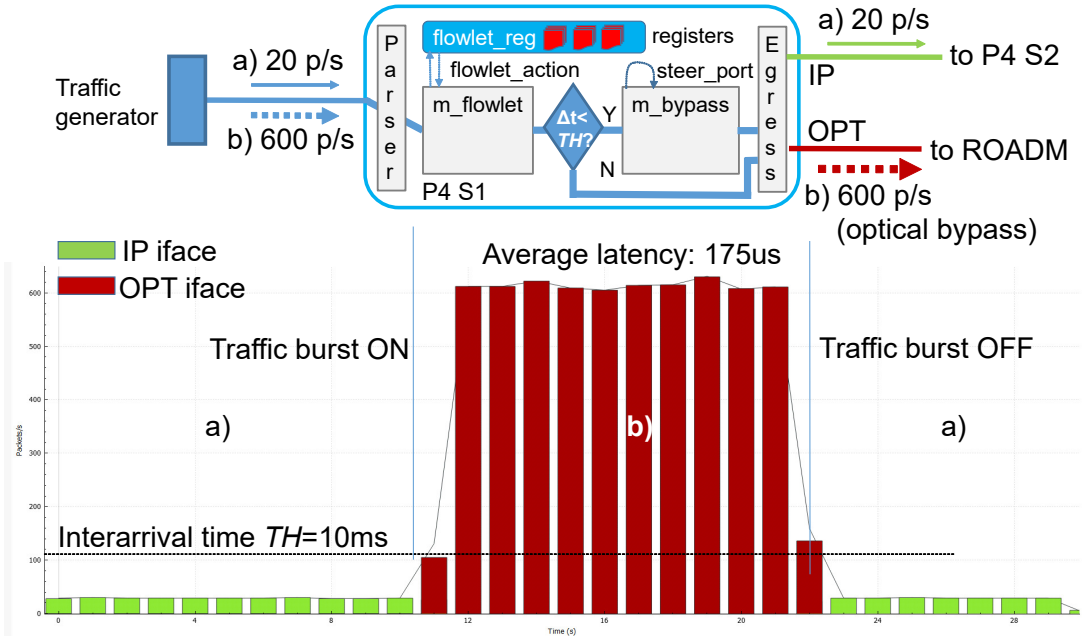


Figure 4.9. BMV2 results: TE optical bypass behavior (packet-s/s versus experiment time [s])

forwarded along the default output port (to S2) towards B. When incoming traffic increases with a traffic burst (see Figure 4.9 case *b*), exceeding *TH*, (i.e., the inter-arrival time decreases below the `FLOWLET_INTERVAL` constant, set to 10ms) the control condition imposes an additional flow table transit (i.e., *m_bypass*), successfully enforcing for packet forwarding the optical bypass, i.e. selecting the optical output port. This means the whole matching traffic is redirected to the optical pipe. As shown in the figure, when traffic burst terminates and rate decreases below *TH*, matching traffic is rerouted again to S2 at the packet layer. Note that, hysteresis-based conditions relying on two thresholds can also be easily implemented to improve network stability. In this case the measured average switching latency is 175us, again with no dependance on the selected TH values. Cyber security P4 program of Sec. 1.4.1 has been implemented over the BMV2 (at switch S5) and evaluated. Figure 4.10 shows the capture collected at switch S5 related to a simple cyber security attack use case. A DDoS block of port scan with incremental Dest TCP Port is implemented (1 packet/s rate). In particular (see P4 program workflow in Figure 4.7), for each matching flow, a *port* register stores the TCP port of the last matched frame, while an additional

No.	Time	MAC src	Source	Destination	Dest TCP port	Protocol
1	0.000000000	aa:aa:aa:aa:aa:aa	10.0.2.15	10.0.1.10	81	FTP-DATA
2	0.000505017	00:01:02:03:04:05	10.0.2.15	10.0.1.10	81	TCP
3	1.003937981	aa:aa:aa:aa:aa:aa	10.0.2.15	10.0.1.10	82	FTP-DATA
4	1.004260026	00:01:02:03:04:05	10.0.2.15	10.0.1.10	82	TCP
5	2.007440645	aa:aa:aa:aa:aa:aa	10.0.2.15	10.0.1.10	83	FTP-DATA
6	2.007774753	00:01:02:03:04:05	10.0.2.15	10.0.1.10	83	TCP
7	3.011349585	aa:aa:aa:aa:aa:aa	10.0.2.15	10.0.1.10	84	FTP-DATA
8	4.014408417	aa:aa:aa:aa:aa:aa	10.0.2.15	10.0.1.10	85	FTP-DATA
9	5.019441735	aa:aa:aa:aa:aa:aa	10.0.2.15	10.0.1.10	86	FTP-DATA
10	6.024572603	aa:aa:aa:aa:aa:aa	10.0.2.15	10.0.1.10	87	FTP-DATA
11	7.029723610	aa:aa:aa:aa:aa:aa	10.0.2.15	10.0.1.10	88	TCP
12	8.034614386	aa:aa:aa:aa:aa:aa	10.0.2.15	10.0.1.10	89	FTP-DATA
13	9.040150296	aa:aa:aa:aa:aa:aa	10.0.2.15	10.0.1.10	90	FTP-DATA
14	10.046014727	aa:aa:aa:aa:aa:aa	10.0.2.15	10.0.1.10	91	FTP-DATA
15	11.051169279	aa:aa:aa:aa:aa:aa	10.0.2.15	10.0.1.10	92	FTP-DATA
16	12.056916813	aa:aa:aa:aa:aa:aa	10.0.2.15	10.0.1.10	93	FTP-DATA
17	13.062595128	aa:aa:aa:aa:aa:aa	10.0.2.15	10.0.1.10	94	FTP-DATA
18	14.067486547	aa:aa:aa:aa:aa:aa	10.0.2.15	10.0.1.10	95	FTP-DATA
19	15.073465562	aa:aa:aa:aa:aa:aa	10.0.2.15	10.0.1.10	96	FTP-DATA
20	16.078981061	aa:aa:aa:aa:aa:aa	10.0.2.15	10.0.1.10	97	FTP-DATA
21	17.083672414	aa:aa:aa:aa:aa:aa	10.0.2.15	10.0.1.10	98	FTP-DATA
22	18.088731432	aa:aa:aa:aa:aa:aa	10.0.2.15	10.0.1.10	99	FTP-DATA
23	19.094381633	aa:aa:aa:aa:aa:aa	10.0.2.15	10.0.1.10	100	FTP-DATA
24	20.099637481	aa:aa:aa:aa:aa:aa	10.0.2.15	10.0.1.10	101	FTP-DATA

Figure 4.10. BMV2 results: Wireshark capture of TCP SYN Flood port scan received and blocked after three attempts by the cyber security P4 program

occurrences register stores the number of consecutive scan condition occurrences. If scan is detected, a threshold of N=3 packets is allowed to be forwarded by the switch while the subsequent ones (Port ¿83) are dropped, successfully blocking the considered cyber-attack.

The cyber security program has been evaluated with more complex attack scenarios. Figure 4.11 shows the scalability analysis as a function of the P4 program size in terms of configured matching and forwarding conditions. Up to 1000 flow entries of table *IP_Match* (matching IP source and destination) have been configured on switch S5. Then, three types of traffic flows are considered. In the *rand* case, the switch is loaded with random traffic with no attacks; the *att* case only includes packets referring to the attack; the *mix* case includes a 50:50 combination of both. Results show that very constant latency performance of around 200us is achieved at the increase of P4 flow table entries, while latency variations (up to 40us) are experienced as a function of the actual internal P4 operations according to traffic conditions. In particular, the attack case requires a longer workflow execution with respect to non-attack scenario, impacting the BMV2 total processing time of a packet in the two different scenarios.

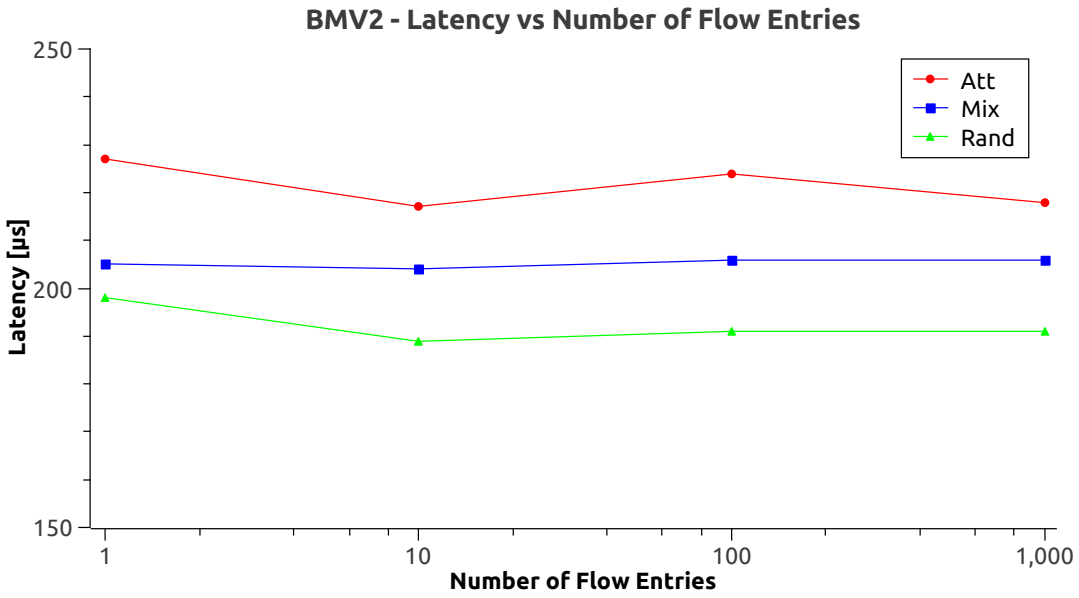


Figure 4.11. BMV2 results: scalability performance of the cyber security P4 program in different attack rate scenarios

1.5.2. *P4-based NetFPGA.* To evaluate the impact of P4 over real programmable hardware devices, the P4 switch implementing cyber security program has been also implemented on a NetFPGA-Sume board [**WSD+17**,**ZACM14**]. The board is based on the Xilinx Virtex 7 FPGA capable of supporting 4x10 Optical 10 Gigabit Ethernet interfaces SFP+ ports. The board is equipped with 8 GB of DDR3-SODIMM RAM and a x8 Gen3 PCIe that allows to control the NetFPGA from an external host. In this case, the board can be plugged as a standard Network Interface Card (NIC) to the PC, with the possibility of reconfiguring the hardware. The NetFPGA hardware is reconfigured by means of the Vivado, SDK Toolkit and Xilinx SDNet software toolkits. P4-based hardware enforcement resorts to proprietary drivers interpreting the JSON files produced by the P4 compiler (p4c version 16). Two NetFPGA SFP+ 10G optical interfaces are connected by means of optical fibers to the Spirent SPT N4U traffic generator and analyzer. The generator is equipped with the MX-10G-S8 card providing up to 8 SFP+-based 10G optical Ethernet interfaces, with traffic profiles obtained by setting different values of the total transmitted throughput. Besides generic TCP traffic profile, specific attack profiles were created emulating TCP SYN Flood attack sequences with configurable percentage of the total throughput.

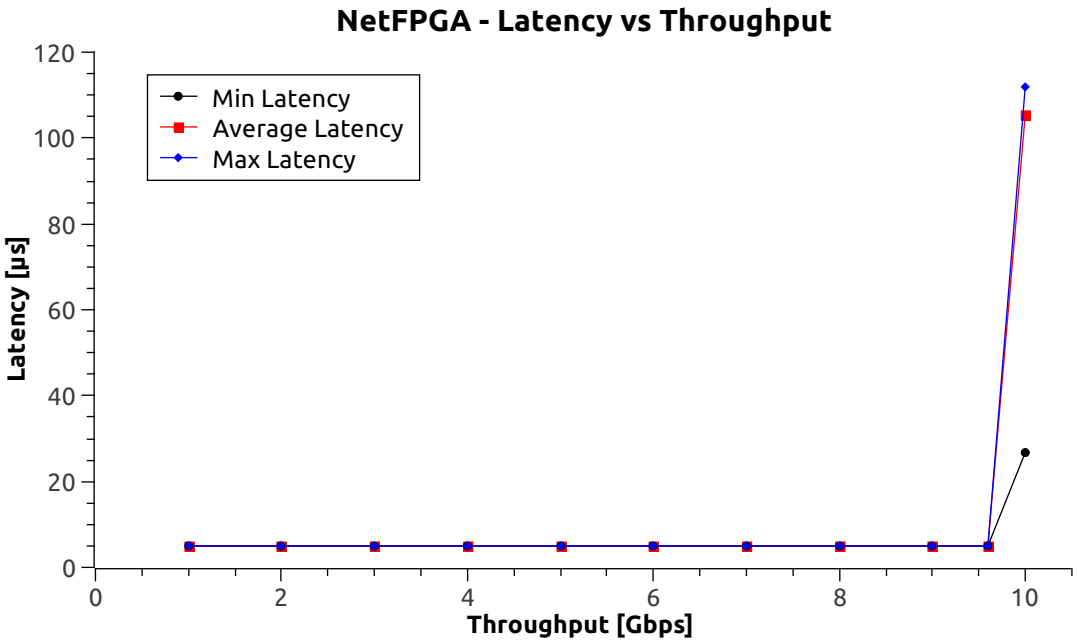


Figure 4.12. NetFPGA results: latency as a function of the traffic throughput

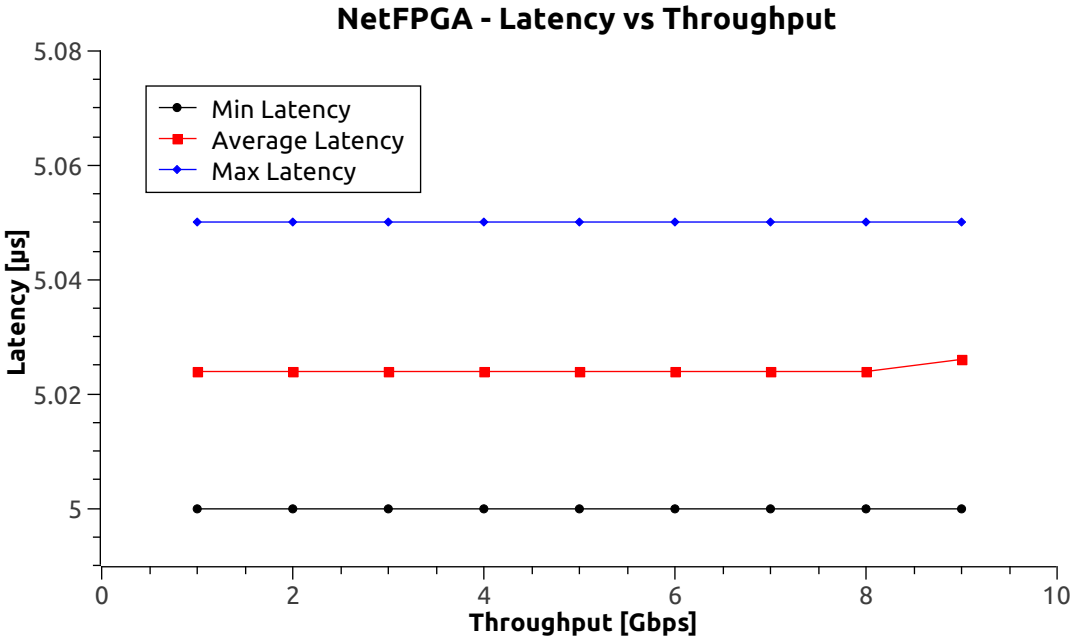


Figure 4.13. NetFPGA results: zoomed version of Figure 4.12 in the 1-9Gbps range.

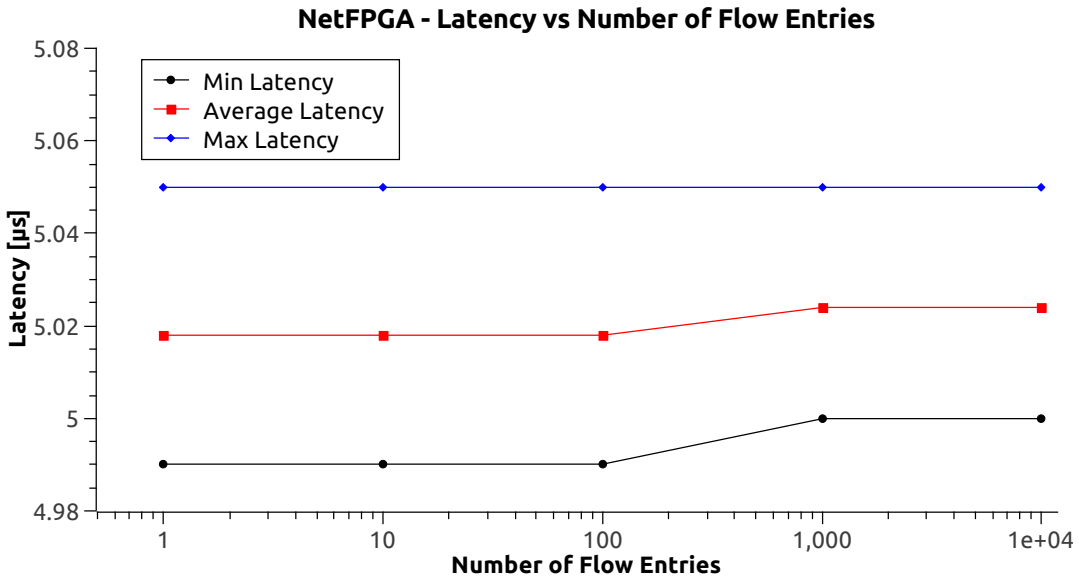


Figure 4.14. NetFPGA results: latency as a function of installed flow entries

Figure 4.12 and its zoomed version (Figure **??**) report the P4-NetFPGA latency in the worst case scenario (i.e., 10 K entries in the IP Match table and 100% attack scenario, in this specific case the discard action has been disabled to measure the latency) as a function of the optical 10 Gigabit Ethernet traffic throughput. Results show an average latency of 5us, practically constant, with constant and very low variance (i.e., min and max latencies differ of around, 50ns) and independent on the traffic throughput in the range 0-9.6Gbps. Latency increases up to 110us in the range 9.6-10Gbps upon quasi full-rate condition when NICs typically introduce significant packet loss. Similar latency (i.e., 4.8us) has been measured on the commercial HP3800 SDN switch at 10Gpbs (exploiting dedicated ASIC), where traffic was simply filtered in a static and stateless configuration, using a standard OpenFlow 1.3 rule in the hardware table matching the TCP port. Thus, this comparison proves that the P4 processing is enforced without introducing significant latency degradation with respect to dedicated ASIC.
Figure 4.14 shows the latency as a function of the monitored IP sessions in the worst case (9Gbps, 100% attack). Excellent scalability performance is provided, achieving a quasi-constant profile of average (5.01us at 10k entries), min and max values. This result, from the point of view of the SDN framework, is particularly noticeable. The reason behind the quasi-constant latency resides in the

Table 4.1. P4-NetFPGA latencies in attack and non-attack scenarios

TCP SYN attack rate (f/s)	TCP non attack rate (f/s)	Throughput (Gbps)	Min latency (µs)	Avg latency (µs)	Max latency (µs)
100	751202	9.134	5.01	5.023	5.05
752188	100	9.146	5	5.024	5.05

Table 4.2. P4-NetFPGA hardware resource utilization

NetFPGA Resource	Reserved by P4 program (%)
LookUp Tables (LUTs) used	23.18
FFss used	16.76
MUX used	0.43
DSPs used	0
RAM blocks used	37.04

usage of fully-associative Content Addressable Memory (CAM) within the NetFPGA [**AL15**]. In fact, tables with flow entries are instantiated as CAM tables. Unlike RAM that receives an input address and returns data word associated to that address, a CAM memory receives a data word in input and searches the entire memory in a parallel fashion to detect at which memory offset the input data word is stored. The NetFPGA implements CAM memory by means of its on-chip Block RAM, enabling massive paralleling search [**WSD$^+$17**]. This means that, given the maximum size of flow entries that may be stored in a table, the time needed to perform a memory look up is kept constant and therefore, the performance of a P4 SDN device over the NetFPGA is practically independent on the number of installed flow entries, thus enabling processing over a large amount of flow sessions.

The impact of attack events on latency is shown in Table 4.1, in which the first (second) row reports results at 9Gbps almost all regular (attack) traffic, respectively. The additional impact of attack attempts results in around 1ns average, blocking all attack profiles. This means that the full P4 workflow processing takes few additional ns with respect to the only parser and deparser sections.

Finally, Table 4.2 reports the NetFPGA hardware resources utilized by P4, consuming around 23% of programmable logic, expressed as LUT rate and 37% of

memory (RAM). Such results show that a single NetFPGA can support more complex P4 programs, e.g. multi-profile or parallel workflows, thus validating the P4 effectiveness for NetFPGA in the cyber security framework.

2. Hardware acceleration for Processing Function Virtualization

As described in Section 2 of Chapter 2, many tasks can be deployed in the NGCO like services prioritization, dynamic service provisioning, security and data processing. Thus, not only Network Functions (NFs) can be deployed at the edge nodes by means of NFV-chain but also data computation can be performed at the edge. For this reason, these services can be called Processing Functions (PFs) and implemented at the edge by means of PFV-chain. In this scenario, the acceleration of the PFs by means of hardware offloading can reduce the processing time and improve the performance. However, unlike NFV service chain orchestrations, the control and the orchestration of such offloaded resources are yet undiscussed, mainly due to the difficulty to expose programmable hardware equipments as a set of independent service chain functions. Indeed, the solutions that have been investigated rely on the hardware acceleration of the single network or processing function. For instance, in [**LTL+16**], the authors propose to offload software logic onto programmable hardware (e.g., FPGA) to accelerate individual network functions. However, many NFVs should be deployed in different edge nodes to have different services addressed in parallel and not always the hardware acceleration can be exploited [**FGC+17**].

In this section, a novel approach based on pipelining multi-service chain is presented to accelerate the PFs. Such a chain is based on many hardware accelerators connected in a FPGA pipeline that works with an aggregated input stream. In particular, the idea is to deploy configurable PFs-chain of many hardware accelerators, deployed into a single board, that independently process input data and in parallel, without any interactions between each other. Such deployed PFs-chain is then controlled by means of a specific framework acting as controller (e.g., SDN) which has the role of supervising the hardware process and interact with it. Host controller flexibility enables specific flows to be subject to a different selected chain, with the objective to minimize the overall experienced latency, thus assuring effective and scalable service differentiation. Such a co-design scenario enables the possibility of using hardware architecture at the edge, improving the

processing of the PFV and maintaining software flexibility. The use case considered to validate the proposed architecture relies on the data processing based on a NF to mitigate a striker traffic at the NGCO edge node as well as an image processing PF to detect a pedestrian from a smart-camera input stream.

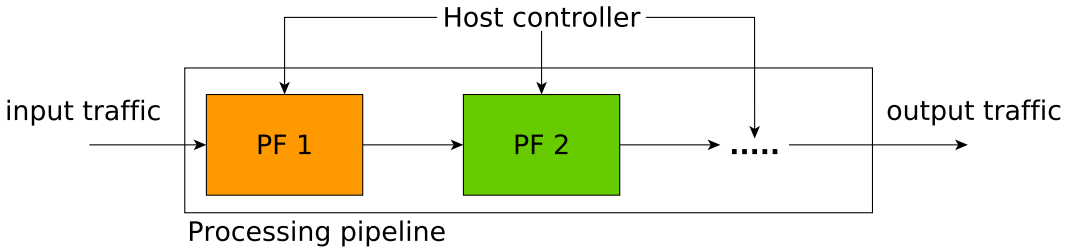


Figure 4.15. PF implemented in the FPGA processing pipeline

2.1. Processing functions chain as FPGA pipeline. A programmable and configurable processing service chain operating as function offload resource within a single hardware board deployed inside an edge node is hereafter proposed. Figure 4.15 shows the PFs implemented in a FPGA processing pipeline considered for the acceleration of the NGCO functionalities. The idea is to deploy several services or processing function accelerators (referred as PF1, PF2, ... PFn) dedicated to different services (e.g., security enhancement, image processing, IoT data analysis, entertainment, etc.) into the single hardware board, so that the input data traffic is processed by the single processing pipeline. Such accelerators can be dynamically enabled or disable (i.e., bypassed) by the controller that also configures the parameters of each acceleration block exploiting the control commands. For example, the software host application may dynamically enable specific traffic flows to be applied to a different PFs of the pipeline. Indeed, the FPGA accelerator can be plugged as a NIC using the PCIe peripheral that allows to control the hardware dataflow and change the parameters from the host control. The processing pipeline can be applied to the aggregated data stream received from an Ethernet optical interface and each stream inside the aggregated traffic can be processed using different acceleration flavouring based on the nature of the input stream. For instance, the traffic coming from cameras are only processed by PF dedicated to the processing without affecting the remaining traffic that passes through the pipeline. The key benefit of using FPGA for acceleration is that they support wide, heterogeneous and unique pipeline

implementations. This feature is in counter-trend with other acceleration platform such as symmetric multiprocessors, DSPs and GPGPUs. Indeed, such kind of devices reaches high level of parallelism by replicating the computation and spreading it among the platform resources. In this scenario, the application can be represented as collection of data elements that can be concurrently processed. However, FPGAs is able to achieve parallelism by duplicating the logic required by the function algorithms, thus many input streams, received from many Ethernet interfaces, can be computed in parallel with a dedicated pipeline. As matter of example, we can consider a FPGA with 4 physical Ethernet interfaces. Each interface receives the traffic and a pipeline is applied to process the input packets. A single pipeline is fully customizable and dedicated to the input stream based on many aggregated dataflows. Furthermore, each pipeline can be further divided in a set of PFs that have different computation functionalities to address the processing of the different dataflows within the input stream. Thus, a pipeline processes different types of traffic and each PF in the pipeline can be enabled/disabled and properly configured by the controller. In this scenario, the latency can be minimized since each data stream has to only pass through the pipeline blocks dedicated to the computation of that stream, without affecting the latency of the others.

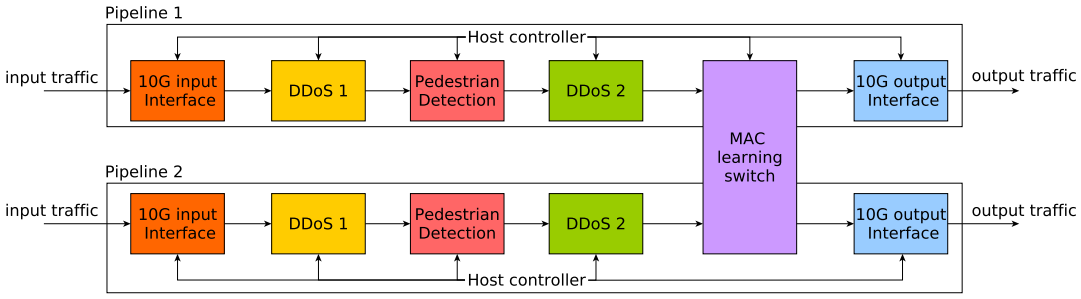


Figure 4.16. Custom processing pipeline to accelerate PFs-chain to mitigate DDoS attacks and detect pedestrian

2.2. Implementation. This section describes an experimental implementation of the configurable hardware-based multi-PFs pipeline. The considered use case targets both computation function (i.e., computer vision) and network function (i.e., cyber security). Image processing applications may require low

processing latency, especially in real-time use cases such as automotive scenarios. Thus, the acceleration of the PFs related to image processing can reduce the computational delay and improve real-time performance. For cyber security, advanced network security functions are required to be demanded to SDN control inspection. However, scalability issues and higher wire speed channels prevent such option: online hardware processing can aim at sustaining the required bandwidth and reducing latency.

The implementation of the considered pipeline is dedicated to mitigate a DDoS attack as well as process data from an input camera for pedestrian detection. Both solutions are deployed using a custom Intellectual Property (IP) core. The board considered for the experimental evaluation is the NetFPGA-Sume, presented in Section 1.5.2 of this Chapter. To evaluate the impact of the pipeline in terms of latency, a FPGA pipeline implementing two DDoS mitigation blocks and a pedestrian detection PFs is considered to demonstrate the flexibility of the hardware acceleration at the NGCO node. In Figure 4.16, the setup is detailed with the PFs-chain implemented in the FPGA. Here, the FPGA acts as a learning switch with two different custom pipelines deployed on two input dataflows. The other two input ports of the NetFPGA serve as output ports to measure the performance of the pipeline. Each processing pipeline is composed by an input block that receives the Ethernet traffic and converts it into an AXI Stream flow. The AXI Stream is the bus considered to connect all entities of the pipeline. Then, two DDoSs and a pedestrian detection PFs are connected to the 10G interface and they respectively have the role to mitigate the traffic and process the input image stream, once enabled by the controller. Moreover, DDoS PFs implement the token-bucket algorithm to mitigate the striker traffic [**TT99**]. This algorithm considers a non-negative counter that increases at a certain rate up to the maximum capacity of the bucket. The counter represents the number of tokens to be added to the bucket. For each input packet, the counter of tokens decreases of one unit. If the packet flow rate is higher than the token counter rate, the bucket will become empty and the packets will be dropped. If the packet flow rate is lower than the token counter rate, the packets will be forwarded. Indeed, the token counter will reach the upper bound of the bucket and the accumulation of the token will be suspended till the arrival of next packet. In this way, the traffic is forwarded if its rate is constant and lower than the token counter rate while bigger data bursts with rate above the threshold are only allowed for short time

slot.
The pedestrian detection PF is based on two main functional blocks: feature extractor and classifier. The extractor produces Histogram of Oriented Gradients (HOG), computed on a dense grid of uniformly spaced cells and normalized. The input image is divided into small connected regions, called cells, and within each cell a histogram of gradient directions is computed. The resulting descriptors are processed by the classifier, based on a pre-trained SVM. The SVM algorithm compares the input vector descriptor with a reference model, which is produced by a supervised learning phase [**MBQ+18**]. The fifth step of the pipeline is the MAC learning switch PF that is common for all the input streams. This PF is responsible for deciding the output port for a packet. After the decision, the packet goes to the output queue waiting for being sent via 10G output PF. The PF is based on Xilinx CAM. Unlike RAM that receives an input address and returns data word associated to that address, a CAM memory receives a data word in input and searches the entire memory in a parallel fashion to detect at which memory offset the input data word is stored. CAM memory is implemented by means of its on-chip Block RAM (BRAM).
Each pipeline stage is controlled by the host exploiting the PCIe interface. Indeed, the system consider a DMA module that exposes an AXI4-LITE bus master interface through which all AXI registers can be accessed from the host (over PCIe). A Linux kernel module permits to access the PCIe, thus the registers of the memory mapped bus are accessed through DMA core. Hence, the controller can handle all the parameters of the different PFs inside the pipeline.

2.3. Results. To evaluate the architecture proposed in the previous Sections, both latency and throughput are considered as performance parameters. To measure the values of such parameters, two NetFPGA SFP+ 10G optical interfaces are connected by means of optical fibers to the Spirent SPT N4U traffic generator and analyzer. The generator is equipped with the MX-10G-S8 card providing up to 8 SFP+-based 10G optical Ethernet interfaces, with traffic profiles obtained by setting different values of the total transmitted throughput. The generator permits to send the packets to the NetFPGA and it can calculate throughput and latency after receiving the packet forwarded by the NetFPGA.
Six different use cases are considered to demonstrate the approach validity, where the parameters of the DDoSs and pedestrian detection pipeline PFs can be changed by means of controller. For each DDoS PF, the parameters taken into

		Latency avg (µs)	Pipeline Input Throughput (Mbps)	Pipeline Output Throughput (Mbps)	Drop	Pipelined Processing Function
Case 1	S1	1.1	574	574.7	NO	DDoS 1
	S2	1.1	6568	6568	NO	–
	S3	47.1	670	670	NO	Pedestrian Detection
Case 2	S1	–	817	0	YES	DDoS 1
	S2	1.1	6535	6535	NO	–
	S3	47.1	670	670	NO	Pedestrian Detection
Case 3	S1	1.1	744	744	NO	DDoS 1 and DDoS 2
	S2	1.1	6550	6550	NO	–
	S3	47.1	670	670	NO	Pedestrian Detection
Case 4	S1	–	817	0	YES	DDoS 1 and DDoS 2
	S2	1.1	6535	6535	NO	–
	S3	47.1	670	670	NO	Pedestrian Detection
Case 5	S1	1.1	744	744	NO	DDoS 1
	S2	1.1	744	744	NO	DDoS 2
	S3	47.1	670	670	NO	Pedestrian Detection
Case 6	S1	–	817	0	YES	DDoS 1
	S2	–	817	0	YES	DDoS 2
	S3	47.1	670	670	NO	Pedestrian Detection

Table 4.3. Latency and throughput evaluation for processing pipeline considering different use cases and input streams

account are the size of the bucket (i.e., number of tokens), the frequency of the token generation, source and destination MAC of the stream on which the block is applied and the enable/disable of the block. For pedestrian detection PF, enabled/disabled parameter and source and destination MAC of the stream are considered. All the cases are based on an aggregated input stream composed by three streams, S1, S2 and S3. For each use case, different throughputs are imposed for S1 and S2 to evaluate the functionality of the DDoS PFs while S3 remains constant at 670 Mbps which corresponds to about 270 frame per second of VGA resolution stream in grey-scale. Table 4.3 shows the latency, imposed and measured throughput for all the use cases to test the pipeline under study. Moreover, the token bucket size and token counter rate are fixed to drop the

traffic with throughput higher than 800 Mbps. The first use case considers the S1 input stream as striker traffic, thus the DDoS 1 is active on the S1 flow, while the S2 stream can be forwarded without any processing. The S1 throughput is under the threshold, thus traffic is forwarded without drop. Case 2 considers the same scenario of Case 1 with S1 throughput higher than the threshold (i.e., 800 Mbps). In this case, the S1 traffic is dropped, while the S2 flow is not affected. Case 3 refers to the scenario where both DDoS 1 and DDoS 2 are applied to the same stream. This case is useful to understand the latency impact of the two PFs enabled on the same traffic. The results show that the latency is equivalent to the one measured in Case 1. Case 4 replicates Case 3 with S1 stream imposed above the 800 Mbps threshold. The system presents the same performance shown in Case 2. Case 5 and Case 6 refer to the DDoS PFs respectively applied to S1 and S2. In particular, DDoS 1 is applied on S1, while DDoS 2 to S2. In Case 5, the two traffics are below the threshold without any drop. In Case 6, the two traffics exceed the threshold of 800 Mbps, thus the Spirent analyzer is not able to receive any traffic. During all the experiments, the pedestrian detection PF is always active on the traffic S3.

The results show the efficiency of the approach in terms of latency. Considering S1 and S2 streams, the average latency measured at the Spirent is the same in all the considered scenarios, also in case of relevant aggregated throughput (i.e., > 7 Gbps). In general, activating two PFs on the same pipeline should increase the latency. The reason of this behaviour lies on the architecture of DDoS PF. Such developed peripheral permits to control the traffic without stalling the traffic itself. Indeed, the DDoS PF listens to the input traffic and calculates the number of tokens without intercepting the traffic and in less clock cycles of those necessaries to forward the traffic. Otherwise, the PF could compute the number of token remained in more AXI stream transactions. In this way, the system will lose packets, especially when the input packet rate is close to 10 Gbps. Hence, the PF does not intercept the traffic and the output is always available for the next pipeline block. When the token number reaches the lower bound of the bucket, the PF will discard the traffic by simply changing the signals of the AXI Stream bus. On the other side, the pedestrian detection PF intercepts the traffic to perform the processing. In this case the output of the pipeline block will be the data computed consuming more clock cycles. Thus, the pipeline latency measured for traffic S3 is 46 µs more than the latency of S1 and S2 due to image

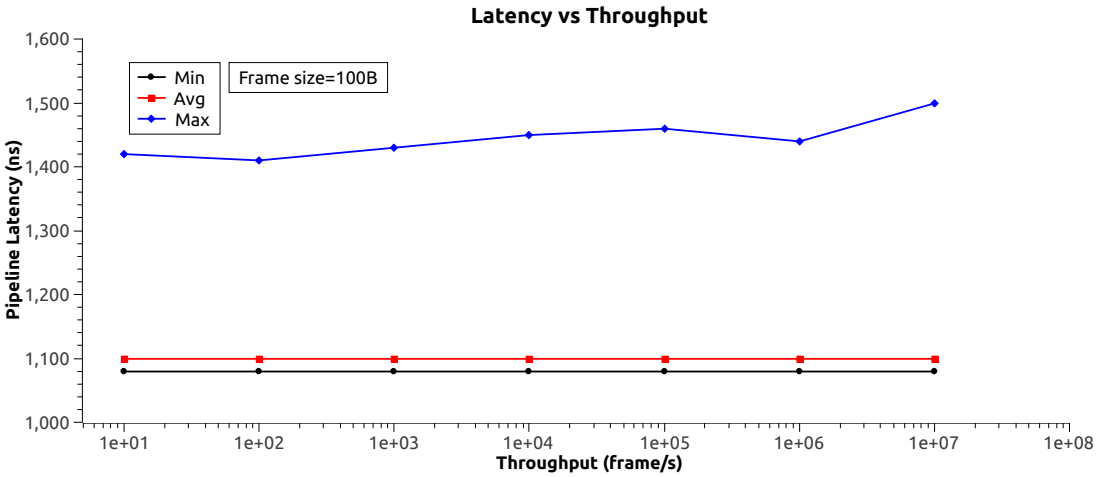


FIGURE 4.17. Pipeline latency as function of the aggregated input throughput

processing computation.

To complete the analysis, Figure 4.17 shows overall pipeline latency, with DDoS pipelined PFs active and pedestrian detection PF disabled, as a function of the aggregated input throughput. The trend is in line with the results shown in Table 4.3 with the average latency constant during all tests up to 10 Mframe/s and packet size of 100 Bytes.

3. Conclusion

In this Chapter, the networking and processing functionalities are deployed at the edge of the datacenter architecture, where the NGCO will host them. Traffic engineering, cyber security and data processing have been considered as functions to be deployed at the edge. Some of these functions are then accelerated by means of reconfigurable hardware to achieve the best performance in terms of latency. In particular, a P4-based architecture of a edge packet-over optical node was presented, along with P4 solutions suitable multi-layer networks, designed and implemented to provide dynamic TE enforcement of optical bypass and traffic offloading. In addition, P4 was also exploited to effectively react against DDoS cyber-attacks requiring stateful capabilities, acting as an active cyber security barrier.

The P4 solutions have been experimentally validated on BMV2 P4 switches and NetFPGA boards, showing impressive scalability performance with the size of the

P4 program and in terms of switch latency to perform P4 operations, especially in the NetFPGA implementation. For example, only 5 µs overall switch latency were experienced running the cyber security P4 code, with no performance degradation using even 10^4 flow entries, thanks to the NetFPGA parallel architecture. Noticeably, results show no significant performance degradation with respect to fixed function commercial switches while gaining a remarkable degree of flexibility and open source programmability. All dynamic TE and cyber security solutions have been successfully implemented within the P4 switch without involving the SDN controller for modifying flow rules during networking operations.

The work demonstrated P4 scalability and flexibility in key multi-layer edge node use cases, thus opening the road to innovative and disruptive open-source traffic forwarding and manipulation procedures to be programmed in the data plane of next-generation converged networks.

Moreover, a novel architecture based on the hardware acceleration of PFs-chain, considering many hardware accelerators connected in a pipeline, has been described. The hardware pipeline has been designed to both mitigate the aggregated input traffic and perform the pedestrian detection at the edge node. A controller (e.g., SDN) can monitor the pipeline and change the parameters of the blocks or enable/disable them. The performance parameters considered for the evaluation are the latency offered by the whole pipeline to forward the traffic and the throughput handled by the FPGA.

Results show the functionalities of the PFs tested with an aggregated stream that contains normal and striker traffic. For DDoS PFs, the latency remains constant since they are designed to not stall the input streams. Moreover, the computation of the DDoS PFs requires less clock cycles than the transaction of the single packet. The pedestrian detection PF intercepts the traffic and requires more clock cycles for data processing. In this case, the latency increases but it only affects the stream dedicated to the image processing, without affecting the other traffic. However, the PF has been designed to minimize the clock cycles necessaries for the computation. Finally, the latency of the pipeline has been also shown without image data traffic as a function of aggregated input throughput and the average pipeline latency remains constant to 1.1 µs in all the considered scenarios.

CHAPTER 5

Conclusions

This thesis discussed the architecture considered for accelerating 5G infrastructure at the edge. Both 5G RAN and 5G access network are investigated as scenarios where the hardware acceleration can be deployed for the next generation of mobile communications. The motivation behind this work is to enhance 5G processing since the requirements proposed by the international standard institutes are stringent. Moreover, 5G envisions to provide connectivity to a large number of users (humans or machines) that require connectivity for different applications. At first, in Chapter 1, an excursus from 1G to 5G has been presented to discuss the innovations and weaknesses of each mobile phone generation. Then, the challenges of the 5G have been introduced to focus on the requirements of the 5G model, deepened in Chapter 2. This Chapter also describes the structure of the overall 5G network considering the 5G RAN and the datacenter access network as scenarios where the acceleration can be placed.
Once the 5G structure has been formally described, in Chapter 3 reconfigurable computing approach is considered to accelerate the functionalities of the 5G protocol stack. The implementation and optimization process to accelerate the OFDM part of the 5G downlink is conducted on a high-end FPGA. The proposed scenarios considers OpenCL as a reconfigurable computing architecture. Implementation fulfills the 5G processing timing requirements since the computation time is consistent with the present 5G deployment. However, to be suitable for 5G, the OpenCL platform must improve the data latency transfer between hardware and software. Moreover, a further enhancement for the proposed OpenCL implementation with respect to optimized software on vectorized high-end processor is to improve the code optimization due to the fact the OpenCL is a general purpose programming language.
Chapter 4 have shown the enhancements that can be reached by means of accelerating network functions at the edge access network (in particular the NGCO). Experimental evaluations have been conducted exploiting reference P4 software

switches and FPGA to accelerate the processing. Extensive results report effective dynamic TE and cyber security mitigation enforcement at P4 switches. Moreover, the Chapter proposed an innovative approach to accelerate network functions at the edge exploiting programmable hardware. A FPGA pipeline for multi-service chain permits to improve the processing close to the end user, enabling dynamic deployment and hardware resources management. Results show the effectiveness of the approach in terms of processing latency.

Concerning the perspectives, many are the future outcomes. At first, this thesis has shown that hardware acceleration model is suitable for 5G acceleration, but it clearly requires further evaluation with real-world deployments. Moreover, this work has considered the software application to control the hardware processing and dynamic offloading. However, the hardware-software interface (i.e., PCIe) does not allow concurrent access to the hardware platform. A scheduling for the hardware access is not yet provided. This extension would be extremely of interest when dealing with virtualization since many virtual instances should be interact with the hardware in parallel. Secondly, this work does not consider the GPGPU processing that should be investigated as acceleration platform that can be an alternative to improve the computation.

To conclude this work, the author would try to give an answer to the question proposed in the title of the thesis. 5G network needs strict latency and bandwidth requirements to meet the demand of an ever-growing number of connected devices from many different services. Thus, the data and network processing must be improved to reach the required performance. In this context, edge computing is a fundamental part that must be considered for 5G communication. Indeed, it permits to only send metadata to the Cloud and to faster react to the final users requests since the processing is physically moved to the network edge. For these reasons, the processing at the edge must be accelerated by hardware or dedicated platforms that ensure the best performance for all the users. Nowadays, the computation for the present mobile phone generation (i.e., 4G) uses ASICs or DSPs to improve the processing, but such architecture are extremely dedicated and they cannot be reprogrammed remotely. Conversely, one of the key performance indicator of 5G is the flexibility. This means that the entities of the network will be virtualized to meet the user demand and saving energy at the same time. Thus, many virtualized software applications can be remotely moved, changed or cancelled and the hardware must follow the software changes.

Reconfigurable computing with reconfigurable hardware applied at the edge can be the key to follow the changes and shape 5G communication.

List of Acronyms

5GC: 5G Core
AI: Artificial Intelligence
ALUTs: Adaptive Look Up Tables
API: Application Programming Interface
ASIC: Application Specific Integrated Circuit
API: Application Programming Interface
AVX2: Advanced Vector Extension 2
BGP: Border Gateway Protocol
CAM: Content Addressable Memory
CD: Compute Device
CLI: Command Line Interface
CNN: Convolutional Neural Network
CDMA: Code Division Multiple Access
CORD: Central Office Re-architected as a Datacenter
CPRI: Common Public Radio Interface
CPU: Central Processing Unit
CU: Central Unit
CvP: Configuration via Protocol
DAC: Digital to Analog Conversion
DDoS: Distributed Denial of Service
DLA: Deep Learning Accelerator
DMA: Direct Memory Access
DSP: Digital Signal Processor
DU: Distributed Unit
DWDM: Dense Wavelength-Division Multiplexing
eCPRI: ethernet-based CPRI
eNB: evolved Node B
EON: Elastic Optical Network

EPC: Evolved Packet Core
FDFIRs: Frequency-Domain Finite Impulsive Response
FDMA: Frequency Division Multiple Access
FFs: Flip Flops
FFT: Fast Fourier Transform
FIR: Finite Impulsive Response
FPGA: Field Programmable Gate Array
gNB: next generation eNB
GPGPU: General Purpose computing on Graphics Processing Unit
HI: Hiperspectral Image
HLS: High-Level Synthesis
HLD: High-Level Design
HOG: Histogram of Oriented Gradients
HDL: Hardware Description Language
iFFT: inverse Fast Fourier Transform
IEEE: Institute of Electrical and Electronics Engineers
IoT: Internet of Things
I/Q: In-phase/Quadrature
KNN: K Nearest Neighbours
LUT: LookUp Table
MIMO: Multiple Input Multiple Output
NF: Network Function
NFV: Network Function Virtualization
NGCO: Next Generation Central Office
NGFI: Next Generation Fronthaul Interface
NG-RAN: Next Generation RAN
NIC: Network Interface Card
NOC: Network Operation Center
NnPU: Neural network Processing Unit
NPU: Network Processing Unit
OAI: OpenAir Interface
OpenCL: Open Computing Language
OpenCL CU: OpenCL Compute Unit
OFDM: Orthogonal Frequency Division Multiplexing
P4: Programming Protocol-independent Packet Processor

PE: Processing Element
PF: Processing Function
PFV: Processing Function Virtualization
PR: Partial Reconfiguration
QoS: Quality of Service
RAM: Random Access Memory
RAN: Radio Access Network
RF: Radio Frequency
RoE: Radio over Ethernet
SDN: Software Defined Networking
SR: Segment Routing
SIMD: Single Instruction Multiple Data
SIMT: Single Instruction Multiple Thread
SoC: System on Chip
SOC: Security Operation Center
SVM: Support Vector Machine
SWI: Single Work-Item
TDFIRs: Time-Domain Finite Impulsive Response
TE: Traffic Engineering
TDMA: Time Division Multiple Access
UE: User Equipment

Publication list

International Journals

Federico Civerchia, Maxime Pelcat, Luca Maggiani, Koteswararao Kondepu, Piero Castoldi and Luca Valcarenghi. Is OpenCL Driven Reconfigurable Hardware Suitable for Virtualising 5G Infrastructure? Major revision submitted to *IEEE Transactions on Network and Service Management*

Francesco Giannone, Koteswararao Kondepu, Himank Gupta, F Civerchia, Piero Castoldi, A Antony Franklin, and Luca Valcarenghi. Impact of virtualization technologies on virtualized ran midhaul latency budget: A quantitative experimental evaluation. *IEEE Communications Letters*, 23(4):604–607, 2019

F Paolucci, F Civerchia, A Sgambelluri, A Giorgetti, F Cugini, and P Castoldi. P4 edge node enabling stateful traffic engineering and cyber security. *Journal of Optical Communications and Networking*, 11(1):A84–A95, 2019

Federico Civerchia, Stefano Bocchino, Claudio Salvadori, Enrico Rossi, Luca Maggiani, and Matteo Petracca. Industrial internet of things monitoring solution for advanced predictive maintenance applications. *Journal of Industrial Information Integration*, 7:4–12, 2017

International Conferences

Federico Civerchia, Maxime Pelcat, Piero Castoldi, and Luca Valcarenghi. Exploiting reconfigurable computing in 5g: a case study of latency critical function. In *2019 IEEE 20th International Conference on High Performance Switching and*

Routing (HPSR). IEEE, 2019

Federico Civerchia, Maxime Pelcat, Piero Castoldi, and Luca Valcarenghi. Exploiting programmable and reconfigurable hardware in 5g. In *2019 IEEE Summer Topicals Meeting Series (SUM)*. IEEE, 2019

F Civerchia, L Kondepu, F Giannone, N Sambo, P Castoldi, and L Valcarenghi. Exploiting pdcp filtering for implementing a capacity-efficient virtual ran recovery. In *2019 21th International Conference on Transparent Optical Networks (ICTON)*. IEEE, 2019

Antonia Napolitano, Gabriele Cecchetti, Francesco Giannone, Anna Lina Ruscelli, Federico Civerchia, Koteswararao Kondepu, Luca Valcarenghi, and Piero Castoldi. Implementation of a mec-based vulnerable road user warning system. In *2019 International Conference of Electrical and Electronic Technologies for Automotive (Automotive 2019)*. AEIT, 2019

Antonia Napolitano, Francesco Giannone, Federico Civerchia, Koteswararao Kondepu, Gabriele Cecchetti, Anna Lina Ruscelli, Luca Valcarenghi, and Piero Castoldi. Italian 5g trials: A vertical view. In *2018 IEEE 4th International Forum on Research and Technology for Society and Industry (RTSI)*, pages 1–5. IEEE, 2018

F Civerchia, L Kondepu, F Giannone, S Doddikrinda, P Castoldi, and L Valcarenghi. Encapsulation techniques and traffic characterisation of an ethernet-based 5g fronthaul. In *2018 20th International Conference on Transparent Optical Networks (ICTON)*, pages 1–5. IEEE, 2018

Federico Civerchia, Enrico Rossi, Luca Maggiani, Stefano Bocchino, Claudio Salvadori, and Matteo Petracca. Lightweight error correction technique in industrial ieee802. 15.4 networks. In *IECON 2016-42nd Annual Conference of the IEEE Industrial Electronics Society*, pages 6044–6048. IEEE, 2016

Bibliography

[19118] IEEE 1914.3. Radio over ethernet encapsulations and mappings. Standard, IEEE, September 2018.

[3GP17] 3GPP. Study on New Radio Access Technology: Radio Access Architecture and Interfaces (Release 14). TR 38.801, March 2017.

[3GP18] 3GPP. 5G, NR, Physical channels and modulation (Release 15). TS 38.211, July 2018.

[ABBS17] Y. Afek, A. Bremler-Barr, and L. Shafir. Network anti-spoofing with sdn data plane. In *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, pages 1–9, May 2017.

[ABC+14] Jeffrey G Andrews, Stefano Buzzi, Wan Choi, Stephen V Hanly, Angel Lozano, Anthony CK Soong, and Jianzhong Charlie Zhang. What will 5g be? *IEEE Journal on selected areas in communications*, 32(6):1065–1082, 2014.

[ADF+13] David Astely, Erik Dahlman, Gabor Fodor, Stefan Parkvall, and Joachim Sachs. Lte release 12 and beyond [accepted from open call]. *IEEE Communications Magazine*, 51(7):154–160, 2013.

[AGE+04] Stefano Avallone, S Guadagno, Donato Emma, Antonio Pescapè, and Giorgio Ventre. D-itg distributed internet traffic generator. In *First International Conference on the Quantitative Evaluation of Systems, 2004. QEST 2004. Proceedings.*, pages 316–317. IEEE, 2004.

[AKL+18] Ijaz Ahmad, Tanesh Kumar, Madhusanka Liyanage, Jude Okwuibe, Mika Yliant-tila, and Andrei Gurtov. Overview of 5g security challenges and solutions. *IEEE Communications Standards Magazine*, 2(1):36–43, 2018.

[AL15] A. M. S. Abdelhadi and G. G. F. Lemieux. Modular SRAM-based binary content-addressable memories. In *2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines*, pages 207–214, May 2015.

[AOC+17] Utku Aydonat, Shane O'Connell, Davor Capalija, Andrew C Ling, and Gordon R Chiu. An opencl™ deep learning accelerator on arria 10. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 55–64. ACM, 2017.

[ASP15] Ali Al-Shabibi and L Peterson. Cord: Central office re-architected as a datacenter. *OpenStack Summit*, pages 1–38, 2015.

[BBCC14] Giuseppe Bianchi, Marco Bonola, Antonio Capone, and Carmelo Cascone. OpenState: Programming platform-independent stateful openflow applications inside the switch. *SIGCOMM Comput. Commun. Rev.*, 44(2):44–51, April 2014.

[BDG+14] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95, July 2014.

[Ben14] Howard Benn. Vision and key features for 5th generation (5g) cellular. *Samsung R&D Institute UK*, 2014.

[BFS+17] Colm Browning, Arman Farhang, Arsalan Saljoghei, Nicola Marchetti, Vidak Vujicic, Linda E Doyle, and Liam P Barry. 5g wireless and wired convergence in a passive optical network using uf-ofdm and gfdm. In *2017 IEEE International Conference on Communications Workshops (ICC Workshops)*, pages 386–392. IEEE, 2017.

[BHM17] W. Braun, J. Hartmann, and M. Menth. Demo: Scalable and reliable software-defined multicast with bier and p4. In *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 905–906, May 2017.

[BMC04] Jeffrey Bannister, Paul Mather, and Sebastian Coope. *Convergence technologies for 3G networks: IP, UMTS, EGPRS and ATM*. John Wiley & Sons, 2004.

[bmv17] Behavioral Model version 2 (BMV2). https://github.com/p4lang/behavioral-model, 2017.

[BMW18] Rory Browne, Paul Mannion, and Eoin Walsh. Cord: Central office re-architected as a datacenter. *White Paper, Intel Corporation*, pages 1–16, 2018.

[C+17] CPRI Consortium et al. ecpri specification v1. 0, 2017.

[CBS+17] Federico Civerchia, Stefano Bocchino, Claudio Salvadori, Enrico Rossi, Luca Maggiani, and Matteo Petracca. Industrial internet of things monitoring solution for advanced predictive maintenance applications. *Journal of Industrial Information Integration*, 7:4–12, 2017.

[CKG+18] F Civerchia, L Kondepu, F Giannone, S Doddikrinda, P Castoldi, and L Valcarenghi. Encapsulation techniques and traffic characterisation of an ethernet-based 5g fronthaul. In *2018 20th International Conference on Transparent Optical Networks (ICTON)*, pages 1–5. IEEE, 2018.

[CKG+19] F Civerchia, L Kondepu, F Giannone, N Sambo, P Castoldi, and L Valcarenghi. Exploiting pdcp filtering for implementing a capacity-efficient virtual ran recovery. In *2019 21th International Conference on Transparent Optical Networks (ICTON)*. IEEE, 2019.

[CLS+08] Shuai Che, Jie Li, Jeremy W Sheaffer, Kevin Skadron, and John Lach. Accelerating compute-intensive applications with gpus and fpgas. In *2008 Symposium on Application Specific Processors*, pages 101–107. IEEE, 2008.

[CPCV19a] Federico Civerchia, Maxime Pelcat, Piero Castoldi, and Luca Valcarenghi. Exploiting programmable and reconfigurable hardware in 5g. In *2019 IEEE Summer Topicals Meeting Series (SUM)*. IEEE, 2019.

[CPCV19b] Federico Civerchia, Maxime Pelcat, Piero Castoldi, and Luca Valcarenghi. Exploiting reconfigurable computing in 5g: a case study of latency critical function. In *2019 IEEE 20th International Conference on High Performance Switching and Routing (HPSR)*. IEEE, 2019.

[CPSC15] C. Cascone, L. Pollini, D. Sanvito, and A. Capone. Traffic management applications for stateful sdn data plane. In *2015 Fourth European Workshop on Software Defined Networks*, pages 85–90, Sept 2015.

[CRM+16] Federico Civerchia, Enrico Rossi, Luca Maggiani, Stefano Bocchino, Claudio Salvadori, and Matteo Petracca. Lightweight error correction technique in industrial ieee802. 15.4 networks. In *IECON 2016-42nd Annual Conference of the IEEE Industrial Electronics Society*, pages 6044–6048. IEEE, 2016.

[CSSK+15] Raúl Chávez-Santiago, Michał Szydełko, Adrian Kliks, Fotis Foukalas, Yoram Haddad, Keith E Nolan, Mark Y Kelly, Moshe T Masonta, and Ilangko Balasingham. 5g: The convergence of wireless communications. *Wireless Personal Communications*, 83(3):1617–1642, 2015.

[CT65] James W Cooley and John W Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, 19(90):297–301, 1965.

[CYRN+16] Chang Chia-Yu, S Ruggero, N Navid, S Thrasyvoulos, and B Christian. Impact of packetization and functional split on c-ran fronthaul performance. In *Proc. of IEEE ICC*, 2016.

[DCA+17] T. Dargahi, A. Caponi, M. Ambrosin, G. Bianchi, and M. Conti. A survey on the security of stateful sdn data planes. *IEEE Communications Surveys Tutorials*, 19(3):1701–1725, thirdquarter 2017.

[DDGF+18] Stefan Dahlfort, Antonio De Gregorio, Giovanni Fiaschi, Shahryar Khan, Jonas Rosenberg, and Tomas Thyni. Enabling intelligent transport in 5G networks. Ericsson technology review, Ericsson, March 2018.

[DGDLR18] David Demmer, Robin Gerzaguet, Jean-Baptiste Doré, and Didier Le Ruyet. Analytical study of 5g nr embb co-existence. In *2018 25th International Conference on Telecommunications (ICT)*, pages 186–190. IEEE, 2018.

[DGE+15] Annika Dochhan, Helmut Griesser, Nicklas Eiselt, Michael H Eiselt, and Jörg-Peter Elbers. Solutions for 80 km dwdm systems. *Journal of Lightwave Technology*, 34(2):491–499, 2015.

[dlOHLA16] Antonio de la Oliva, José Alberto Hernández, David Larrabeiti, and Arturo Azcorra. An overview of the cpri specification and its application to c-ran-based lte scenarios. *IEEE Communications Magazine*, 54(2):152–159, 2016.

[DPE88] Pierre Duhamel, B Piron, and Jacqueline M Etcheto. On computing the inverse dft. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 36(2):285–286, 1988.

[DPS16] Erik Dahlman, Stefan Parkvall, and Johan Skold. *4G, LTE-advanced Pro and the Road to 5G*. Academic Press, 2016.

[DSF+17] R Domingo, Rubén Salvador, Himar Fabelo, Daniel Madroñal, Samuel Ortega, Raquel Lazcano, Eduardo Juárez, G Callicó, and César Sanz. High-level design using intel fpga opencl: A hyperspectral imaging spatial-spectral classifier. In *2017 12th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, pages 1–8. IEEE, 2017.

[EGS+15] Hans-Joachim Einsiedler, Anastasius Gavras, Patrick Sellstedt, Rui Aguiar, Riccardo Trivisonno, and Damien Lavaux. System design for 5g converged networks. In *2015 European Conference on Networks and Communications (EuCNC)*, pages 391–396. IEEE, 2015.

[EUO+18] Opeoluwa Tosin Eluwole, Nsima Udoh, Mike Ojo, Chibuzo Okoro, and Akintayo Johnson Akinyoade. From 1g to 5g, what next? *IAENG International Journal of Computer Science*, 45(3), 2018.

[EVBH08] Jörg Eberspächer, Hans-Jörg Vögel, Christian Bettstetter, and Christian Hartmann. *GSM-architecture, protocols and services*. John Wiley & Sons, 2008.

[Fag14] O Fagbohun. Comparative studies on 3g, 4g and 5g wireless technology. *IOSR Journal of Electronics and Communication Engineering*, 9(3):88–94, 2014.

[FGC+17] Silvia Fichera, Molka Gharbaoui, Piero Castoldi, Barbara Martini, and Antonio Manzalini. On experimenting 5g: Testbed set-up for sdn orchestration across network cloud and iot domains. In *2017 IEEE Conference on Network Softwarization (NetSoft)*, pages 1–6. IEEE, 2017.

[FGG+15] Silvia Fichera, Laura Galluccio, Salvatore C. Grancagnolo, Giacomo Morabito, and Sergio Palazzo. OPERETTA: An openflow-based remedy to mitigate TCP synflood attacks against web servers. *Comput. Netw.*, 92(P1):89–100, December 2015.

[G+17] 5G PPP Architecture Working Group et al. View on 5g architecture (version 2.0). *5G PPP Whitepaper*, 2017.

[GBNM+18] David Gomez-Barquero, David Navratil, Andrew Murphy, Jon Hart, Mael Boutin, Roman Odarchenko, Tuan Tran, Peter Sanders, and Menno Bot. Converged core network. Deliverable 4.2, 5G-PPP 5G-Xcast project, August 2018.

[GKG+19] Francesco Giannone, Koteswararao Kondepu, Himank Gupta, F Civerchia, Piero Castoldi, A Antony Franklin, and Luca Valcarenghi. Impact of virtualization technologies on virtualized ran midhaul latency budget: A quantitative experimental evaluation. *IEEE Communications Letters*, 23(4):604–607, 2019.

[GO11] Dominik Grewe and Michael FP O'Boyle. A static task partitioning approach for heterogeneous systems using opencl. In *International Conference on Compiler Construction*, pages 286–305. Springer, 2011.

[GSP+17a] A. Giorgetti, A. Sgambelluri, F. Paolucci, P. Castoldi, and F. Cugini. First demonstration of SDN-based bit index explicit replication (BIER) multicasting. In *2017*

European Conference on Networks and Communications (EuCNC), pages 1–6, June 2017.

[GSP+17b] A. Giorgetti, A. Sgambelluri, F. Paolucci, F. Cugini, and P. Castoldi. Segment routing for effective recovery and multi-domain traffic engineering. *IEEE/OSA Journal of Optical Communications and Networking*, 9(2):A223–A232, Feb 2017.

[GT] ITUT GSTR-TN5G. Technical report "transport network support of imt-2020/5g", feb 2018.

[Gu18] Dennis Gu. Cloud data centers in the 5G era. Communicate issue 85, Huawei, June 2018.

[HG99] J. Heinanen and R. Guerin. A Two Rate Three Color Marker. *IETF, RFC 2698*, Sept. 1999.

[HH17] J. Hyun and J. W. K. Hong. Knowledge-defined networking using in-band network telemetry. In *2017 19th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pages 54–57, Sept 2017.

[HM15] Lee Howes and Aaftab Munshi. The OpenCL Specification. Version 2.0, Khronos OpenCL Working Group, July 2015.

[HPS+15] Yun Chao Hu, Milan Patel, Dario Sabella, Nurit Sprecher, and Valerie Young. Mobile edge computing—a key technology towards 5g. *ETSI white paper*, 11(11):1–16, 2015.

[HT05] Harri Holma and Antti Toskala. *WCDMA for UMTS.: Radio Access for Third Generation Mobile Communications*. john wiley & sons, 2005.

[HT09] Harri Holma and Antti Toskala. *LTE for UMTS: OFDMA and SC-FDMA based radio access*. John Wiley & Sons, 2009.

[IBMZ19] Stephen Ibanez, Gordon Brebner, Nick McKeown, and Noa Zilberman. The p4→netfpga workflow for line-rate packet processing. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 1–9. ACM, 2019.

[Int17] Intel Corporation. Intel FFT IP Core User guide. Version 17.1, Intel® Corporation, June 2017.

[Kal18] Alexandros Kaloxylos. A survey and an analysis of network slicing in 5g networks. *IEEE Communications Standards Magazine*, 2(1):60–65, 2018.

[Kha09] Farooq Khan. *LTE for 4G mobile broadband: air interface technologies and performance*. Cambridge university press, 2009.

[KLL+17] Y. J. Ku, D. Y. Lin, C. F. Lee, P. J. Hsieh, H. Y. Wei, C. T. Chou, and A. C. Pang. 5g radio access network design with the fog paradigm: Confluence of communications and computing. *IEEE Communications Magazine*, 55(4):46–52, April 2017.

[LJU+17] V. Lopez, J. M. Gran Josa, V. Uceda, F. Slyne, M. Ruffini, R. Vilalta, A. Mayoral, R. Munoz, R. Casellas, and R. Martinez. End-to-end service orchestration from access to backbone. *IEEE/OSA Journal of Optical Communications and Networking*, 9(6):B137–B147, June 2017.

[LTL+16] Bojie Li, Kun Tan, Layong Larry Luo, Yanqing Peng, Renqian Luo, Ningyi Xu, Yongqiang Xiong, Peng Cheng, and Enhong Chen. Clicknp: Highly flexible and high performance network processing with reconfigurable hardware. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 1–14. ACM, 2016.

[LTRa+18] Juho Lee, Erika Tejedor, Karri Ranta-aho, Hu Wang, Kyung-Tak Lee, Eliane Semaan, Eiman Mohyeldin, Juyeon Song, Christian Bergljung, and Sangyeob Jung. Spectrum for 5g: Global status, challenges, and enabling technologies. *IEEE Communications Magazine*, 56(3):12–18, 2018.

[LWN+14] Xiao Lu, Ping Wang, Dusit Niyato, Dong In Kim, and Zhu Han. Wireless networks with rf energy harvesting: A contemporary survey. *IEEE Communications Surveys & Tutorials*, 17(2):757–789, 2014.

[Mar09] Guglielmo Marconi. Wireless telegraphic communication. *Nobel Lecture*, 1909.

[MBQ+18] Luca Maggiani, Cédric Bourrasset, Jean-Charles Quinton, François Berry, and Jocelyn Sérot. Bio-inspired heterogeneous architecture for real-time pedestrian detection applications. *Journal of Real-Time Image Processing*, 14(3):535–548, 2018.

[MBQB18] Patrick Marsch, Ömer Bulakci, Olav Queseth, and Mauro Boldi. *5G system design: architectural and functional considerations and long term research*. John Wiley & Sons, 2018.

[MGMG11] Aaftab Munshi, Benedict Gaster, Timothy G Mattson, and Dan Ginsburg. *OpenCL programming guide*. Pearson Education, 2011.

[MGP+17] F. Morales, L. Gifre, F. Paolucci, M. Ruiz, F. Cugini, P. Castoldi, and L. Velasco. Dynamic core VNT adaptability based on predictive metro-flow traffic models. *IEEE/OSA Journal of Optical Communications and Networking*, 9(12):1202–1211, Dec 2017.

[Moh16] W Mohr. 5g empowering vertical industries. *Tech. Rep.*, 2016.

[MPS+11] G. Meloni, F. Paolucci, N. Sambo, F. Cugini, M. Secondini, L. Gerardi, L. Poti, and P. Castoldi. PCE architecture for flexible WSON enabling dynamic rerouting with modulation format adaptation. In *European Conference on Optical Communication (ECOC)*, Sept. 2011.

[MR04] Jelena Mirkovic and Peter Reiher. A taxonomy of DDoS attack and DDoS defense mechanisms. *SIGCOMM Comput. Commun. Rev.*, 34(2):39–53, April 2004.

[MSR+09] P. Marques, N. Sheth, R. Raszuk, B. Greene, J. Mauch, and D. McPherson. Dissemination of Flow Specification Rules. *IETF, RFC 5575, Network Working Group*, Aug. 2009.

[MW10] Minh Mac and Chris Wysocki. Guaranteeing silicon performance with fpga timing models. White paper, Intel Corporation, August 2010.

[NCG+19] Antonia Napolitano, Gabriele Cecchetti, Francesco Giannone, Anna Lina Ruscelli, Federico Civerchia, Koteswararao Kondepu, Luca Valcarenghi, and Piero Castoldi. Implementation of a mec-based vulnerable road user warning system.

In *2019 International Conference of Electrical and Electronic Technologies for Automotive (Automotive 2019)*. AEIT, 2019.

[NGC+18] Antonia Napolitano, Francesco Giannone, Federico Civerchia, Koteswararao Kondepu, Gabriele Cecchetti, Anna Lina Ruscelli, Luca Valcarenghi, and Piero Castoldi. Italian 5g trials: A vertical view. In *2018 IEEE 4th International Forum on Research and Technology for Society and Industry (RTSI)*, pages 1–5. IEEE, 2018.

[NMM+14] Navid Nikaein, Mahesh K Marina, Saravana Manickam, Alex Dawson, Raymond Knopp, and Christian Bonnet. Openairinterface: A flexible platform for 5g research. *ACM SIGCOMM Computer Communication Review*, 44(5):33–38, 2014.

[NPL+15] Anders Nygren, B Pfaff, B Lantz, B Heller, C Barker, C Beckmann, D Cohn, D Malek, D Talayco, D Erickson, et al. Openflow switch specification version 1.5. 1. *Open Networking Foundation, Tech. Rep.*, 2015.

[OBB+14] Afif Osseiran, Federico Boccardi, Volker Braun, Katsutoshi Kusume, Patrick Marsch, Michal Maternia, Olav Queseth, Malte Schellmann, Hans Schotten, Hidekazu Taoka, et al. Scenarios for 5g mobile and wireless communications: the vision of the metis project. *IEEE communications magazine*, 52(5):26–35, 2014.

[OLAL+17] Jose Ordonez-Lucena, Pablo Ameigeiras, Diego Lopez, Juan J Ramos-Munoz, Javier Lorca, and Jesus Folgueira. Network slicing for 5g with sdn/nfv: Concepts, architectures, and challenges. *IEEE Communications Magazine*, 55(5):80–87, 2017.

[ÖRHT16] M Akif Özkan, Oliver Reiche, Frank Hannig, and Jürgen Teich. Fpga-based accelerator design from a domain-specific language. In *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–9. IEEE, 2016.

[P4] P4 Language Consortium. https://p4.org.

[Pao18] Francesco Paolucci. Network service chaining using segment routing in multi-layer networks. *J. Opt. Commun. Netw.*, 10(6):582–592, Jun 2018.

[PASA+16] Larry Peterson, Ali Al-Shabibi, Tom Anshutz, Scott Baker, Andy Bavier, Saurav Das, Jonathan Hart, Guru Palukar, and William Snow. Central office re-architected as a data center. *IEEE Communications Magazine*, 54(10):96–101, 2016.

[PBMB16] Maxime Pelcat, Cédric Bourrasset, Luca Maggiani, and François Berry. Design productivity of a high level synthesis compiler versus hdl. In *2016 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS)*, pages 140–147. IEEE, 2016.

[PCC18] F. Paolucci, F. Cugini, and P. Castoldi. P4-based Multi-Layer Traffic Engineering Encompassing Cyber Security. In *Optical Fiber Communication Conference (OFC)*, page paper M4A.5. Optical Society of America, 2018.

[PCCC17] F. Paolucci, F. Cugini, G. Cecchetti, and P. Castoldi. Open network database for application-based control in multilayer networks. *Journal of Lightwave Technology*, 35(9):1469–1476, May 2017.

[PCF+15] F. Paolucci, A. Castro, F. Fresi, M. Imran, A. Giorgetti, B. Bhownik, G. Berrettini, G. Meloni, F. Cugini, L. Velasco, L. Potì, and P. Castoldi. Active PCE demonstration performing elastic operations and hitless defragmentation in flexible grid optical networks. *Photonic Network Communications*, 29(1):57–66, Feb. 2015.

[PCK12] Saurabh Patel, Malhar Chauhan, and Kinjal Kapadiya. 5g: Future mobile technology-vision 2020. *International Journal of Computer Applications*, 54(17), 2012.

[PCS+19] F Paolucci, F Civerchia, A Sgambelluri, A Giorgetti, F Cugini, and P Castoldi. P4 edge node enabling stateful traffic engineering and cyber security. *Journal of Optical Communications and Networking*, 11(1):A84–A95, 2019.

[Per17] Sterling Perrin. Evolving to an open c-ran architecture for 5g. *Fujitsu Heavy reading White Paper*, 2017.

[PSCC18] F. Paolucci, A. Sgambelluri, F. Cugini, and P. Castoldi. Network telemetry streaming services in SDN-based disaggregated optical networks. *Journal of Lightwave Technology*, 36(15):3142–3149, 2018.

[R+13] Moray Rumney et al. *LTE and the evolution to 4G wireless: Design and measurement challenges*. John Wiley & Sons, 2013.

[Rec88] Recommendation CCITT. Pulse code modulation (pcm) of voice frequencies. In *Blue Book*. ITU-T, 1988.

[Rod02] Paul Rodríguez. A radix-2 fft algorithm for modern single instruction multiple data (simd) architectures. In *2002 IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 3, pages III–3220. IEEE, 2002.

[RRM+16] Azad Ravanshid, Peter Rost, Diomidis S Michalopoulos, Vinh V Phan, Hajo Bakker, Danish Aziz, Shreya Tayade, Hans D Schotten, Stan Wong, and Oliver Holland. Multi-connectivity functional architectures in 5g. In *2016 IEEE International Conference on Communications Workshops (ICC)*, pages 187–192. IEEE, 2016.

[S+97] Steven W Smith et al. The scientist and engineer's guide to digital signal processing. 1997.

[Sch03] Jochen H Schiller. *Mobile communications*. Pearson education, 2003.

[SGS10] John E Stone, David Gohara, and Guochun Shi. Opencl: A parallel programming standard for heterogeneous computing systems. *Computing in science & engineering*, 12(3):66, 2010.

[SH18] Ahmed Sanaullah and Martin C Herbordt. Fpga hpc using opencl: Case study in 3d fft. In *Proceedings of the 9th International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies*, page 7. ACM, 2018.

[Sil18] W. J. A. Silva. Avoiding inconsistency in OpenFlow stateful applications caused by multiple flow requests. In *2018 International Conference on Computing, Networking and Communications (ICNC)*, pages 548–553, March 2018.

[SIZG+18] A. Sgambelluri, J.-L. Izquierdo-Zaragoza, A. Giorgetti, Ll. Gifre, L. Velasco, F. Paolucci, N. Sambo, F. Fresi, P. Castoldi, A. Chiadò Piat, R. Morro, E. Riccardi, A. D'Errico, and F. Cugini. Fully disaggregated ROADM white box with NETCONF/YANG control, telemetry, and machine learning-based monitoring. In *Optical Fiber Communication Conference*, page Tu3D.12. Optical Society of America, 2018.

[SPC+13] A. Sgambelluri, F. Paolucci, F. Cugini, L. Valcarenghi, and P. Castoldi. Generalized SDN control for access/metro/core integration in the framework of the interface to the routing system (I2RS). In *2013 IEEE Globecom Workshops (GC Wkshps)*, pages 1216–1220, Dec 2013.

[Spe] CPRI Specification. V7. 0,"common public radio interface (cpri); interface specification", 2015. *CPRI Specification*, 6.

[SPG+16] A. Sgambelluri, F. Paolucci, A. Giorgetti, F. Cugini, and P. Castoldi. Experimental demonstration of segment routing. *Lightwave Technology, Journal of*, 34(1):205–212, 2016.

[Sto02] Ivan Stojmenovic. *Handbook of wireless networks and mobile computing*. Wiley Online Library, 2002.

[TS12] Jonathan Tompson and Kristofer Schlachter. An introduction to the opencl programming model. *Person Education*, 49:31, 2012.

[TT99] Puqi Perry Tang and T-YC Tai. Network traffic characterization using token bucket model. In *IEEE INFOCOM'99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No. 99CH36320)*, volume 1, pages 51–62. IEEE, 1999.

[VGIZ+18] L. Velasco, L. Gifre, J. L. Izquierdo-Zaragoza, F. Paolucci, A. P. Vela, A. Sgambelluri, M. Ruiz, and F. Cugini. An architecture to support autonomic slice networking. *Journal of Lightwave Technology*, 36(1):135–141, Jan 2018.

[VK16] P. Voros and A. Kiss. Security middleware programming using P4. *HAS 2016, Human Aspects of Information Security Privacy, Lecture Notes in Computer Science*, 9750:277–287, 2016.

[vLYJ+17] F. van Lingen, M. Yannuzzi, A. Jain, R. Irons-Mclean, O. Lluch, D. Carrera, J. L. Perez, A. Gutierrez, D. Montero, J. Marti, R. Maso, and a. J. P. Rodriguez. The unavoidable convergence of NFV, 5G, and fog: A model-driven approach to bridge cloud and edge. *IEEE Commun. Mag.*, 55(8):28–35, 2017.

[VSC+18] L. Velasco, A. Sgambelluri, R. Casellas, L. Gifre, J. L. Izquierdo-Zaragoza, F. Fresi, F. Paolucci, R. Martinez, and E. Riccardi. Building autonomic optical whitebox-based networks. *Journal of Lightwave Technology*, pages 1–1, 2018.

[Wei91] Mark Weiser. The computer for the 21st century. *Scientific american*, 265(3):94–104, 1991.

[WLH+14] Yi Wang, Jian Li, Lei Huang, Yao Jing, Andreas Georgakopoulos, and Panagiotis Demestichas. 5g mobile: Spectrum broadening to higher-frequency bands to support high data rates. *IEEE Vehicular technology magazine*, 9(3):39–46, 2014.

[WS15] Haomiao Wang and Oliver Sinnen. Fpga based acceleration of fdas module for pulsar search. In *2015 International Conference on Field Programmable Technology (FPT)*, pages 240–243. IEEE, 2015.

[WSD+17] Han Wang, Robert Soulé, Huynh Tu Dang, Ki Suh Lee, Vishal Shrivastav, Nate Foster, and Hakim Weatherspoon. P4fpga: A rapid prototyping framework for p4. In *Proceedings of the Symposium on SDN Research*, pages 122–135. ACM, 2017.

[WTS18] Haomiao Wang, Prabu Thiagaraj, and Oliver Sinnen. Fpga-based acceleration of ft convolution for pulsar search using opencl. *arXiv preprint arXiv:1805.12280*, 2018.

[Yil16] ON Yilmaz. Ultra-reliable and low-latency 5g communication. In *Proceedings of the European Conference on Networks and Communications (EuCNC'16)*, 2016.

[Yu16] Yifan Yu. Mobile edge computing towards 5g: Vision, recent progress, and open challenges. *China Communications*, 13(Supplement2):89–99, 2016.

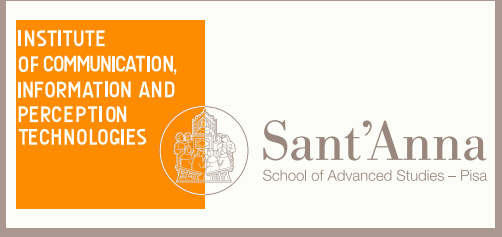
[YWZ+18] H. Yang, Y. Wu, J. Zhang, H. Zheng, Y. Ji, and Y. Lee. BlockONet: Blockchain-based trusted cloud radio over optical fiber network for 5G fronthaul. In *2018 Optical Fiber Communications Conference and Exposition (OFC)*, pages 1–3, March 2018.

[YZZ+15] H. Yang, J. Zhang, Y. Zhao, Y. Ji, J. Han, Y. Lin, and Y. Lee. CSO: cross stratum optimization for optical as a service. *IEEE Communications Magazine*, 53(8):130–139, August 2015.

[Z+15] Y Zhiling et al. White paper of next generation fronthaul interface v1. 0. *China Mobile Research Institute, Tech. Rep*, 2015.

[ZACM14] N. Zilberman, Y. Audzevich, G. A. Covington, and A. W. Moore. Netfpga sume: Toward 100 gbps as research commodity. *IEEE Micro*, 34(5):32–41, Sept 2014.

[ZL17] Jialiang Zhang and Jing Li. Improving the performance of opencl-based fpga accelerator for convolutional neural network. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 25–34. ACM, 2017.


INSTITUTE
OF COMMUNICATION,
INFORMATION AND
PERCEPTION
TECHNOLOGIES
Sant'Anna
School of Advanced Studies – Pisa