Research Executive Agency (REA)

Small cEllS coordinAtion for Multi-tenancy and Edge services

Grant Agreement No.671596

Topic: H2020-2014-ICT-14

Advanced 5G Network Infrastructure for the Future Internet

Research and Innovation Action

Deliverable D4.2

Virtualization Extensions for Acceleration of Light DC capabilities

Document Number:	H2020-5GPPP-GA No.671596/*WP4/D4.2/31.12.2016*
Contractual Date of Delivery:	*31.12.2016*
Editor:	Pavel Bliznakov, Michele Paolino, Sébastien Pinneterre, Daniel Raho - VOSYS
Work-package:	*WP4*
Distribution / Type:	Public (PU) / Report (R)
Version:	*1.0*
Total Number of Pages:	45
File:	*SESAME_Deliverable 4.2_v1.0_Final*

Abstract

SESAME aims to provide a platform for the forthcoming 5G networks, bringing "network intelligence closer to the customer" by consolidating computational power and radio resources at the network edge in the form of Cloud-Enabled Small-Cells (CESC). Micro-servers in the CESC form a cloud, namely Light DC, allowing the execution of various applications spanning from network services to content processing. The Light DC concept leverages technologies such as NFV/SDN and combines them with hardware and software acceleration mechanisms needed to meet performance criteria.

The present deliverable outlines research and implementation efforts resulting in an infrastructure that supports the execution of accelerated VNFs. The outcome of the associated task will be used in the PoC deployment of the SESAME architecture. Additionally, D4.2 motivates the decision to convert the Light DC concept from a cluster of homogeneous micro-servers to a versatile heterogeneous platform.

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	25/11/2016	Initial draft by VOSYS	P. Bliznakov, M. Paolino, S. Pinneterre, D. Raho
0.2	02/12/2016	Contributions by ST	M. Coppola
0.3	02/12/2016	Contributions by ITL	A. Albanese, P. Paglierani, C. Meani
0.4	13/12/2016	Integration by VOSYS	P. Bliznakov, M. Paolino
0.5	15/12/2016	Technical review by ITL	A. Albanese, C. Meani
0.6	21/12/2016	Pre-Final version of the Deliverable, to be reviewed by OTE	M. Paolino
0.7	22/12/2016	Review performed by SMNET	A. Dardamanis
1.0	23/12/2016	Final version of the deliverable, including a fully editorial and conceptual review by OTE	I. Chochliouros

Contributors

First Name	Last Name	Partner	Email
Pavel	Bliznakov	VOSYS	p.bliznakov@virtualopensystems.com
Michele	Paolino	VOSYS	m.paolino@virtualopensystems.com
Sébastien	Pinneterre	VOSYS	s.pinneterre@virtualopensystems.com
Daniel	Raho	VOSYS	s.raho@virtualopensystems.com
Marcello	Coppola	STM	marcello.coppola@st.com
Antonino	Albanese	ITL	antonino.albanese@italtel.com
Pietro	Paglierani	ITL	pietro.paglierani@italtel.com
Claudio	Meani	ITL	claudio.meani@italtel.com
Athanassios	Dardamanis	SMNET	adardamanis@smartnet.gr
Ioannis	Chochliouros	OTE	ichochliouros@oteresearch.gr

Glossary

Acronym	Explanation
ACC	Attribute, Component, Capability
AES	Advanced Encryption Standard
ALU	Arithmetic Logic Unit
API	Application Programming Interface
APP	Application
APPS	Applications
ARM	Advanced RISC Machine
ASIC	Application-Specific Integrated Circuit
ATF	ARM-Trusted-Firmware
AVX	Advanced Vector Extensions
BPI	Byte Peripheral Interface
CAPI	Coherent Accelerator Processor Interface
CESC	Cloud Enabled Small Cell
CESCM	CESC Manager
CFP	C Form-factor Pluggable
CL	Computing Language
CLI	Command-Line interface
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
DC	Data Centre
DCE	Data Circuit-terminating Equipment
DDR3	Double Data Rate type three
DDR4	Double Data Rate fourth generation
DIMM	Dual In-line Memory Module
DMA	Direct Memory Access
DMIPS	Dhrystone MIPS
DOI, doi	Digital Object Identifier
DoW	Description of Work
DP	Data Plane
DPACC	Data Plane Acceleration
DPDK	Data Plane Development Kit
DPI	Deep Packet Inspection
RAM	Dynamic Random-Access Memory
DVFS	Dynamic Voltage and Frequency Scaling
ECC	Error-Correcting Code
eMMC	embedded Multi-Media Controller
ETSI	European Telecommunications Standards Institute
EPC	Evolved Packet Core
EU	European Union
FC	Financial Cryptography
FCBGA	Flip Chip Ball Grid Array
FD-SOI	Fully Depleted Silicon On Insulator
FIFO	First In, First Out
FPGA	Field-Programmable Gate Array
GA	Grant Agreement
Gbe, GbE	Gigabit Ethernet
GPGPU	General-Purpose GPU

GPU	Graphics Processing Units
H2020	Horizon 2020
HDL	Hardware Description Language
HDMI	High-Definition Multimedia Interface
HiPC	High Performance Computing
HPC	High-Performance Computing
HPEC	High Performance Extreme Computing
HW	Hardware
I/O, i/o	Input / Output
IA	Innovation Action
ICT	Information and Communication Technology
ID, Id, id	Identifier
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IFA	Interfaces and Architecture
IOMMU	Input–Output Memory Management Unit
IPS	Instructions per Second
ISPA	International Symposium on Parallel and Distributed Processing with Applications
JIT	Just in Time
JTAG	Joint Test Action Group
KPI	Key Performance Indicator
KVM	Kernel-*based* Virtual Machine
LAN	Local Area Network
LCD	Liquid Crystal Display
Light DC	Light Data Centre
LUT	Lookup Table
MCU	Micro Controller Unit
MEC	Mobile Edge Computing
MII	Media-Independent Interface
MIPS	Million Instructions per Second
MMC	Multi-Media Controller
MMU	Memory Management Unit
MPSoC	Multiprocessor System-on-Chip
µS	micro server
NF	Network Function
NFS	Network File System
NFV	Network Functions Virtualization
NFVI	NFV Infrastructure
NIC	Network Interface Controller
ODL	Open Daylight
ODP	Observer Design Pattern
ODP	Open Data Plane
OF	Open Flow
OpenCL	Open Computing Language
OPNFV	Open Platform for NFV
OVS	Open Virtual Switch
PCI	Peripheral Component Interconnect
PCIe	Peripheral Component Interconnect Express
PF	Physical Function

PMBus	Power Management Bus
PNF	Physical Network Function
PoC	Proof of Concept
PowerPC	Performance Optimization With Enhanced RISC – Performance Computing
PPPJ	Principles and Practice of Programming in Java
Qemu, QEMU	Quick Emulator
QoE	Quality of Experience
QoS	Quality of Service
QSFP	Quad Small Form-factor Pluggable
RAID	Redundant Array of Independent Disks
REA	Research Executive Agency
RFC	Request for Comments
RGMII	Reduced Gigabit Media-Independent Interface
RIA	Research and Innovation Action
RJ	Registered Jack
R/W, r/w	Read/Write
SATA	Serial Advanced Technology Attachment
SC	Small Cell
SD	Secure Digital
SDK	Software Development Kit
SDN	Software Defined Network
SFC	Service Function Chaining
SFP	Small Form-Factor Pluggable
SIGPLAN	Special Interest Group on Programming Languages
SIMD	Single Instruction, Multiple Data
SLC	Single-Level Cell
SM	Streaming Multiprocessor
SMMU	System Memory Management Unit
SMP	Symmetric Multi-Processing
SoC	System on Chip
SOTA	State-of-the-Art
SP	Service Provider
SPI	Serial Peripheral Interface
SPMD	Single Program, Multiple Data
SR-IOV, SRIOV	Single Root Input/Output Virtualization
SSE	Streaming SIMD Extensions
SW	Software
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
VirtManager	VOSYS Hardware Accelerators Virtualization Manager
VIM	Virtualised Infrastructure Management
VF	Virtual Function
VLAN	Virtual Local Area Network
VM	Virtual Machine
VM2VM	Virtual Machine-to-Virtual Machine
VNF	Virtual Network Function
VMM	Virtual Machine Manager
VMM	Virtual Machine Monitor
VS	Virtual Switch
WAN	Wide Area Network

WG	Working Group
Wi-Fi, WiFi	Wireless Fidelity
WP	Work Package

Table of Contents

List of Figures

List of Tables

1. Introduction

One of the main SESAME goals is reducing data processing time and latency within the network. To achieve this, the "mobile edge computing" (MEC) paradigm is leveraged in the form of micro-servers constituting together a distributed datacenter (DC) of low-power, small form factor devices, namely the "Light DC".

The SESAME architecture has been outlined in D2.2 [1]; interactions between components have been specified in D2.3 [2] and the hardware platform of the Light DC micro-server has been defined in D4.1 [3]. The present deliverable aims to bring the focus on the efforts for offloading heavy computational tasks to specialized hardware and software accelerators on the micro-server, in order to increase the overall efficiency of the system by optimizing processing time, network throughput and reducing latency.

This document outlines the results of Task 4.2, which includes investigation of the state-of-the-art (SOTA), architectural concepts, implementation and proof-of-concept (PoC). D4.2 is organized as follows: Section 2 presents the platforms selected for the micro-server prototype as a result of "joint" effort between Tasks 4.1 and 4.2, along with the software that would provide the required functionalities. This section summarizes the work that has been done during SESAME on both hardware and software platforms.

Section 3 details the investigations and development of the virtualization of hardware accelerators in the Light DC. The first subsection gives details upon VOSYS architecture and implementation to enable the use of dynamically programmable hardware accelerators on top of FPGA chips that will be showcased as part of the SESAME PoC. The second subsection brings the focus on the studies carried by ITL on a selected GPU, provided by NVIDIA.

Section 4 focuses on the acceleration of the network in the SESAME micro-server and within the Light DC. It provides information about investigation, design and implementation of software acceleration solutions to enable high performance (high bandwidth, low latency) communication between VNFs.

Finally, Section 5 concludes this deliverable.

2. Specification of the Light DC micro-server

The SESAME project proposes a micro scale virtualized execution infrastructure in the form of a Light Data Centre (Light DC) to enhance the virtualization capabilities of the Small Cell deployment providing high processing power at the network edge. The Light DC, formed by clustering micro servers attached to radio heads, provides a high manageable architecture optimized to reduce power consumption, cabling, space and cost. To achieve these requirements, it relies on an infrastructure that aggregates and enables sharing of computing, networking and storage resources.

Micro servers based on multi core ARMv8[1] 64-bit CPU have been identified as main building block of Light DC, because of their efficiency. These CPUs, usually packaged as System-on-Chip (SoC) solutions, include HW accelerators and interfaces dedicated to networking and packet processing.

However, to further improve efficiency and performance, specific requirements could be fulfilled with standard PCI Express (PCIe) cards[2] equipped with different types of HW accelerators. In case of high capacity storage requirements, the micro server can host a disk controller with the related disks.

The SESAME choice of an ARM-*based* reference platform does not mean the preclusion of hardware resources based on x.86[3] CPUs. In fact, this ARM-*based* platform could be used together with a x.86-*based* system, realizing -in this way- a hybrid architecture where every block aims at maximizing efficiency (performance/watt ratio) of specific sets of workloads. This effectively makes of the SESAME architecture a flexible and generic solution to be used in various scenarios.

The hardware architecture of the Light DC envisages that each micro server will be able to communicate with all others via a dedicated network, guaranteeing the latency and bandwidth requirements needed for sharing resources. Like traditional data centres, the Light DC is a "pool" of resources (computational, storage, network) interconnected by using a communication network. The characteristic aspect of this DC is its decentralised nature, as the whole architecture is built by small, distributed components, i.e. the CESCs. Therefore, the network topology that interconnects the CESCs plays a pivotal role in the architecture, defining the capabilities of the Light DC as "a single unit".

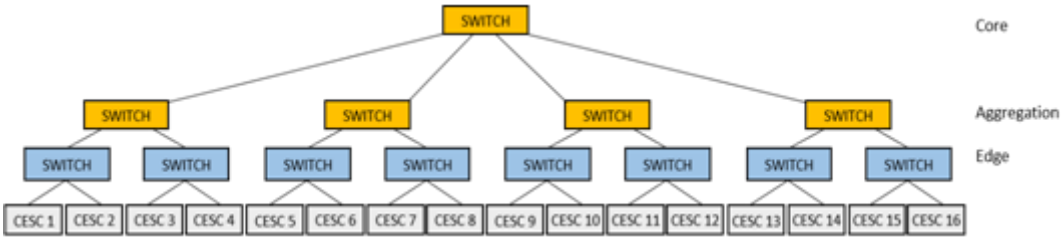

Figure 1: Three-tier topology

As a result of many considerations a three-tier topology has been chosen for the Light DC (Figure 1). In small areas with no geographical barriers, such as an enterprise building or a hospital, this topology might be simplified to a flat tree or even a star topology as shown in Figure 2.

[1] The ARMv8 architecture introduces 64-bit support to the ARM architecture with a focus on power-efficient implementation, while maintaining compatibility with existing 32-bit software. More related information can be found at: *https://www.arm.com/products/processors/armv8-architecture.php*

[2] See, *for example*: *https://en.wikipedia.org/wiki/PCI_Express*

[3] x.86 is a family of backward compatible instruction set architectures based on the Intel 8086 CPU and its Intel 8088 variant. More relevant information can be found, *inter-alia*, at: *https://en.wikipedia.org/wiki/X86*

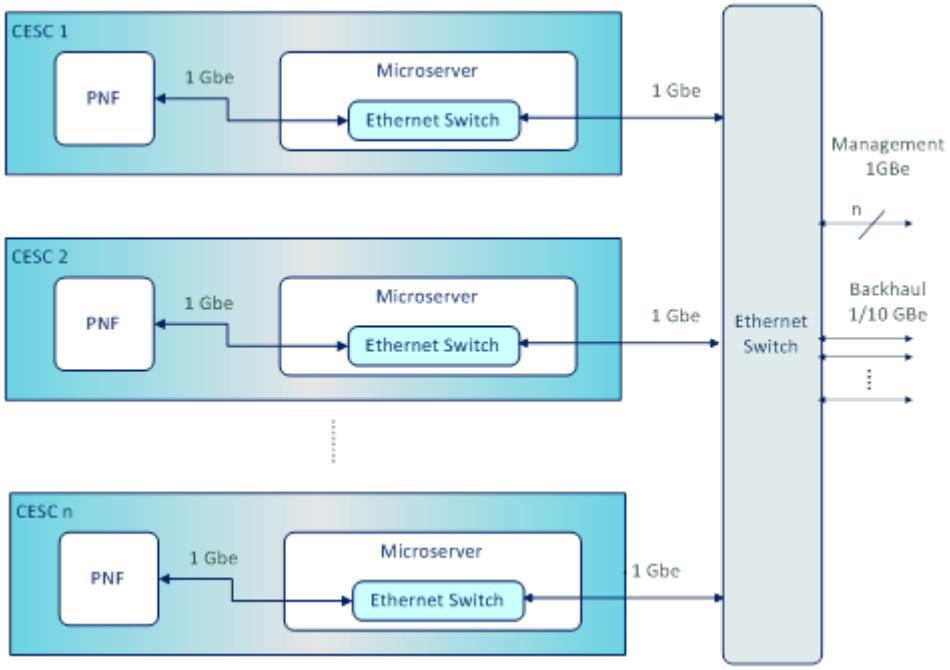

Figure 2: Simplified CESC cluster physical architecture (Source: D4.1)

A further optimization in cost and power of the CESC cluster physical architecture can be achieved leveraging on the internal switch functionality incorporated in the SoCs of the micro servers.
The interfaces that have been identified for the communication between CESCs and external elements and between CESCs that belong to the same cluster are:

- The *CESC fronthaul* - is the interface between the small cell physical network function (SC-PNF) and the micro server.

- The *CESC backhaul* - In the context of SESAME, the backhaul connection is always done through the micro server placed inside each CESC, and refers to CESC-CESC (direct communication between two different CESCs in the cluster), the CESC-EPC communication, as well as the CESC-CESCM communications.

- The *management interface* - It is required for setting up and configuring both individual CESCs and the whole virtual infrastructure. This interface is used to communicate information between SESAME management system (i.e. CESCM) and underlying infrastructure (i.e. Light DC).

2.1. Micro-server hardware architecture

The most distinctive element of the micro server is the host, which, in case of the CESC, is an ARM-*based* SoC including a multicore 64-bit ARM CPU. The ARMv8 architecture introduces 64-bit support to ARM, providing backward compatibility for the 32-bit software. In addition to enhance the performance of ARM processors, the ARMv8 architecture provides low power consumption as well.

The internal general architecture of an ARM-*based* SoC provides a converged platform with:

- A 64bit multi-core ARMv8 processor,
- integrated HW accelerators,
- integrated fabric for I/O communication.

Together with HW accelerators integrated in the SoC, this new server platform is able to provide other powerful HW accelerators (realized on add-in cards) that can be inserted in PCIe Slots to optimize specific

tasks such as Deep Packet Inspection[4] (DPI), video/audio transcoding, graphics, etc. This is another fundamental element to consider, because constrained edge cloud requires acceleration, whenever possible. One of the MEC architecture key performance indicators is the low latency through the proximity of end-users and the serving nodes[5]. The SESAME architecture envisages the deployment of micro servers at the edge cloud leading to achieve very low latency, with clear benefits for enhanced Quality of Experience (QoE) of media flows.

In case of a required local storage (e.g. for video and content caching) one or more micro servers must be able to host a Disk Controller in a PCIe slot, with the related external disks, or one or more SATA disks, directly connected via a SATA links[6]. This feature adds the necessary flexibility for the micro server platform in order to be used with maximum efficiency and flexibility in various scenarios.
The micro server has to provide also the interfaces, internal and external, used in the CESC cluster: (fronthaul, backhaul and management). The following Figure 3 shows the block diagrams and interfaces of the proposed micro server architecture:

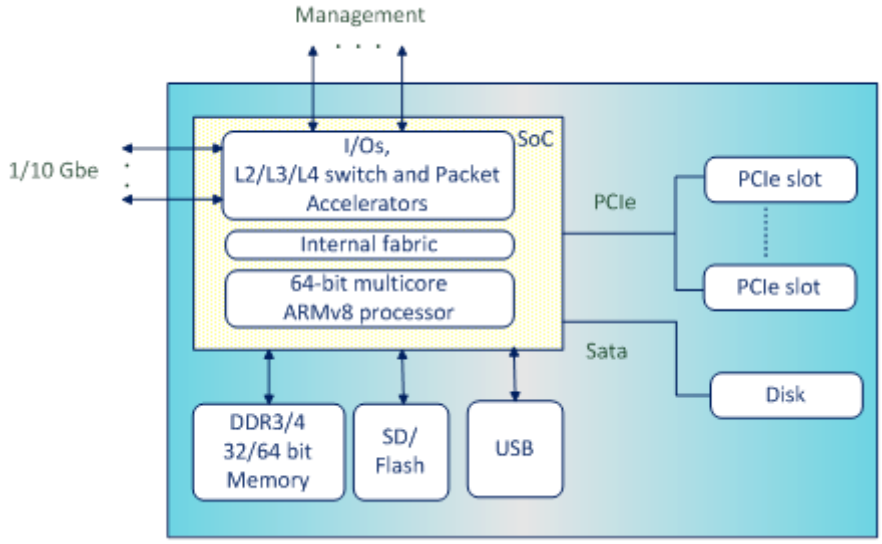

Figure 3: Micro-server architecture

Regarding storage resources, the Light DC HW infrastructure has constraints in order to ensure cost reduction and performance through feature minimisation. Therefore, we can assume that the storage system of the Light DC can be composed by one or two storage devices hosted in the micro server and attached via SATA interfaces.

In the case where only one storage device is attached, reliability of the data stored cannot be guaranteed without other measures. In order to "address" reliability and capacity requirements, the design of the Light DC storage system will be based upon a distributed storage system, addressing the following purposes and goals:

1. Allow storage of cached data including the VNF image at small cell's Light DC site;

2. allow distributed storage of VNF ("application") data;

3. ensure that if data corruption happens on the local storage device, this can be restored through the Light DC storage system.

---

[4] For more information see, for example: *https://en.wikipedia.org/wiki/Deep_packet_inspection*

[5] See, *for example*, the discussion proposed in: C.-Y. Chang, K. Alexandris, N. Nikaein, K. Katsalis, and T. Spyropoulos (2016): MEC Architectural Implications for LTE/LTE-A Networks. In *Proceedings of the Workshop on Mobility in the Evolving Internet Architecture (MobArch'16)*, pp.13-18. ACM, New York, *October 03-07, 2016*.

[6] More related informative details can be found, for example, at: *https://en.wikipedia.org/wiki/Serial_ATA*

2.2. Hardware platforms

One of the main Light DC design goals is to bring computational power at the network edge at low price in a limited space. These considerations were taken into account during the development of the ST Barcelona board, provided by STM[7]. However, for more demanding applications in terms of computing resources, the NXP LS2085A[8] - which has bigger form factor and higher power consumption, but offers high core density and disposes of hardware accelerators in the SoC - has been considered.
In case of high-workload scenarios and if space or power consumption restrictions are present at the CESC, the NXP LS2085A can be deployed as remote, auxiliary compute node with the ST Barcelona being better suitable for supporting the CESC. Without such restrictions the NXP LS2085A can be directly employed as micro-server part of the CESC. Further on, the investigations on some wide-spread hardware accelerators, such as the GPU described in section 3.2 of the present deliverable, lead to the decision to include x86 compute nodes in the Light DC in order to cope with all the possible NFVI scenarios.
Figure 4 shows a possible implementation of a Light DC for the SESAME Proof of Concept (PoC).

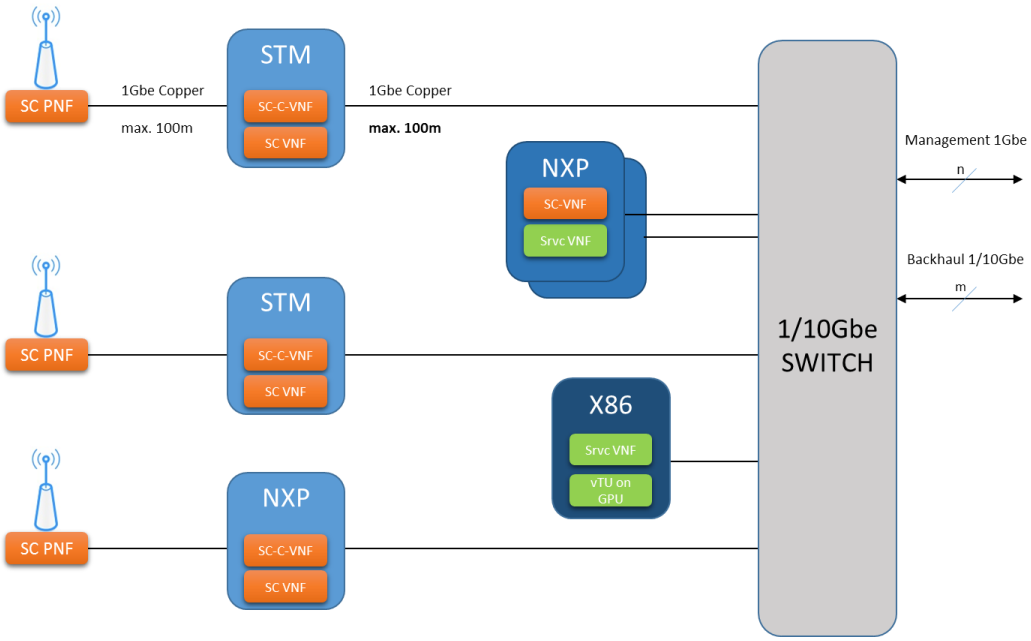

Figure 4: Light DC PoC

There are three Small Cells connected through a 1Gbe Ethernet link with their relative micro servers (two STM and one NXP in the picture) realizing three CESCs. The CESCs are connected to a centralized Ethernet switch, together with other micro servers (NXP and x.86), whose computing/storage resources are ready to be shared among the cluster.

STM and NXP micro servers are equipped with multi-core ARM processor, the x.86 is based on Intel Xeon E5[9] CPU. Following is a brief description of these parts.

2.2.1. ST Barcelona board

The ST SoC is STMicroelectronics' next generation device for low power micro-server for 5G applications. The ST SoC provides a future-proof solution for operators looking for the best performance vs. power trade-off. It embeds hardware-accelerated switching and routing, as well as a powerful multi-core CPU,

[7] See: http://www.st.com/en/evaluation-tools.html
[8] See: http://www.nxp.com/products/microcontrollers-and-processors/arm-processors/qoriq-arm-processors/qoriq-ls2085a-rdb-reference-design-board:LS2085A-RDB
[9] See: http://www.intel.com/content/www/us/en/processors/xeon/xeon-processor-e5-family.html

delivering up to 15 K DMIPS. Using innovative 28 nm FD-SOI silicon technology[10], the ST SoC provides a cost-effective path for operators towards large-scale deployments. The Host processing of the ST SoC includes four ARM® Cortex™-A53 cores[11].

Main features

- Neon SIMD co-processor
- ARM Cortex-A53-MP4[12] up to 1.3 GHz, 1 Mbyte L2 cache[13]
    - 32 Kbyte instruction cache with parity[14]
    - 32 Kbyte data cache with parity
    - Dynamic voltage and frequency scaling[15] (DVFS)
    - Dynamic CPU balancing on system workload
- Supports ARM virtualization and cryptographic extensions
- Supports I/O coherency with high-speed I/O interfaces (PCIe and Ethernet)
- Comprehensive integrated networking solution
    - Hardware-accelerated L3/L4 router and L2 switch, combined with ARM Cortex-A53 processing, up to 16 Gbit/s networking and 4096 accelerated sessions.
    - Rich number of Ethernet interfaces:
        - 3 x RGMII[16] ports (including one LAN/WAN port), 4 x Gigabit Ethernet ports with integrated Ethernet PHYs (including one LAN/WAN port), switching/routing line-rate throughput on every port
    - Wi-Fi bridging for deployment of dual-band concurrent Wi-Fi (2.4 GHz and 5 GHz)
    - Supports I/O coherency with high-speed I/O interfaces (PCIe and Ethernet)
- Flash memory interfaces: SLC[17] NAND, Serial NOR and two eMMCs[18] (one used for SD card[19])
- Secure boot[20]
- Hardware-accelerated cryptographic engine[21], (AES[22], HASH[23]) and security hardening
- Connectivity
    - 2 x PCIe
    - 1 x SATA
- 28 nm FD-SOI silicon technology process for optimum power/performance ratio
- Package: FCBGA[24] 27 x 27

---

10 See: *http://www.st.com/content/st_com/en/about/innovation---technology/FD-SOI.html*
11 For more details see: *https://www.arm.com/products/processors/cortex-a/cortex-a53-processor.php*
12 See the discussion in: *https://www.arm.com/files/event/A2_Advanced_Implementing_ARM_Cortex-A57_and_Cortex-A53_based_big.LITTLE_SoCs_in_16nm_FinFET_Technology.pdf*
13 For more informative details see, *for example*: *https://en.wikipedia.org/wiki/CPU_cache*
14 See: *https://www.manualslib.com/manual/557230/Ibm-Ppc440x5-Cpu-Core.html?page=114*
15 For more details see, *inter-alia*: *https://en.wikipedia.org/wiki/Dynamic_frequency_scaling*
16 For more informative details see, *for example*: *https://en.wikipedia.org/wiki/Media-independent_interface*
17 See: *http://searchsolidstatestorage.techtarget.com/definition/single-level-cell-SLC-flash*
18 For more informative details see: *https://www.datalight.com/solutions/technologies/emmc/what-is-emmc*
19 See: *https://en.wikipedia.org/wiki/Secure_Digital#Microhttps://en.wikipedia.org/wiki/Secure_Digital#Micro*
20 For more informative details see: *https://technet.microsoft.com/en-us/library/hh824987.aspx*
21 See: *https://en.wikipedia.org/wiki/Cryptographic_hash_function*
22 See: *https://vyatkins.wordpress.com/2013/12/06/cryptography-advanced-encryption-standard-aes-and-hash-java-based-examples/*
23 See: *https://en.wikipedia.org/wiki/Hash_function*

After the fabrication of the ST SoC and board implementation, ST has mainly focused to the baseline software development that are required for SESAME. In this context, ST have ported the multistage bootloader[25] and the Linux 64bits.

The bootloader is a piece of code responsible for the Basic hardware initialization and for loading the Linux operating system kernel. Since the SoC includes a CORTEX-A53 processor with ARMv8 architecture, the ARM TRUSTED FIRMWARE[26] (ATF) has been used as the "first stage". Thus, the bootloader implemented by ST (ST-ATF) starts with a valid bootstrap image from the NOR Flash and loads it into RAM. After that, the "second stage" of boot is launched via the U-Boot that is responsible to initialize other hardware devices (network, USB, etc.) and to load the kernel image. Finally, the Linux Kernel[27] takes over the system replacing the bootloaders.
The other activity performed by ST was the initial porting of the Linux Kernel 3.10[28] and then the Linux 64bits v4.1 that involves the support of the ARM A53 CPU core, the coding of the drivers for the ST SoC peripherals and the SoC specific features (SMP[29], power management) and, finally, the coding of the drivers for the board peripherals. The ported kernel at the beginning was mounting a custom file system from ST exported via NFS[30]. Since the ST file system was not handy for SESAME because it is not possible to update and install standard packages using utilities like yum or apt-get[31], ST has decided to support a standard Ubuntu 16.04[32] base file system. To increase portability, ST has decided to store this file system on the SD card present on the board and mount it from there instead from the network. The Ubuntu file system has been configured to "meet" the SESAME requirements and the installation of the most useful packages that have been done. In addition to that, to enable the use of KVM, the bootloader has been modified to increase the security level after boot to EL2[33] (Hypervisor) and the KVM kernel options have been enabled. QEMU[34] has also been installed and by means of that we are capable to run virtual machines.

2.2.2. NXP LS2085A based micro-server

The NXP based micro server is a commercial platform (NXP Freescale LS2085A based) that in SESAME PoC can be used as an alternative -or addition- to STM platform to host Small Cell VNFs and service VNFs.

Figure 1. LS2085ARDB Front View

Main features:

- LS2085A processor (with 8x64-bit up to 1.8 GHz ARM A57 cores[35])
- 16GB DDR4[36] system memory distributed on two 8GB DIMM[37] modules operating at 2.133GT/s (72-bit System Memory)

---

[24] See: https://en.wikichip.org/wiki/Flip_Chip_Ball_Grid_Array
[25] See: https://en.wikipedia.org/wiki/Booting
[26] For more details also see: https://github.com/ARM-software/arm-trusted-firmware
[27] For more relevant information also see: https://www.kernel.org/
[28] See, *for example*: https://kernelnewbies.org/Linux_3.10
[29] For more details see, *for example*: https://en.wikipedia.org/wiki/Symmetric_multiprocessing
[30] See, *for example*: https://en.wikipedia.org/wiki/Network_File_System
[31] See, *for example,* the context within: https://kb.iweb.com/hc/en-us/articles/230241628-Using-yum-or-apt-get-to-install-software-packages
[32] For more details see: http://releases.ubuntu.com/16.04/
[33] For informative purposes also consider: https://github.com/ARM-software/tf-issues/issues/353
[34] For more details see: http://wiki.qemu.org/Main_Page
[35] For more details also see: https://www.arm.com/products/processors/cortex-a/cortex-a57-processor.php
[36] For more details see, *inter-alia*: https://en.wikipedia.org/wiki/DDR4_SDRAM
[37] For more details see, *inter-alia*: https://en.wikipedia.org/wiki/DIMM

- 8GB DDR4 (one DIMM module) for data path connected to one port of 40-bits DDR4 (including ECC[38]) up to 1.67GT/s
- Four RJ45 connectors[39] for 10/1GE support
- Four SFP+ cages[40] for XFI support[41]
- Two PCIe connectors supporting
  - PCIe card (x4/x8 Gen3)
  - PCIe card (x4 Gen3)
- Two SATA 3.0 connectors
- Two USB 3.0 ports
- One SD/MMC card slot
- NOR/NAND flash interface[42]
- 64MB high speed flash with SPI interface[43]

More in detail, Figure 5 depicts the block diagram of the LS2085A processor.

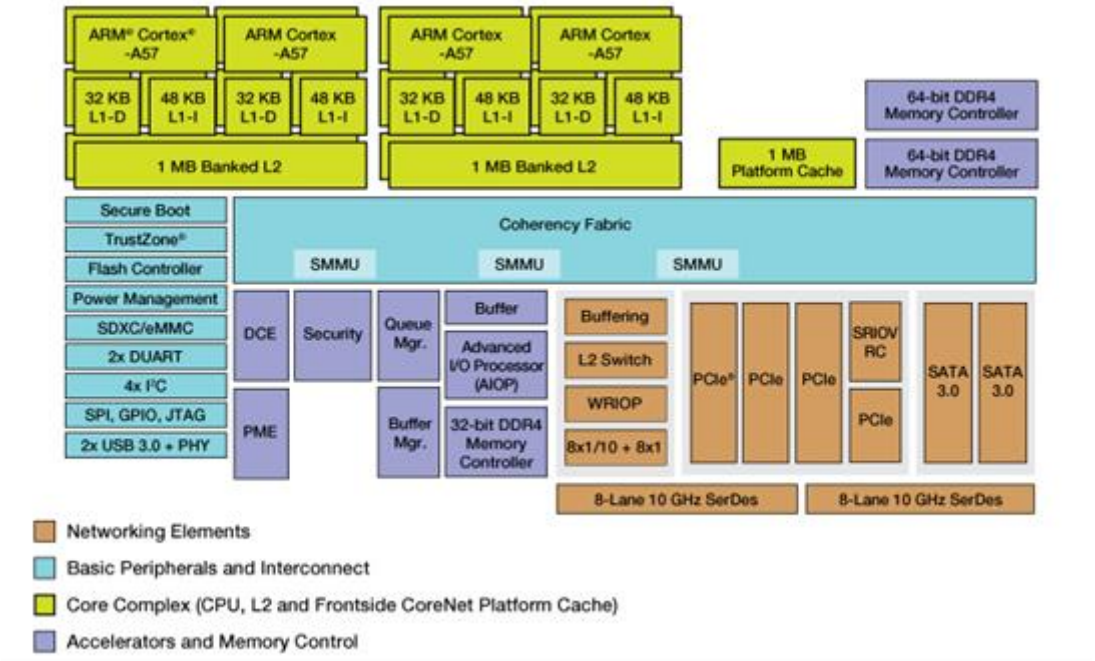

Figure 5: LS2085A block diagram

2.2.3. Intel x.86 based micro-server

The Intel server used in the SESAME PoC to host a HW-accelerated virtual Transcoding Unit (through an NVIDIA QUADRO M4000 GPU) is the GOMA[44] FlexPAC[45] Industrial portable workstation based on Intel Xeon E5-2630v3 CPU.

This server has been chosen for practical reasons and not for low power considerations. Anyway, because of the well-known transparent SW portability between x.86 platforms, this choice does not affect the SESAME PoC. Additionally, considering that Intel is today also focusing on low power computing solutions

[38] For more information also see: *https://en.wikipedia.org/wiki/ECC_memory*
[39] See: *https://en.wikipedia.org/wiki/Registered_jack#RJ45*
[40] See: *https://en.wikipedia.org/wiki/Small_form-factor_pluggable_transceiver*
[41] See: *https://en.wikipedia.org/wiki/E-mu_20K* Also see: *https://en.wikipedia.org/wiki/Sound_Blaster_X-Fi#X-Fi_USB_products*
[42] For more related information also see: *https://en.wikipedia.org/wiki/Flash_memory*
[43] For more related information see, *for example*: *https://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus*
[44] *http://www.gomaelettronica.it/it/*
[45] *http://www.acmeportable.com/products/flexpac*

(e.g., Intel Atom cores), with the possibility of deploying x.86-*based* micro-servers in SESAME, the project architecture is open to future x.86 technological evolutions in that direction.

Main features

FlexPAC Industrial portable workstation configured with:

- Aluminium Chassis including 17.3" LCD Display (Resolution 1920*1080)
- US Keyboard with integrated touchpad
- CPU Intel Xeon E5-2630v3 2.4GHz, 8 Core, 16 Threads
- 64 GB DDR4 RAM (max. 256 GB)
- Intel® C612 chipset[46]
- 2x PCI-E 3.0 x16, (one reserved for GPU)
- 1x PCI-E 3.0 x8,
- 1x PCI-E 3.0 x4,
- 1x PCI-E 2.0 x4 (When CPU 2 is installed)
- 10x SATA3,
- 2x RJ45 GbE LAN ports,
- 4x USB3.0, 2x USB2.0, RAID
- 1 x HDD 4 TeraByte SATA 3.5" in Drive bay
- 3x 3.5 "Removable Drive bay
- DVD R/W

Figure 6a: GOMA FlexPAC portable workstation

2.3. Software platform

The software platform of the micro-server is constructed around the Linux operating system. The software baseline is composed of hypervisor platform, VIM agents, hardware acceleration virtualization layer and accelerated virtual networking as shown in Figure 7.

[46] See: *https://ark.intel.com/products/81759/Intel-C612-Chipset*

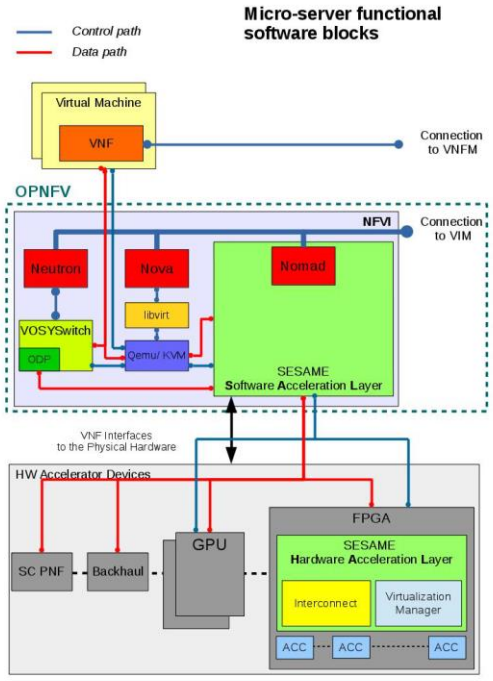

Figure 7b: Micro-server software blocks

2.3.1. Hypervisor platform

The hypervisor platform takes advantage on the virtualization capabilities of the Linux kernel and consists of a Virtual Machine Monitor (VMM), virtualization management tools and a virtual network switch.

The Linux kernel provides a module called Kernel-based Virtual Machine[47] (KVM), providing access to the virtualization extensions of the underlying platform through the special device /dev/kvm. Ioctl[48] calls to this device are used by the VMM to setup guest memory regions, control I/O in the virtual machine, map its display to the host, attach physical devices, etc. Since the main targeted platform by the Light DC is ARM-*based*, the support of ARM virtualization instructions is essential - it has been integrated in the mainstream kernel since version 3.9.

Another element of the hypervisor platform is the VMM. The one selected in SESAME is QEMU which is the most widely used user-space VMM on Linux. Besides providing a frontend to KVM, QEMU also implements mechanisms for live migration of VMs between virtualization hosts.
An essential component of the hypervisor platform is libvirt[49]. It is a collection of API, a daemon and a command line tool, enabling seamless configuration of various virtualization solutions through a unified interface. In fact, it provides the interface and is required for the integration of the micro-server with the SESAME VIM – OpenStack [4].

The management of hardware accelerators virtualization is confined to the Software and Hardware Acceleration Managers, described in details in section 3.1 under the common name VirtManager.
In the scope of Task 4.2, VOSYS developed a user-space virtual switch to provide network connectivity to the VMs, detailed in section 4 of this deliverable. The switch has been successfully ported to the NXP LS2085A platform and its integration has been described in a publically released guide [5].

---

47 For more information see, *for example*: *https://en.wikipedia.org/wiki/Kernel-based_Virtual_Machine*
48 Also see: *https://kernel.org/doc/Documentation/virtual/kvm/api.txt*
49 For more details see: *https://libvirt.org/*

2.3.2. VIM integration

The SESAME VIM of reference is OpenStack, the free and open-source software platform for cloud computing. As a consequence, three agents need to be deployed on each SESAME micro-server.

The first agent is Nova[50], which is responsible of reporting host parameters such as CPU and RAM, and configuring, launching, destroying and migrating virtual machines via the libvirt provided API.

On the other hand, to configure VM networking on virtualization hosts, a Neutron agent[51] adapted to the specific virtual networking solution must be available on the host.

While one of the main advantages of the SESAME Light DC architecture is offloading compute intensive operations to hardware accelerators, OpenStack does not provide a component to manage those accelerators and distribute VM instances according to their availability in the cluster. To overcome this obstacle, some of the efforts in SESAME have been focused towards contributions to the development of a newly introduced component, namely Cyborg[52] (previously Nomad) [9]. Cyborg functionalities and challenges along with FPGA and GPU virtualization are subject of section **Σφάλμα! Το αρχείο προέλευσης της αναφοράς δεν βρέθηκε.** of this deliverable.

[50] Also see: *http://bootrackspacecom.readthedocs.io/en/latest/nova_agent/*
[51] Also see: *https://github.com/openstack/neutron/tree/master/neutron/agent/linux*
[52] Also see: *https://wiki.openstack.org/wiki/Cyborg*

3. Compute acceleration extensions

NFV allows consolidating a great variety of network functions which could previously run only on hardware by specific vendors, as software appliances executed on top of standard commodity servers by any provider. This way, service/network operators can significantly reduce OPEX and CAPEX costs, accelerate time-to-market and achieve the high level of automation and flexibility in service provisioning that the Cloud Computing paradigm can offer.

On the other hand, however, the outburst of innovative services and applications coming from the Internet world is generating an ever-increasing demand for low cost and energy-efficient processing power, which can be hardly satisfied only by the technological evolution of general purpose CPUs.
In fact, while it is generally true that performance can be improved by increasing CPU capabilities, the current technology has power [6] and performance limitations that have led to investigate alternative acceleration technologies.

These limitations are well known by the NFV industry actors, which created the Interfaces and Architecture (NFV IFA) [7] Working Group in *December 2014* to specify the hardware accelerators integration in NFV infrastructures (e.g., lifecycle management, feature discovery and fault tolerance, etc.) through the ETSI NFV IFA 004 document [8]. The open source community is going in the direction of implementing these specifications through OpenStack Nomad [9] (today renamed as Cyborg) proposal under the OPNFV DPACC [10].

In fact, there is today a great interest on the use of hardware accelerators, so as to offload the commodity server general purpose CPU's of the most intensive workloads. To this end, many different types of specialized devices are now available, which can significantly accelerate networking functions, video applications, cryptographic algorithms, audio coding, etc. Among these, general purpose Graphic Processing Units (GPUs) and FPGAs represent a very appealing solution, because of their efficiency and performance.
The SESAME hardware acceleration based on GPU and FPGAs is described in this section. While the former is very well fitting VNFs with high parallelizable algorithms, the latter provides the best performance with low latency/high bandwidth VNFs.

3.1. SESAME FPGA virtualization

An FPGA is an integrated circuit designed to be configured through a hardware description language[53] (HDL), similar to that used for an application-*specific* integrated circuit[54] (ASIC). More in particular, FPGAs contain programmable logic components (called "Logic cells") and a hierarchy of reconfigurable interconnects that allow the blocks to be "wired together" - somewhat like many logic gates that can be inter-wired in different configurations. Logic cells can be configured to perform complex combinational functions, or simple logic gates like AND and XOR. In most FPGAs, the logic cells include memory elements as lookup table[55] (LUT) and D flip-flops[56]. The HDL description of an FPGA block is translated into a bit file (bit-stream), which when downloaded on the device configures the logic blocks to implement the hardware functionality specified in the design.

By using FPGAs, developers can design custom hardware accelerators which, *if compared with a general purpose CPU*, are able to achieve higher performance and lower power consumption. This can be applied to VNFs as well as to any other type of computing, and it is going to radically change the data centres computing paradigm of the next decade, as suggested by recent market moves like the announcement of the Amazon F1 Elastic Cloud [11], Intel's Altera [12] acquisition and emerging products such as the XILINX UltraScale+ MPSoC [13].

---

[53] Also see: *https://en.wikipedia.org/wiki/Hardware_description_language*
[54] For more related information also see: *https://en.wikipedia.org/wiki/Application-specific_integrated_circuit*
[55] For more related information also see: *https://en.wikipedia.org/wiki/Lookup_table*
[56] For more details see, *inter-alia*: *https://en.wikipedia.org/wiki/Flip-flop_(electronics)*

Today FPGAs can be found in the same SoC with a general purpose CPU (as it is the case of UltraScale+ MPSoC[57] and Zynq-7000[58]) or exposed to the processor through a bus such as PCIe, or Coherent Accelerator Processor Interface[59] (CAPI). The former solution is interesting because it provides memory coherency between the FPGA and all the system peripherals (CPU, GPU, etc.). However, the availability of UltraScale+ MPSoC devices is still very scarce at the time of writing of this deliverable. On the other hand, Zynq-7000 SoCs are not considered as a solution, as they not powerful enough and based on processors which are not equipped with hardware virtualization extensions.

For this reason, the SESAME FPGA hardware acceleration developed by VOSYS is targeting bus-exposed FPGAs, and more in particular, PCIe FPGAs. In fact, FPGA PCIe boards today are a well-established technology and are already used for niche (and non-virtualized) networking functions such as data capturing, deep packet inspection, security functions, etc. The ambition of SESAME is to bring this acceleration to the virtual machines in NFV environments.

Thanks to the hardware and software technologies developed by VOSYS during the project, a VM will be able to be connected with multiple accelerators on the same FPGA at the same time. The allocation/deallocation of these accelerators will be done either at boot time, or at runtime. The VIM, the hypervisor as well as the VM itself will be able to perform such operations on FPGA accelerators. Additionally, to enable an efficient FPGA resources utilization, multiple VMs will be able to share the same hardware accelerator.

In the next section the VOSYS VirtManager, which is the key FPGA component designed in SESAME to enable FPGA virtualization, will be detailed. A PoC of this technology will be included in the SESAME final demonstrator.

3.1.1. Hardware accelerators virtualization manager (VirtManager)

The FPGA Virtualization Manager (VirtManager) is a VOSYS patented technology [14] to enable FPGA virtualization, allowing VMs to directly access hardware accelerators. The SESAME VirtManager supports PCIe FPGAs, and more specifically the Xilinx Virtex UltraScale FPGA VCU108 Evaluation Kit[60] (Figure 8), a Xilinx FPGA development board equipped with a Virtex UltraScale XCVU095-2FFVA2104E FPGA, JTAG[61] connector and 4x28Gbps CFP2 & QSFP28 optical interfaces[62].

The PCIe interconnection is used by the SESAME hypervisor to interact with the VirtManager, which partitions the FPGA and enables VMs to share accelerators.

[57] For more details see: *https://en.wikipedia.org/wiki/MPSoC*
[58] Also see: *https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html*
[59] Also see: *https://en.wikipedia.org/wiki/Coherent_Accelerator_Processor_Interface*
[60] For more details see: *https://www.xilinx.com/products/boards-and-kits/ek-u1-vcu108-g.html*
[61] See: *https://en.wikipedia.org/wiki/JTAG*
[62] See, *for example*: *http://www.completeconnect.co.uk/cfp-qsfp28-100g-singlemode-transceivers/*

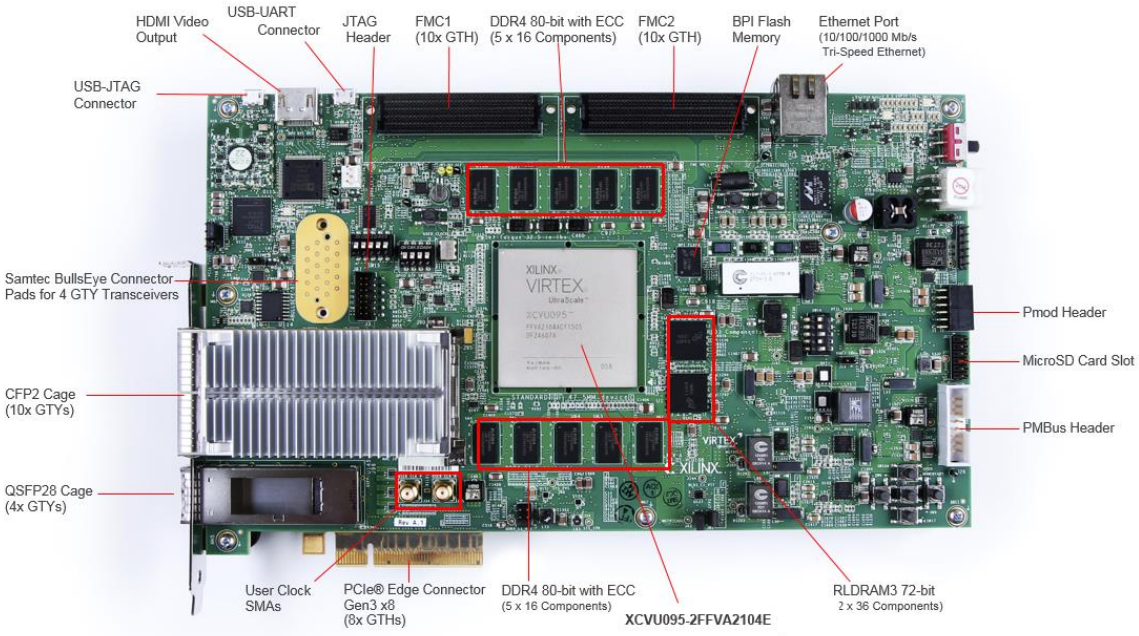

Figure 8: Xilinx Virtex UltraScale FPGA VCU108 Evaluation Kit

3.1.1.1 VirtManager architecture

As shown in Figure 9, the FPGA VirtManager is composed by both hardware and software components.

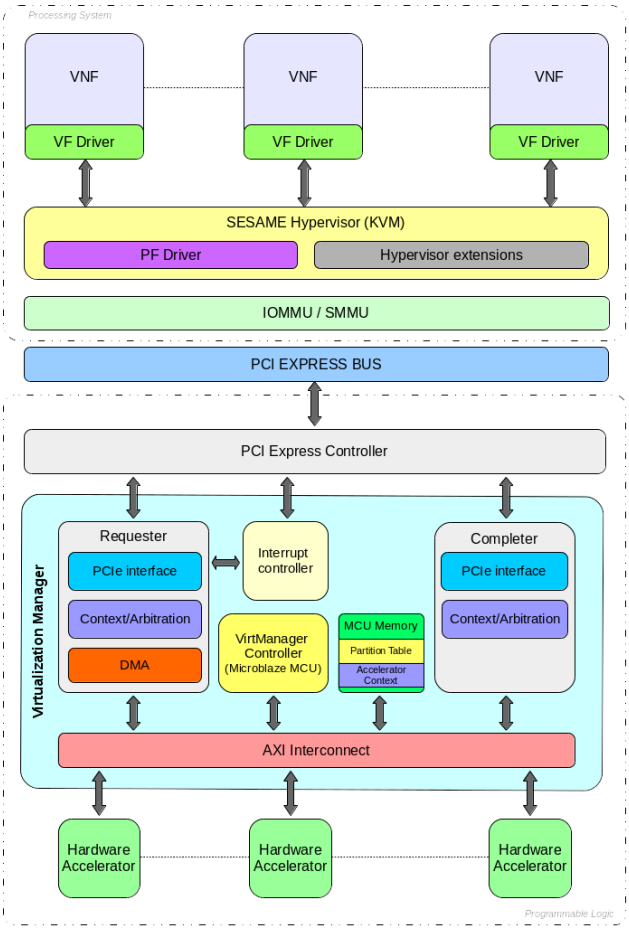

Figure 9: VirtManager architecture

From the software point of view, a set of guest and host drivers expose the accelerators capabilities to the VNFs. Additionally, hypervisor extensions (QEMU, libvirt, etc.) which enable the VIM to "instantiate" accelerated VMs are also considered.

As for the hardware side, a Single Root I/O Virtualization (SR-IOV) [15] compliant PCI controller enables the interaction between the general purpose processor and the accelerators. The VirtManager uses specific PCI-Express interface components (Completer and Requester blocks) to link VMs to accelerators, thus allowing each VNF to be directly interfaced to hardware accelerators in an isolated manner. The binding between accelerators and VMs is established at runtime by the context manager, which enables accelerator isolation through a set of banked registers used to save/restore the guest contexts.

These operations are orchestrated by a Micro Controller Unit[63] (MCU) which implements the hypervisor communication control plane. The data plane, on the other hand, is managed by a specific DMA[64] engine in the FPGA.

In the following part of this section, the aforementioned hardware components are described in details.

3.1.1.1.2 PCI-Express Controller

The VirtManager PCI-Express Controller is based on SR-IOV, an extension of the PCI Express specification[65] (PCI-SIG). SR-IOV allows a device such as the SESAME FPGA board to "partition" its resources in different

[63] For more related information see, *for example*: *https://en.wikipedia.org/wiki/Microcontroller*
[64] For more related information see, *for example*: *https://en.wikipedia.org/wiki/Direct_memory_access*
[65] Also see: *https://pcisig.com/*

PCI-Express hardware functions, which can be of two types: Physical and Virtual Functions (PF and VF). The former is the primary interface which is used by the SESAME hypervisor to interact with the VirtManager. The PF configures and manages VFs, which are lightweight PCI-Express functions that are used as accelerator interfaces. Conversely from the PF, VFs are mainly focused to move data to/from the accelerators, and have a minimal set of configuration resources.

3.1.1.1.3 Completer and requester modules

The PCI-Express controller interacts directly with the completer and requester modules inside the VirtManager.

The completer engine module, in particular, manages the memory transactions received from the PCI-Express through the AXI4-Stream protocol[66]. Therefore, this interface is used each time that the hypervisor or the VMs want to send a request to the VirtManager or to the accelerators. The completer consists of two separate interfaces used to send memory-read or memory-write requests to the VirtManager components and to the accelerators. The completer is mainly involved in configuration operations, because to move packets and data, faster and more efficient mechanisms can be used (i.e., the DMA in the requester engine). In fact, the DMA engine performs the data transfers between accelerators and the requester interfaces by using multiple channels connected with SR-IOV VFs to "isolate" data transfers between accelerators and improve performance.

On the other hand, the requester engine component enables an accelerator to initiate PCI transactions as a bus master across the PCI-Express link to the host memory. Similarly to the completer, also the requester interface with the PCI-Express interface controller is based on the AXI4-Stream protocol. The transactions on this component are similar to those on the completer engine, except that the accelerator and PCI-Express interface controller roles are reversed.

Additionally, both completer and requester are equipped with a context/arbitration module, which routes requests to the right accelerator when multiple VMs are sharing it.

For example, when the VMs or the hypervisor require copying data in the accelerator from the VNF, it sends a DMA descriptor in the VirtManager by sending a memory-write request through the completer interface. The destination of this message is read and routed by the context/arbitration module, which delivers the data to the right accelerator. The accelerator is at this point able to configure one of the DMA channels to start the data transfer.

3.1.1.1.4 VirtManager Controller (Microblaze Micro Controller Unit)

The VirtManager Controller is a Micro Controller Unit (MCU), implemented as a Xilinx MicroBlaze[67] soft core processor in the FPGA, which acts as a scheduler and manages the configuration of the hardware accelerators. It interacts with a memory, which contains the accelerators contexts needed to perform the VM context switch for the accelerators, as well as a Partition Table[68] which contains information related to the FPGA partitions: Partition ID, Accelerator type, VM ID, VM configuration and accelerator status registers. The MCU memory is implemented in the internal memory of the FPGA device and is accessible through the VirtManager Interconnect.

The VirtManager MCU is able to decode and execute commands arriving from the VNFs or the hypervisor. Examples of commands are attach/detach partition, read partition table, etc.

[66] For more details see: *https://www.xilinx.com/products/intellectual-property/axi4-stream_interconnect.html*
[67] For more details also see: *https://www.xilinx.com/products/intellectual-property/microblazecore.html*
[68] For more related information see: *https://en.wikipedia.org/wiki/Partition_table*

3.2. SESAME GPU virtualization

This subsection presents an overview of the investigations lead by ITL on GPU in the scope of Task 4.2. In the beginning, the GPU hardware architecture is detailed. The second part of the subsection "outlines" the available programming languages and tools for interacting with GPUs. In the third part, different interfacing mechanisms between the GPU and a VM are presented. Finally, the focus is taken to the integration of the GPU in the micro-server.

General-purpose Graphic Processing Units (GPUs) represent a very appealing solution, owing to their high computation performance (one or two order of magnitude faster than a general purpose CPU) and relatively low cost, guaranteed by the huge demand of such devices originated by the gaming market [16].

With respect to the extremely complex and closed-source graphic processors available a decade ago, current GPUs can be programmed with general purpose, high level languages, such as CUDA (by NVIDIA[69]) or OpenCL[70], coming from the open source community[71]. Moreover, effective development and debugging/profiling tools are also available, and are continuously enhanced by device manufacturers, so as to facilitate the rapidly increasing community of GPU developers [17]-[19].

However, to fully exploit the huge potentialities offered by GPUs to the cloud computing context, it is necessary to "extend" virtualization also to this type of processors. Fortunately, many activities have been ongoing in this field in the last years [20]-[22].

The first attempts to virtualize GPUs have been made in the areas of scientific high performance computing and education, in order to share highly expensive GPU clusters, often under-utilized, among a larger number of users. The next step was moving GPUs to the cloud within the "Infrastructure as a Service" framework.

3.2.1. GPU computing overview

Driven by the insatiable demand for real-time, high-definition 3D graphics, the programmable Graphic Processor Unit or GPU has evolved into a highly parallel, multithreaded, multicore processor, with extraordinary computational power and very high memory bandwidth compared to standard CPUs.

The reason behind the discrepancy in floating-point capability between the CPU and the GPU[72] is that the GPU is specialized for compute-intensive, highly parallel computation - exactly what graphics rendering is about - and therefore designed such that more transistors are devoted to data processing rather than data caching and flow control, as in general-purpose CPUs, which are optimized for performance in sequential processing.

[69] See: *http://www.nvidia.com/content/global/global.php*
[70] For more informative details see, *for example*: *https://en.wikipedia.org/wiki/OpenCL*
[71] Also see: *https://opensource.com/tags/community*
[72] Also see: *http://www.nvidia.com/object/what-is-gpu-computing.html*

Figure 10: Schematic chip subdivision into different resources (CPU vs. GPU) In a GPU, much more circuitry is devoted to computation (source: NVIDIA).

For this reason, GPUs are now widely used as "GPGPU" (General-Purpose GPU), providing high-performance computing power that can be used to accelerate a wide range of HPC applications.

The general architecture of a modern GPU is shown in Figure 10. It is organized into an array of streaming multiprocessors (SMs), in which all processors can work in parallel, as long as they execute the same instruction, each on its own data (following a SIMD – Single Instruction, Multiple Data[73], or SIMT – Single Instruction, Multiple Thread[74] computation model). Each core within a SM is equipped with its own registers and local memory space; there is furthermore a fast cache memory shared by all cores of the SM (for shared data) and a global memory, common to all SMs contained in the GPU.

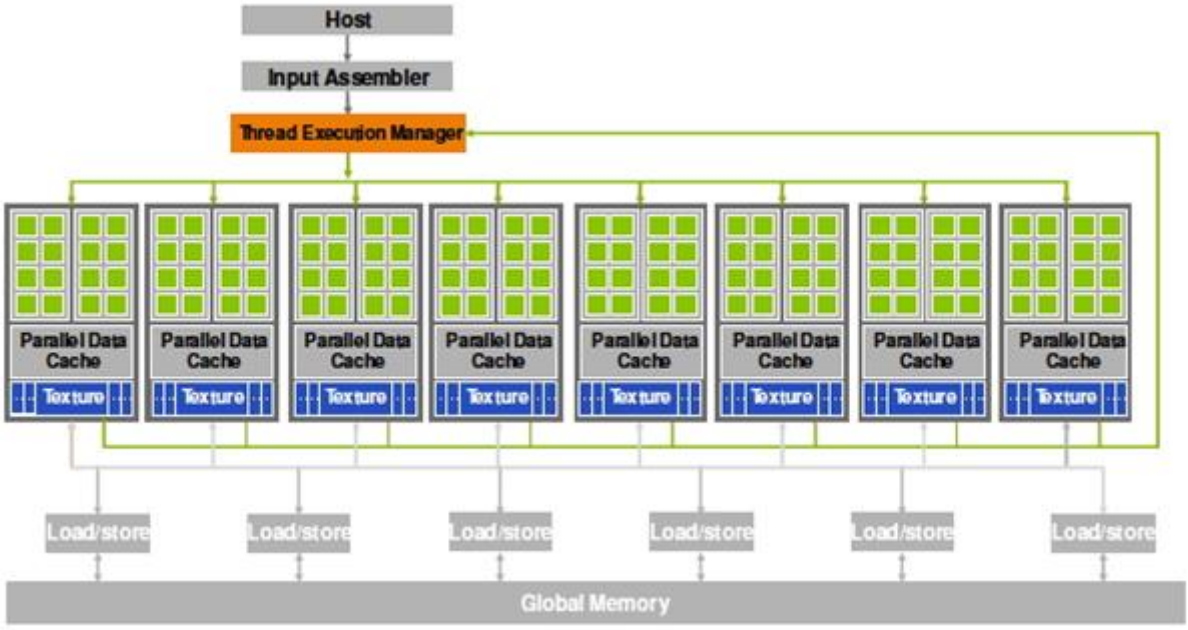

Figure 11: General architecture of a GPU

[73] For more informative details, see: *https://en.wikipedia.org/wiki/SIMD*
[74] For more informative details, see: *https://en.wikipedia.org/wiki/Single_instruction,_multiple_threads*

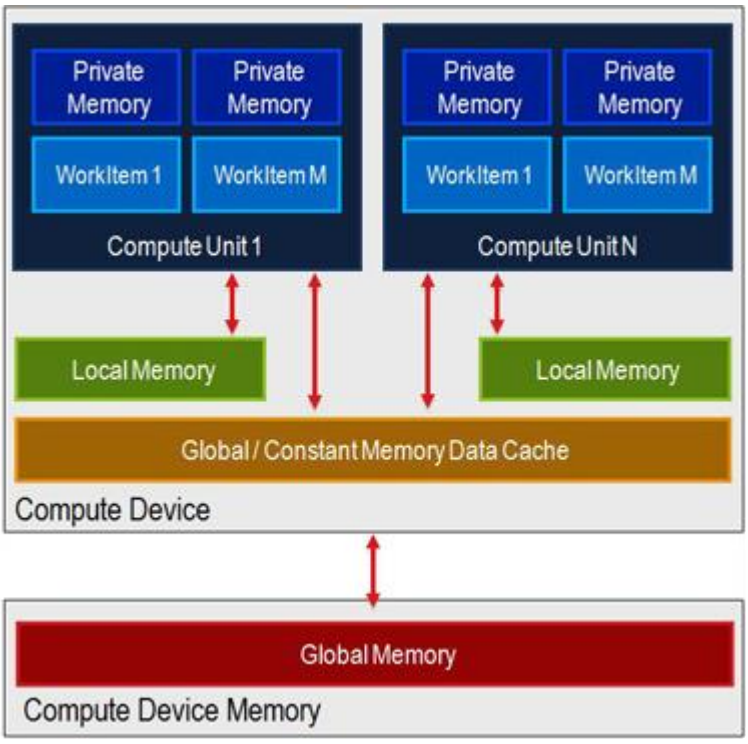

Figure 12: Typical memory organization

GPUs processing typically follows a hierarchical computing model: a GPU task (called "kernel") launches a grid of parallel tasks (called "blocks") each composed of an arbitrary number of threads. The threads execute their task following a SIMT (Single Instruction, Multiple Thread) model, similar to SIMD, but allowing more flexibility and some possibility of divergence among the threads.

3.2.2. Current GPU programming languages and tools

In order to efficiently exploit the parallel resources in the complex architecture of a GPU, specific programming tools and languages are necessary. They give the developer the full control over the GPU resources; for instance, the means to explicitly describe how the parallel threads have to be distributed under the computing units, or how and when data should be moved in parallel between global memory and local memories, and so on.

In the history of GPU computing, the need for such a programming framework was first satisfied by the producer by means of proprietary solutions, typically APIs, giving a partial control of the computing architecture and/or the possibility to call library routines running on GPU. Examples are the DirectX and OpenGL libraries [16].

In the subsequent development, all producers have evolved from such tools into programming tools giving full control over the GPU resources, thus enabling the development of general-purpose algorithms able to fully exploit the device. The most important and successful tool of this kind is NVIDIA's CUDA [17]. The consequent successes in this field has brought all the GPU producers to converge to very similar programming models (also due to the increasing similarity among their hardware architectures). Recently, they have decided to cooperate to the support and development of a common open programming language for GPU and, more generally, multicore and manycore[75] architectures: OpenCL [18].

[75] Also see: *https://en.wikipedia.org/wiki/Manycore_processor*

Today, the GPU programming environments currently supported for general-purpose HPC are: CUDA, only for NVIDIA GPUs, and OpenCL, supported by all GPUs.

3.1.1.2 NVIDIA CUDA

CUDA (Compute Unified Device Architecture) was introduced by NVIDIA in 2006. It defines both a general purpose parallel computing platform and a dedicated programming model that enables the parallel compute engines in NVIDIA GPUs to solve general-purpose computational problems in a more efficient way than on a CPU. At the user side, CUDA comes with a software environment that allows developers to use C as a programming language.

Execution hierarchy: CUDA extends C by allowing the programmer to define C functions, called kernels, that, when called, are executed N times in parallel by N different CUDA threads.

The thread index can be 1, 2, or 3-dimensional, consequently forming a 1-, 2-, or 3-dimensional thread block. All threads of a block can be executed in parallel, depending on the device capability, following a SIMT (Single Instruction, Multiple Thread) model. Blocks, as well, are organized into a 1-, 2-, or 3-dimensional array grid. The number of thread blocks in a grid is usually dictated by the size of the data being processed and can arbitrarily exceed the number of available processors.

Memory hierarchy: Corresponding to the thread execution hierarchy, there is also a hierarchy in the organization of memory in CUDA devices. Each thread has its private local memory. Each thread block has access to its shared memory visible to all threads of the block and with the same lifetime as the block. All blocks have access to the same global memory. There are also two additional read-only memory spaces accessible by all threads: the constant and texture memory spaces. The global, constant, and texture memory spaces are persistent across kernel launches by the same application.

3.1.1.3 OpenCL

The OpenCL development framework consists of three main parts: Language specification, Platform layer API and Runtime API. The language specification describes the syntax and programming interface for writing kernel programs that run on the supported accelerator. Kernels can be precompiled or the developer can decide to compile the kernel program at runtime.

The OpenCL platform model is defined as a host connected to one or more OpenCL devices, each having multiple Compute Units, each of which have multiple Processing Elements. OpenCL devices can be a GPU, DSP, or a multi-core CPU. An OpenCL device consists of a collection of one or more compute units (cores). Processing elements execute instructions as SIMD or SPMD (Single Program, Multiple Data). SPMD instructions are typically executed on general purpose devices such as CPUs, while SIMD instructions require a vector processor such as a GPU.

The Execution Model is quite similar as in CUDA: the host program is responsible for setting up and managing the execution of kernels on the OpenCL device. As in CUDA, when launching a kernel, it is specified how the tasks can be parallelized.

The memory organization is also similar to CUDA's and is schematized in, each Processing Element (work item) has its Private Memory, but also shares a Local Memory space with all other elements of its Compute Unit (the workgroup). The Global/Constant Memory is common to all Units of a device (e.g. a GPU board).

3.1.1.4 CUDA vs. OpenCL

In order to choose the GPU hardware that best fits the requirements, it can be meaningful to draw some considerations about the different alternatives present on the market.

Actually, this question has been already investigated in literature, [19]. In synthesis, OpenCL proposes a more abstract, higher-level platform model with respect to CUDA's model; this makes OpenCL code nearly device-*independent*. AMD, NVIDIA and Intel are all providing OpenCL SDK. This is a big advantage in terms of portability, allowing the same source code to be run on any GPU (and, in the near future, on hybrid GPU-CPU architectures). On the other side, this abstraction introduces an additional overhead, for mapping the high-level model to the physical underlying device. This leads generally to a performance loss, compared to the CUDA approach, in which the programming model is exactly matched to the physical NVIDIA architecture. Experimental results show that CUDA code generally outperforms the same algorithm in OpenCL, even on the same NVIDIA GPU engine.
In conclusion, although OpenCL would probably become the best-promising solution in the future, NVIDIA CUDA still remains for the moment the best approach for performance, obtaining the highest level of computing and bandwidth efficiency from a GPU device.

As computing performance is of crucial importance, the adoption of NVIDIA hardware and CUDA programming framework seems to be the most reasonable choice for SESAME.

3.1.1.5 GPU Virtualization

In the literature, GPU virtualization is usually addressed in its most simplified form, which can be stated as follows: on a hardware platform, consisting of both general purpose CPUs and specialized GPU-*based* accelerators, the guest VMs running on the CPUs must be able to concurrently and independently access the GPU computing resources without security issues [20], [22], [22].

The proposed techniques to achieve GPU virtualization can be divided in two main categories, which are usually referred to as API remoting (also known as split driver model[76] or driver paravirtualization[77]) and PCI pass-through[78] (also known as direct device assignment), *respectively.*

3.1.1.6 API remoting

Various approaches based on API remoting have been proposed in the literature. The most well-known solutions are rCUDA [20], gVirtuS [20], vCUDA [21], and GVim [21]. All such approaches adopt the same technique, which consists in splitting the GPU driver into a front-end and a back-end driver. The front-end driver is included in each VM. The back-end driver runs in a privileged domain (like the Dom0 in Xen [22]) and includes the GPU drivers and (in the case of NVIDIA devices) the CUDA library. Calls to the GPU library made within a VM are sent from the front-end to the back-end driver, which executes the CUDA API on the physical GPU on behalf of the VM. This way, many VMs can concurrently interact with one physical GPU; in particular, a software emulation of the GPU can be assigned to each VM. However, the performance of such techniques can be adversely affected by the performance of the transport channel between the front-end and the back-end drivers, as well as by the amount of data (which is application-dependent) that must be moved [22].

The open source rCUDA framework seems to be the most popular API-remoting technique to virtualize GPUs. Its main advantages with respect to the other techniques quoted above are its hypervisor-independence, as well as its maintained alignment with the CUDA APIs.

3.1.1.7 Pass-through techniques

Pass-through techniques are based on the pass-through mode made available by the PCI-Express channel [22], [22]. To perform PCI pass-through, an Input/Output Memory Management Unit[79] (IOMMU) is used. The IOMMU acts like a traditional Memory Management Unit, i.e. it maps the I/O address space into the CPU virtual memory space, so enabling the access of the CPU to peripheral devices through Direct Memory Access channels. The IOMMU is a hardware device which provides, besides I/O address translation, also device isolation functionalities, thus guaranteeing secure access to the external devices

[76] For more information see: *http://stackoverflow.com/questions/5768222/xen-split-driver-model*
[77] See, *for example*: *https://en.wikipedia.org/wiki/Paravirtualization*
[78] For more information also see: *https://wiki.openstack.org/wiki/Pci_passthrough*
[79] Also see: *https://en.wikipedia.org/wiki/Input%E2%80%93output_memory_management_unit*

[22]. Currently, two IOMMU implementations exist, one by Intel (VT-d) and one by AMD (AMD-Vi). To adopt the pass-through approach, this technology must also be supported by the adopted hypervisor. Nonetheless, Xenserver, open source Xen, VMWare ESXi, KVM and also the Linux containers can support pass-through [22].

In general, the performance that can be guaranteed by the pass-through approach are higher than the one offered by API-remoting [22], [22]. Also, the pass-through method gives immediate access to the latest GPU drivers and development tools [22].
A comparison between the performance achievable using different hypervisors (including also Linux Containers) is given in [22], where it is shown that pass-through virtualization of GPUs can be achieved at low overhead, with the performance of KVM and of Linux container very closed to the one achievable without virtualization.

One major drawback of pass-through is that it can only assign the entire physical GPU accelerator to one single VM. Thus, the only way to share the GPU is to assign it to the different VMs one after the other, in a sort of "time sharing" approach [22]. This limitation can be overcome by a technique also known as Direct Device Assignment[80] with SR-IOV (Single Root I/O Virtualization).
A single SR-IOV capable device can expose itself as multiple, independent devices, thus allowing hardware multiplexing of the physical resources. This way, the hypervisor can assign each device to a VM, and thus the physical GPU can be concurrently shared among several different tenants.
However, the only GPUs enabled to this functionality belong to the NVIDIA grid family [22]. Also, the only known hypervisors which can support this type of hardware virtualization are VMWare Sphere[81] and Citrix XenServer 6.2[82].
However, since now also KVM can support SR-IOV, there is a path towards the use of GPU hardware virtualization also with this hypervisor, though this possibility has not been mentioned so far in NVIDIA documentation.

3.2.3. GPU HW and SW integration in the micro server

The GPU, as other type of accelerators, could be connected to the micro server CPU by using different techniques:

- Via PCIe using a GPGPU PCIe Add-in Card (there are many commercially available, for example from NVIDIA);
- could be Network Attached through Ethernet using an external SoC with embedded GPU;
- as embedded resources of a SoC (integrated in the same die), as already done in many complex SoC targeting mobile computing market.

In the Light DC PoC an NVIDIA GPU QUADRO M4000 will be plugged on a PCIe slot of the micro server.
The SW development on the NVIDIA GPUs is supported by the CUDA Toolkit provided by NVIDIA itself. The Toolkit supports different operating systems (Windows, Linux and Mac OSX[83]) each one on different architecture (e.g. x.86_64).

Starting from CUDA Toolkit version 7.5 for Linux, NVIDIA has not included the ARMv8 architecture. The ARMv7 architecture was instead supported on the 6.5 version. It seems that NVIDIA has decided not to support the ARMv8 architecture, probably because ARM does not define a standard, but only a reference architecture, and then Several SoC should have been included in CUDA.

---

[80] See: *https://blogs.technet.microsoft.com/virtualization/2015/11/19/discrete-device-assignment-description-and-background/*
[81] For more details see: *http://www.vmware.com/products/vsphere.html*
[82] For more details see, *for example*: *http://xenserver.org/open-source-virtualization-download.html*
[83] Also see: *https://en.wikipedia.org/wiki/MacOS*

Considering that NVIDIA, currently, does not support with CUDA the Linux/ARMv8 architecture, in SESAME the development of NVDIA GPU based application will be done on Linux/x86_64 architectures. To this scope, the GPUs used will be plugged in a x.86 based micro server (hybrid Light DC).

4. Network Acceleration Extensions

Current solutions for networking between virtual machines on virtualization hosts, such as veth, tap and OVS represent an important number of limitations in terms of performance[84].
In fact, they are unable to provide communication at speeds higher than several Gbps and sometimes even the common speed of 1Gbps is hardly achievable. This is mainly due to their design which spans between the user-space and the kernel space.

To overcome those limitations, VOSYS extended its virtual switch solution VOSYSwitch by adding support for OpenFlow, OpenStack and ARMv8 platforms.
The details of the VOSYS virtual switch technology will be detailed in the first part of this section. After, the extensions developed in Task 4.2 will be presented. At the end of the section, an overview of a benchmark between VOSYSwitch and OVS-DPDK is presented.

4.1. VOSYSwitch virtual switch

Based on the Snabb [26] NFV framework, VOSYSwitch is a completely user-space based, modular and NFV-ready virtual switch, further enhancing the concepts that lie behind Snabb.
VOSYSwitch is configured through a JSON file which defines the switch components and their links in the form of network forwarding graph. The configuration file can be edited by the network administrator or by the OpenStack Neutron agent [27]. Furthermore, the switch architecture implements a master-worker multi-process scenario where workers are configured and controlled via shared memory communication (Figure 13).

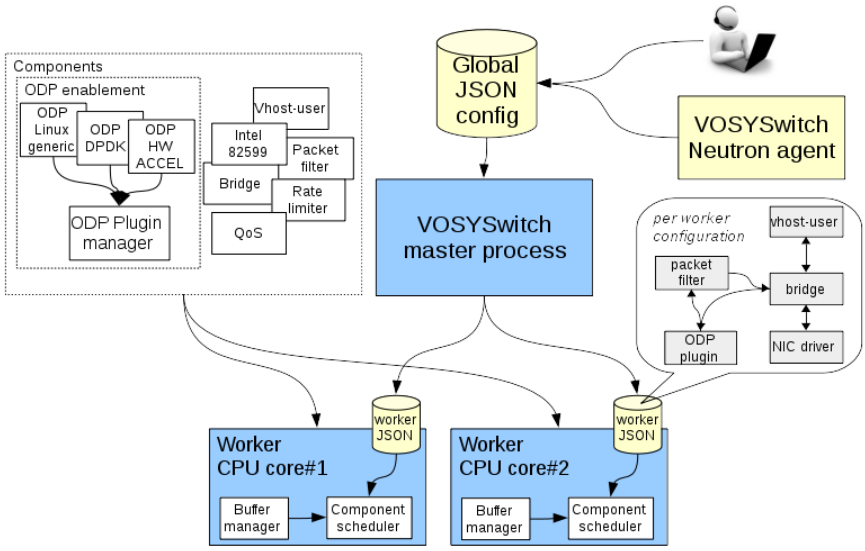

Figure 13: VOSYSwitch architecture

The number of workers that need to be created by the master is determined by the configuration file. Each worker is a single threaded process which can be pinned to as given CPU core and represents the processing graph of a single network domain. When the worker is spawned, it reads the configuration from its assigned shared memory, instantiates the required modules from the VOSYSwitch component pool and creates the links according to the graph description. As of today, I/O and processing component

[84] See: *http://linux-blog.anracom.com/2016/02/02/fun-with-veth-devices-linux-virtual-bridges-kvm-vmware-attach-the-host-and-connect-bridges-via-veth/*

types are supported by the switch. I/O components are in charge with input/output operations to physical NICs, virtual communication devices such as Vhost-user or the ODP interface which is further detailed in section 4.2.2. On the other hand, processing components implement packet and flow manipulation functions such as rate limiting, packet modification, QoS, L2 switching, VLAN support, etc.

After components instantiation, the processing graph is put together with links which are essentially FIFO queues, holding packet descriptors. The worker sequentially executes each component in the graph, which processes its input and output links. When a packet enters the graph I/O components allocate and publish a descriptor to the packet buffer on their output queue. When a packet is ready to leave for the outside world, the I/O components free the descriptor and send out the associated buffer. A buffer manager handles packet descriptors and their respective buffers.

In the scope of Task 4.2 an OpenFlow bridge processing component was designed. This enables dataplane configuration by OpenFlow controllers (e.g. OpenDayLight[85]) and makes VOSYSwitch eligible for SFC deployments. This architecture is described in subsection 4.2.3 of the present deliverable, while its implementation will be detailed in D5.2.

Since Snabb was initially meant as framework running on x.86 hardware and with respect to the SESAME innovation of using ARM-*based* micro-server, VOSYS undertook the effort to port VOSYSwitch to ARM [28]. This process is described in subsection 4.2.1.

4.1.1. VOSYSwitch supporting technologies

4.1.1.1 LuaJIT

LuaJIT[86] is a high performance implementation of Lua[87], a scripting language born in 1993 and evolved over the years to a powerful language widely used today in robotics, image processing, bioinformatics, game development, etc. [38].
LuaJIT includes a target specific interpreter written in assembly language for x.86, ARMv7, MIPS and PPC architectures.

By leveraging on LuaJIT, VOSYSwitch can benefit of a well-engineered trace based just in time (JIT) compiler [39]. Trace-*based* compilers rely on profiling execution information collected at runtime to detect and compile performance critical application fragments. Unlike most just in time compilers that operate at the method level, trace-*based* compilers delve in deeper into the control-flow of a method by profiling the execution of program paths.
The ultimate goal is to capture the smallest set of execution traces that are representative of the dynamic behaviour of the application. Doing so, a trace-*based* compiler can focus its entire optimization budget on a tiny, yet very important part of an application [40]. This concept, well known in compilers literature, is applied to networking virtualization by VOSYSwitch. As a result, the optimized machine code of this virtual switch is reflecting the actual network traffic which is passing through it. In contrast with statically compiled switches such as OVS, whose binary file is optimized once forever at compile time, VOSYSwitch is able to optimize its machine code (e.g., emitting machine code only for branches that are taken at run-time, etc.) when the traffic changes because of a new service installed in a VNF, or a new customer requests the existing service with a specific Quality of Service (QoS), or because of an unexpected event, etc.
Moreover, LuaJIT extends the basic implementation of Lua with several extensions modules which implement functional and performance enhancements. VOSYSwitch in particular leverages bit and ffi[88] modules: the former provides functions for bitwise operations, while the latter allows calling external C functions and data structures from pure Lua code.
In conclusion, the usage of Lua and LuaJIT guarantees both portability of the code on different CPU architectures and high performance [41], as well as rapid VNF development and prototyping.

---

85 See: *http://www.odl.com/*
86 For more details see: *http://luajit.org/*
87 For more details also see: *https://www.lua.org/*
88 For more details see: *http://luajit.org/ext_ffi.html*

4.1.1.2 User-space execution approach

As mentioned above, VOSYSwitch is entirely executed in user space. The reason behind this choice is avoiding CPU constantly switching between kernel and user mode, which leads to decreased performance. Moreover, execution in user-space allows dedicating resources for networking by using fine controls on processes, such as core pinning, thus eliminating a significant number of cache invalidation operations and reducing processing time.

For a packet to reach the network with currently available kernel-*based* solutions (e.g. Linux bridge[89], OVS), there is need of at least two switches between kernel and user space. In fact, when the packet is generated in the VM, which runs in kernel mode, it has to reach QEMU which then returns it to the host kernel via TAP virtual device[90] in order to reach the dataplane. The same operations are performed in reverse order when a VM receives a packet from the network. In total, there are four mode switching operations to transfer a single packet from one VM to another, as shown in Figure 13.

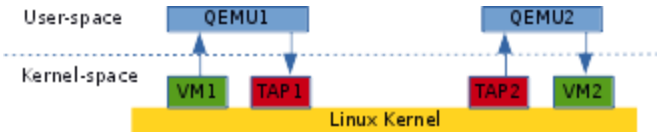

Figure 14: Packet flow between kernel and user space in traditional network virtualization solutions

On the other hand, the VOSYSwitch approach is to commute packets from QEMU in the user-space, therefore eliminating two context switches, as depicted in Figure 15.

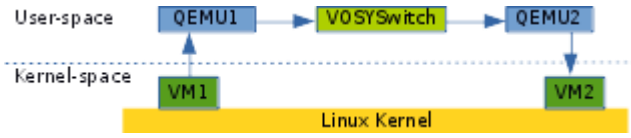

Figure 15: Packet flow between kernel and user space with VOSYSwitch

4.1.1.3 Zero-copy data transfer with Vhost-user

As shown in the previous section, traditional virtual networking solutions mostly use TAP interfaces to attach VMs to the network. This, however, comes at the price of several packet copies in memory for a single transfer. To cope with this problem VOSYSwitch takes advantage on the Vhost-user technology [31]. Vhost-user provides a standard virtio transport for network communication between two processes using shared memory, ioeventfds and irqfds[91]. On the VM side the network interface is presented as paravirtualized virtio-net device[92], while on the host side it can be accessed by any user space process implementing a virtio-net backend. An UNIX[93] domain socket on the host allows the configuration of Vrings which will be placed in the shared memory between the processes and the necessary eventfds[94] to process notifications when a Vring is modified by either sides.

[89] *https://wiki.linuxfoundation.org/networking/bridge*
[90] See, *for example,* the discussion in: *https://en.wikipedia.org/wiki/TUN/TAP*
[91] See, *for example*: *http://blog.allenx.org/2015/07/05/kvm-irqfd-and-ioeventfd*
[92] For more related information see, *for example*: *http://www.linux-kvm.org/page/Virtio*
[93] *https://en.wikipedia.org/wiki/Unix*
[94] See: *http://man7.org/linux/man-pages/man2/eventfd.2.html*

4.2. SESAME extensions to VOSYSwitch

This section documents the VOSYSwitch extensions and enhancement specifically developed in T4.2.

4.2.1. Porting VOSYSwitch on ARM

The original Snabb implementation which serves as a base for VOSYSwitch was intended for use on 64-bit Intel x.86 systems. In order to leverage the VOSYSwitch performance in the SESAME micro-servers based on ARMv8, VOSYS ported this solution to the new 64 bit architecture from ARM.

To achieve this, a number of extensions have been developed going into the direction of closing the gaps between these two architectures. For example, Snabb effectively implements checksum algorithms using the Intel SSE[95]/AVX[96] instruction set. Since those instructions are only available on Intel x.86 systems, VOSYS ported all of them to the alternative ARM instruction set extensions, namely NEON.

Moreover, since LuaJIT is only running in interpreted mode on 64-bit ARM systems (aarch64[97]), it is impossible to take advantage on the hot tracing mechanisms of the JIT. To overcome this, VOSYSwitch for ARM runs with 32-bit mode LuaJIT. In fact, aarch32 instruction set is natively supported on ARMv8 platforms. However, running in 32-bit mode presents several important incompatibilities with the implementation of Snabb, mainly due to the memory addressing and to the Linux kernel pages.
Such incompatibilities have been resolved, and today VOSYSwitch is able to run on platforms such as the NXP LS2085.

Today, 64-bit ARM platforms are starting to be supported in LuaJIT by the open source community (version 2.1 beta [30]). An important role on this has been played by the VOSYS dissemination of the LuaJIT capabilities in the networking field, done in the context of SESAME.

Thanks to this LuaJIT support for ARMv8, in future VOSYSwitch will be able to fully exploit LuaJIT on aarch64.

4.2.2. ODP Plugins

In order to provide a complete virtual networking solution on the compute host, VMs need to be connected to the external world. This is achieved through a physical NIC. VOSYSwitch disposes of I/O component, implementing a user-space driver for the Intel 82955[98] 10Gbps NIC, however, providing specific drivers for every specific NIC or even only for the widely spread network adapters is virtually impossible.
Moreover, the aim for versatility and portability of VOSYSwitch on software and hardware platforms excludes such approach. The solution is to use an abstraction layer that would provide unified and agnostic interface for communication with network interfaces such as Open Data Plane (ODP), as already mentioned in 4.1.

ODP is a collection of cross-platform API for software defined networking data plane. It enables application portability and allows exploitation of platform specific acceleration and offload capabilities. ODP consists of an abstract API specification, which is vendor and platform neutral and various implementations, targeting different platforms listed in Table 1 [32].
ODP APIs are designed to address application needs rather than exposing specific platform capabilities, thus applications can run on any platform even if developers are agnostic about the fine details of that platform.

---

95 For more details see: *https://en.wikipedia.org/wiki/Streaming_SIMD_Extensions*
96 For more details see: *https://en.wikipedia.org/wiki/Advanced_Vector_Extensions*
97 For more details see, *for example*: *https://fedoraproject.org/wiki/Architectures/AArch64*
98 Also see: *http://ark.intel.com/products/27727/Intel-82955X-Memory-Controller*

Name	Owner/Maintainer	Target platform	Architecture
odp-generic	Linaro Networking Group (LNG), open contribution	Platform independent, Linux	Any
odp-dpdk	LNG, open contribution	Intel x86 using DPDK	Intel x86
odp-netmap	LNG, open contribution	Linux + Netmap	X86 + ARM
odp-keystone2	Texas instruments	TI Keystone II	ARM A15
odp-nadk	NXP	NXP LSxxxx	ARMv8
linux-qoriq	NXP	NXP QorIQ	Power, ARMv8
OCTEON	Cavium Networks	Cavium Octeon	MIPS64
THUNDER	Cavium Networks	Cavium ThunderX	ARMv8
odp-mppa	Kalray	Kalray MPPA	Proprietary

Table 1: Currently available ODP implementations

VOSYSwitch leverages ODP implementations as a plugin system, allowing to dynamically load the extensions for the specific platform where the switch is being executed from a precompiled shared object. ODP is made available to VOSYSwitch workers through a separate I/O component, thus can be configured and used as any other device available in the component pool. In fact, VOSYS being a pioneer in using ODP implementations as plugins, several API incompatibilities were reported [28] and were later fixed by the ODP community.

At the Linaro Connect event in Bangkok in March 2016 [28], where SESAME was disseminated, VOSYS used odp-nadk to enable direct access to the 10Gbps NIC, embedded on the NXP LS2085A platform.

4.2.3. OpenFlow support

In the scope of SESAME, OpenFlow 1.3[99] support was added to VOSYSwitch. This enables the switch to be part of complex scenarios involving SDN and SFC, which is the case in SESAME.

The current implementation consists of two distinct components. The first one is the processing component, OpenFlow bridge, which processes packets through the flow table and applies the rules.
The second component is an OpenFlow client which communicates with the controllers. It receives instructions and applies them to the bridge tables. Moreover, it responds to multipart requests from the controller about statics or the state of the bridge. Since VOSYSwitch worker processes are single threaded, the execution of the client is scheduled by an interruption timer and is programmed when the OpenFlow bridge is instantiated.

[99] See: *http://www.brocade.com/content/html/en/configuration-guide/nos-601a-sdnguide/GUID-3CBDC5D1-C379-4CE2-9B5A-36DFDE4B57A4.html*

4.3. Benchmark between VOSYSwitch and OVS-DPDK

In the context of Task 4.2 several benchmarks with OVS-DPDK, which is the VOSYSwitch closest available product in terms of functionality, have been performed. VOSYS presented them in the scope of dissemination activities at international events such as [33], [34].

This section gives an overview these benchmarks results.

4.3.1. Test scenarios

For the purpose of the benchmark, three test scenarios, presented on Figure 15 have been considered:

- **Host vSwitch**: the virtual switch is forwarding the traffic between two ports without any additional packet processing application.
- **Single VM**: traffic flows through a single VM, representing a VNF attached to the virtual switch.
- **VM to VM**: using direct device assignment, traffic from physical NIC ports flow directly into the VMs. The virtual switch then forwards the traffic between the virtual ports of the VMs.

Those scenarios are widely used for benchmarking virtual switches [36], [36].

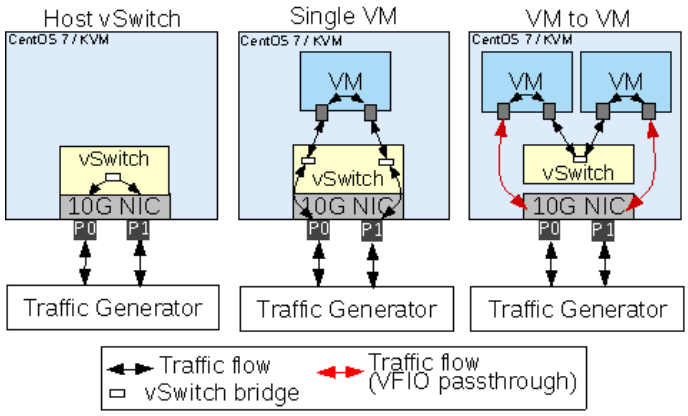

Figure 16: Benchmark scenarios

4.3.2. Benchmark environment

The benchmark was performed using the IXIA XM2 traffic generator. It disposes of two 10Gbps ports connected to the NIC ports of the virtualization host. The traffic is biderctional, resulting in 20Gbps total throughput. VMs are attached through Vhost-user on both VOSYSwitch and OVS-DPDK.

The software configuration for the test is the following:

- CentOS 7[100] on host and VMs
- QEMU v2.2.0

- OVS-DPDK:
  - OVS v2.5.0
  - DPDK v2.2.0
  - OVS bridge with default configuration
- VOSYSwitch v1.0 with L2 learning bridge component inserted in the processing graph

[100] For more details see: *https://www.centos.org/*

Since OVS-DPDK is only available on Intel x.86 platforms, this is the only possible architecture to perform the benchmark. The hardware configuration for this test is:

- Huawei RH2288H V3 server[101]
- 2 x Intel Xeon E5-2697 V3 CPU[102] at 2.6Ghz (14 cores)
- Dual-port Intel X520[103] NIC
- 128GB DDR4 at 2133Mhz

Performance metrics are measured by an IXIA hardware tester[104] and are based on the RFC2544 benchmark specification [37] with three loss tolerances: 0.0004%, 0.01% and 5%.

4.3.3. Benchmark results

4.3.3.3.1 Host vSwitch

This benchmark tests the switching engine and its scalability by flooding it with different L2 flows. It shows that while VOSYSwitch almost reaches line speed with 512 byte packets at all three loss tolerances, OVS-DPDK is only able to achieve line speed with 1024 byte packets and the biggest loss tolerance (Figure 17).

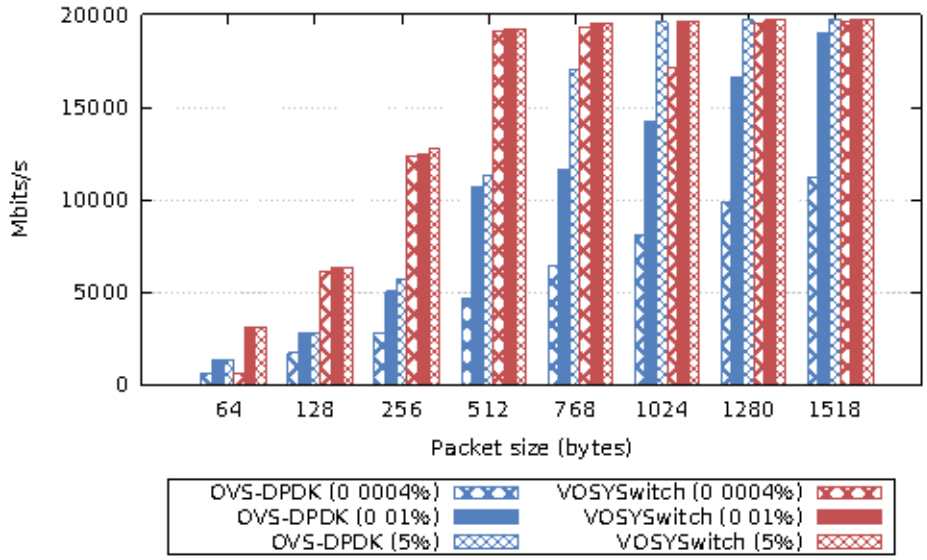

Figure 17: Host switching throughput (1000 flows)

4.3.3.3.2 Single VM

While the first scenario (Section 4.3.3.3.1) was testing purely the switching engine, this second scenario adds a VM attached to the switch, forwarding packets. In this case line speed is impossible to achieve, given the fact that the traffic crosses the VM dataplane. Performance results for 0.0004% packet loss limitation are instable, therefore inconclusive. Still, VOSYSwitch outperforms OVS-DPDK in the tests with 0.01% and 5% limitations, as visible on Figure 18.

[101] See: *http://e.huawei.com/en/products/cloud-computing-dc/servers/rh-series/rh2288h-v3*
[102] See: *http://ark.intel.com/products/81059/Intel-Xeon-Processor-E5-2697-v3-35M-Cache-2_60-GHz*
[103] See: *http://www.intel.com/content/www/us/en/ethernet-products/converged-network-adapters/ethernet-x520.html*
[104] See: *https://www.ixiacom.com/products-services/test-hardware*

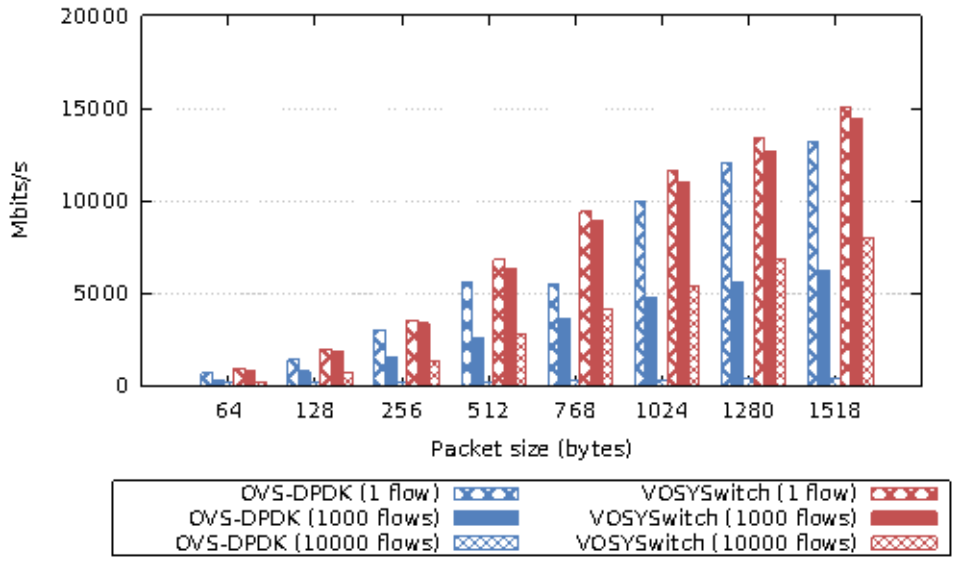

Figure 18: Single VM throughput (1000 flows)

4.3.3.3.3 VM to VM switching

The VM to VM scenario is intended to isolate the vSwitch NIC driver and measure only switching capabilities between the Vhost-user interfaces of the VMs. While VOSYSwitch shows stable results for all tested loss tolerances, OVS-DPDK demonstrates an important performance gap between the results obtained for low loss tolerances and the one for 5% packet loss tolerance (Figure 19). Once again VOSYSwitch performs better than OVS-DPDK.

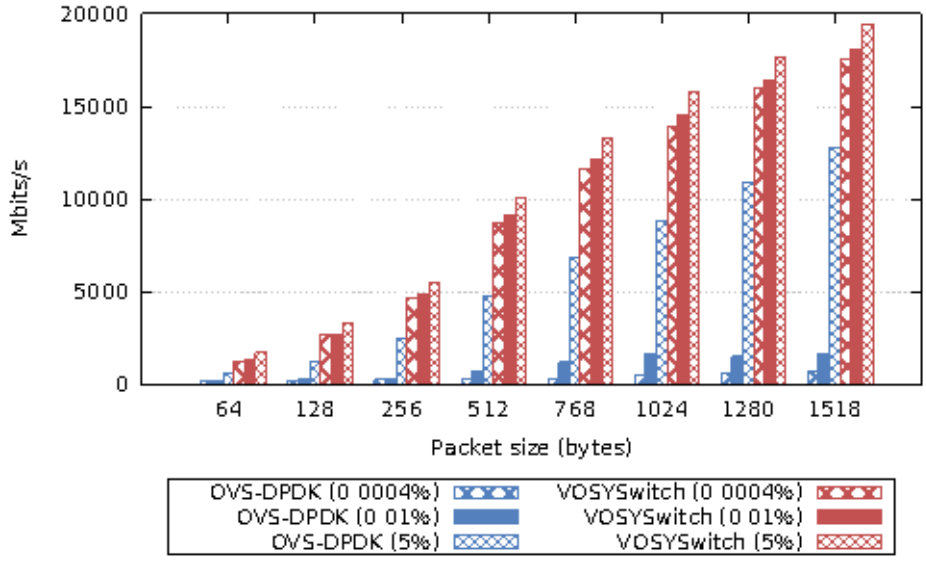

Figure 19: VM to VM throughput (1000 flows)

5. Conclusion

This deliverable outlines the results of Task 4.2 with regard to virtualization extensions for acceleration of the SESAME Light DC, based on the goals described in the DoW.

Partners, effort has been focused in two directions: The first one is accelerating computation capabilities by offloading heavy computational tasks to specialized hardware such as FPGA and GPU. The second one aims to provide solutions for minimizing and even eliminating performance penalty factors related to the concepts of virtual networking, thus accelerating communication between VMs and with the external world.

As for the computation acceleration, presented in Section 3, the FPGA-*based* virtualization and developed in the scope of SESAME provides the flexibility needed for hardware acceleration in the cloud, by enabling the execution of desired software-*defined* hardware accelerators on general purpose hardware.
This is very important for the SESAME architecture to support different VNFs at its edge. For what concerns the GPU virtualization, the investigations on using this type of hardware acceleration together with the new Intel road-map addressing x.86 CPUs specifically thought for edge computing environment, brought the Consortium to a more realistic view about the implementation of the Light DC, transforming it from a homogeneous cluster of identical micro-servers to a heterogeneous platform containing nodes designed to provide specific hardware acceleration capabilities.

Regarding the network acceleration (Section 4), SESAME took further the concept of user-space networking by combining outstanding open-source technologies in this area and by dissecting traditional solutions in order to identify and avoid the bottlenecks of the current state of the art. Moreover, the results of those efforts found their ways in exploitation in the VOSYSwitch virtual switch.

In conclusion, SESAME task 4.2 provides a robust acceleration framework for the NFVI, both in terms of design concepts, hardware and software. This task will be essential for SESAME to meet up with the criteria set for the performance of future 5G infrastructures. In the scope of the Task 4.2, VOSYS conceived and developed solutions for accelerating both the VNFs deployed in VMs and the network infrastructure of the Light DC, ITL focused their efforts to make standard GPUs available for the platform, while ST provided a software baseline supporting virtualization for the ST Barcelona board. The software developed in Task 4.2 will be integrated in the SESAME testbed and will be part of the project final demonstrations.

6. References

[1] SESAME Deliverable D2.2, "Overall System Architecture and Interfaces", available on *http://www.sesame-h2020-5g-ppp.eu*

[2] SESAME Deliverable D2.3, "Specification of the CESC components", available on *http://www.sesame-h2020-5g-ppp.eu*

[3] SESAME Deliverable D4.1, "Light DC Architecture Design", available on *http://www.sesame-h2020-5g-ppp.eu*

[4] Rafi Khardalian, Under the hood with Nova, Libvirt and KVM *https://www.openstack.org/assets/presentation-media/OSSummitAtlanta2014-NovaLibvirtKVM2.pdf*

[5] Virtual Open Systems website, How to create a NXP SDK6 Yocto image with QEMU and VOSYSwitch for LS2085A-RDB, *http://www.virtualopensystems.com/en/solutions/guides/yocto-qemu-kvm-vswitch-nxp-ls2085a/*

[6] Extremetech, Intel's next generation chips will sacrifice speed to reduce power, February 2016: *http://www.extremetech.com/extreme/222590-an-end-to-scaling-intels-next-generation-chips-will-sacrifice-speed-to-reduce-power*

[7] ETSI NFV, IFA001 Acceleration technologies, report on acceleration technologies and use cases: *http://www.etsi.org/deliver/etsi_gs/NFV-IFA/001_099/001/01.01.01_60/gs_NFV-IFA001v010101p.pdf*

[8] ETSI NFV, Acceleration Technologies; Management Aspects Specification, April 2016: *http://www.etsi.org/deliver/etsi_gs/NFV-IFA/001_099/004/02.01.01_60/gs_NFV-IFA004v020101p.pdf*

[9] OpenStack wiki, Nomad project web page, March 2016: *https://wiki.openstack.org/wiki/Nomad*

[10] OPNFV DPACC page, *https://wiki.opnfv.org/display/dpacc*

[11] Amazon's Xilinx FPGA Cloud: Why This May Be A Significant Milestone, *http://www.forbes.com/sites/moorinsights/2016/12/13/amazons-xilinx-fpga-cloud-why-this-may-be-a-significant-milestone/#488c178f3fb4*

[12] Intel, Intel to acquire Altera, June 2015: *http://www.intc.com/releasedetail.cfm?ReleaseID=915707*

[13] Xilinx, UltraScale MPSoC Architecture website, March 2016: *http://www.xilinx.com/products/technology/ultrascale-mpsoc.html*

[14] Christian Pinto, Michele Paolino, and Salvatore Daniele Raho. "Virtualization manager for reconfigurable hardware accelerators": EPO Application No. EP3089035A1, U.S. Patent Application No.15/142,132.

[15] Intel (2013): "P C I-S I G SR-IOV Primer - An Introduction to SR-IOV Technology" .

[16] D.B. Kirk, W.W. Hwu (2013): *Programming Massively Parallel Processors, 2nd ed.*, Morgan Kaufmann.

[17] NVIDIA, CUDA C Programming Guide, 2016.

[18] AMD, Introduction to OpenCLTM Programming, 2014

[19] K. Karimi, N.G. Dickson, F. Hamze (2011, May): A Performance Comparison of CUDA and OpenCL, Cornell University Library, arXiv:*1005.2581v3, arxiv.org.*

[20] J. Duato, A.J. Pena, F. Silla, J.C. Fernandez, R. Mayo, E.-S. Quintana-Orti (2011): "Enabling CUDA acceleration within virtual machines using rCUDA", *in Proceedings of the 18th International Conference on High Performance Computing (HiPC)*, pp.1-10, Bangalore, India, *December 18-21, 2011* doi: *10.1109/HiPC.2011.6152718*

[21] R. Di Lauro, F. Giannone, L. Ambrosio, R. Montella (2012): "Virtualizing General Purpose GPUs for High Performance Cloud Computing: An Application to a Fluid Simulator", in *Proceedings of the IEEE 10th International Symposium on Parallel and Distributed Processing with Applications (ISPA)*, pp.863-864, Leganes, Madrid, Spain *July 10-13, 2012*, doi: *10.1109/ISPA.2012.136*

[22] L. Shi, H. Chen, and J. Sun (2012, June): vCUDA: GPU-Accelerated High-Performance Computing in Virtual Machines, *IEEE Transactions on Computers*, *61*(6) pp.804-816, doi: *10.1109/TC.2011.112*

[23] V. Gupta, A. Gavrilovska, K. Schwan, H. Kharche, N. Tolia, V. Talwar, and P. Ranganathan (2009): "GViM: GPU-accelerated virtual machines", in *Proceedings of the 3rd Workshop on System-level Virtualization for High Performance Computing*, pp.17-24. ACM, NY, USA.

[24] J.P. Walters, A.J. Younge, D.-I. Kang, K.-T. Yao, M. Kang, S.P. Crago, and G.C. Fox (2014, June): "GPU-Passthrogh Performance: A Comparison of KVM, Xen, VMWare ESXi, and LXC for CUDA and OpenCL Applications", in *Proceedings of the 7th IEEE International Conference on Cloud Computing (CLOUD 2014)*, IEEE. Anchorage, AK.

[25] C. Maurice, C. Neumann, Olivier Heen, and A. Francillon (2014): "Confidentiality Issues on a GPU in a Virtualized Environment", in *Proceedings of the Eighteenth International Conference on Financial Cryptography and Data Security (FC'14)*, Barbados, *March 03-07, 2014*

[26] Snabb website, *http://snabb.co/*

[27] Openstack networking neutron home page, *http://docs.openstack.org/developer/neutron/*

[28] Virtual Open Systems website, VOSYSwitch leverages ODP to enable accelerated VNFs chaining on ARMv8 NFV servers, *http://www.virtualopensystems.com/en/solutions/demos/vosyswitch-odp-armv8/*

[29] Optimizations performed by LuaJIT, *http://wiki.luajit.org/Optimizations*

[30] LuaJIT v2.1 commits, *https://github.com/LuaJIT/LuaJIT/commits/v2.1*

[31] Virtual Open Systems website, *Vhost-user for QEMU, http://www.virtualopensystems.com/en/solutions/guides/snabbswitch-qemu/*

[32] B. Fischofer (2016): *Open Data Plane Overview and Direction,* Linaro Networking Group *http://www.opendataplane.org/wp-content/uploads/2014/01/ODP-Overview-Long-January-2016.pdf*

[33] Virtual Open Systems website, VOSYSwitch benchmarked against OVS-DPDK, Interop 2016, Japan, *http://www.virtualopensystems.com/en/solutions/demos/vosyswitch-interop/*

[34] Michele Paolino, Jérémy Fanguède, Nikolay Nikolaev and Daniel Raho, Turning an open source project into a carrier grade vSwitch for NFV: VOSYSwitch challenges & results

[35] Lightreading, Validating adva's virtual switch, *http://www.lightreading.com/nfv/nfv-specs-open-source/validating-advas-virtual-switch/d/d-id/723220*

[36] Lightreading, Validating Cisco NFV infrastructure, *http://img.lightreading.com/downloads/Cisco-ValidatingNFV-Infrastructure-Pt1-and-2-SH-Edits.pdf*

[37] S. Bradnet and J. McQuaid (1999, March): RFC 2544: "Benchmarking methodology for network interconnect devices", IETF

[38] R. Ierusalimschy, L.H. de Figueiredo, and W. Celes (2007): "The evolution of lua", in *Proceedings of the third ACM SIGPLAN conference on History of programming languages*, pp.2.1-2.26. ACM, San Diego, California, *June 09-10, 2007*

[39] C.S. Steele and J. Bonn (2012): "Fast functional simulation with a dynamic language", in *Proceedings of the 2012 IEEE Conference on High Performance Extreme Computing (HPEC),* pp.1-3, Waltham, MA, USA, *September 10-12, 2012*.

[40] M. Bebenita, M. Chang, G. Wagner, A. Gal, C. Wimmer, and M. Franz (2010): "Trace-based compilation in execution environments without interpreters", in *Proceedings of the 8th International Conference on the Principles and Practice of Programming in Java (PPPJ'10)*, pp59-68. ACM, Vienna, Austria, *September 15-17, 2010*.

[41] "Performance of several languages", May 2015. [*Online]. Available: http://blog.carlesmateo.com/2014/10/13/performance-of-several-languages/*