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 D4.4

Light DC prototype

Document Number:	H2020-5GPPP-GA No.671596/*WP4/D4.4/30.06.2017*
Contractual Date of Delivery:	*30.06.2017*
Editor:	Antonino Albanese – ITL (Italtel S.p.A.)
Work-package:	*WP4*
Distribution / Type:	Public (PU) / Report (R)
Version:	1.0
Total Number of Pages:	73
File:	*SESAME_Deliverable 4.4_v1.0_Final*

Abstract

This document describes the HW/SW integration and testing of the Light DC prototype used for the PoC of the SESAME project.

The deliverable describes the physical architecture of the selected PoC for the Light DC, the different types and characteristics of the micro-server nodes (ARM and x86 based), as well as the interfaces, networking and storage resources.

Furthermore, the virtualization platform related activities, specifically developed for the NXP micro-server prototype, are depicted as well as the activities performed for enabling the ARM-*based* node to be managed by OpenStack.

The integration activities includes the development and testing of a service Virtual Network Function (VNF) that provides video transcoding service at the edge (VTU) on the hybrid (ARM/x86) Light DC. The VTU service VNF is used to "get" some preliminary KPI regarding performance, power consumption and cost, considering the ARM and x86 platforms; the case of HW acceleration (GPU) usage is also investigated.

The contents of this deliverable will be used as "input" to the WP7, where there will be included the outcomes of the WP3, WP5 and WP6 development activities and prototypes.

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	02.05.2017	Initial draft by ITL	A. Albanese
0.2	23.05.2017	Inclusion of contributions by ITL	A.Albanese – C. Meani – M. Beccari-P.Paglierani
0.3	25.05.2017	Inclusion of contributions by STM	M. Coppola
0.4	29.05.2017	Inclusion of contribution by VOSYS	N. Nikolaev – M. Paolino
0.5	08.06.2017	Updated ITL contribution (5.3)	C. Meani
0.6	09.06.2017	Inclusion of contribution by ZHAW	G. Ankeshian - A. Edmonds
0.7	12.06.2017	Updated VOSYS contribution (4.3)	N. Nikolaev
0.8	15.06.2017	Inclusion of contributions by i2CAT	A. Betzler, P. Sayyad Khodashenas
0.9	15.06.2017	Updated ZHAW contribution	I. Trajkovska
0.10	21.06.2017	Inclusion of contributions by ORION	E. Kafetzakis
0.11	22.06.2017	Inclusion of contributions by NCSRD	I. Giannoulakis
0.12	23.06.2017	General Review	A. Albanese - C. Meani
0.13	27.06.2017	General Review by SMNET	A. Dardamanis
1.0	29.06.2017	Final editorial and conceptual review by OTE	I. Chochliouros

Contributors

First Name	Last Name	Partner	Email
Antonino	Albanese	ITL	antonino.albanese@italtel.com
Claudio	Meani	ITL	claudio.meani@italtel.com
Pietro	Paglierani	ITL	pietro.paglierani@italtel.com
Marco	Beccari	ITL	marco.beccari@italtel.com
Marcello	Coppola	STM	marcello.coppola@st.com
Nikolay	Nikolaev	VOSYS	n.nikolaev@virtualopensystems.com
Michele	Paolino	VOSYS	m.paolino@virtualopensystems.com
Andrew	Edmonds	ZHAW	edmo@zhaw.ch
Gabriel	Ankeshian	ZHAW	anke@zhaw.ch
August	Betzler	i2CAT	august.betzler@i2cat.net
Pouria	Sayyad Khodashenas	i2CAT	pouria.khodashenas@i2cat.net
Irena	Trajkovska	ZHAW	traj@zhaw.ch
Ioannis	Giannoulakis	NCSRD	giannoul@iit.demokritos.gr
Emmanouil	Kafetzakis	ORION	mkafetz@orioninnovations.gr
Athanassios	Dardamanis	SMNET	adardamanis@smart.net.gr
Evangelos	Sfakianakis	OTE	esfak@oteresearch.gr
Ioannis	Chochliouros	OTE	ichochliouros@oteresearch.gr

Glossary

Acronym	Explanation
3GPP	Third Generation Partnership Project
4G	Fourth Generation of Mobile Communications
5G	Fifth Generation of Mobile Communications
ADSL	Asymmetric Digital Subscriber Line
AES	Advanced Encryption Standard
AIOP	All-in-one-PC
AMPQ	Advanced Message Queuing Protocol
API	Application Programming Interface
ARM	Advanced RISC Machine
ATF	ARM Trusted Firmware
ATX	Advanced technology Extender
CESC	Cloud Enabled Small Cell
CESCM	CESC Manager
CLI	Command-line User Interface
CP	Control Plane
CPU	Central Processing Unit
CSI	Camera Serial Interface
DB	Database, Data Base
DC	Data Centre
DCE	Data Circuit-terminating Equipment
DDR	Double Data Rate
DDR3	Double Data Rate type three
DDR4	Double Data Rate fourth-generation
DIMM	Dual In-line Memory Module
DIP	Dual In-line Package
DMA	Direct Memory Access
DMIPS	Dhrystone MIPS
DP	Data Plane
DRD	Dual-Role Device
DSI	Display Serial In terface
DSL	Digital Subscriber Line
DUART	Dual Universal Asynchronous Receiver/Transmitter
DVD	Digital Video Disc
DVFS	Dynamic Voltage and Frequency Scaling
E2E	End-to-End
ECC	Error-Correcting Code
eMMC	embedded MultiMedia Card
EPC	Evolved Packet Core
FCBGA	Flip Chip Ball Grid Array
FD-SOI	Fully Depleted Silicon On Insulator
FDB	Forwarding Database
FF	Fast Forwarding
FFMPEG, FFmpeg	Fast Forwarding Moving Pictures Experts Group
FMC	Fundamental Modelling Concepts
FPGA	Field Programmable Gate Array
fps	frames per seconds
FSO	Free space optics
GA	Grant Agreement
GB	Giga Bytes
GbE	Gigabit Ethernet

GDB	GNU Debugger
GHz	Giga Hertz
GPIO	General Purpose Input/Output
GPGPU	General-Purpose Computing on Graphics Processing Units
GPU	Graphics Processing Unit
GRE	Generic Routing Encapsulation
GUI	Graphics User Interface
H2020	Horizon 2020
HD	High Definition
HDMI	High-Definition Multimedia Interface
HEVC	High Efficiency Video Coding
HW	Hardware
I/O, i/o	Input/Output
ICT	Information and Communication Technology
IEEE	Institute of Electrical and Electronic Engineers
IOMMU	Input-Output Memory Management Unit
IP	Internet Protocol
iSCSI	internet Small Computer System Interface
JTAG	Joint Test Action Group
JPEG	Joint Photographic Experts Group
JSON	JavaScript Object Notation
KHz	Kilo Hertz
KPI	Key Performance Indicator
KVM	Kernel-*based* Virtual Machine
L1	Physical Layer
L2	Data Link Layer
L2TP	Layer 2 Tunnelling Protocol
L2TPv3	Layer 2 Tunnelling Protocol version 3
LAN	Local Area Network
LCD	Liquid Crystal Display
Light DC	Light Data Centre
LMDS	Local Multipoint Distribution Service
LUN	Logical Unit Number
µs	micro-server
Mbps	Mega-bits per second
MAC	Medium Access Control
MEC	Mobile Edge Computing
MIMO	Multiple-Input Multiple-Output
MIPS	Million Instructions per Second
MMC	MultiMedia Card
MMU	Memory Management Unit
MP3	MPEG Layer 3
MPEG	Moving Pictures Experts Group
NAS	Network Attached Storage
NFV	Network Functions Virtualization
OASIS	Organization for the Advancement of Structured Information Standards
OC	Optical Carrier
ODL	OpenDayLight
OPNFV	Open Platform for NFV
OS	Operating System
OSD	Object Storage Daemon
OVS	OpenvSwitch
P2P	Point-to-Point
PC	Personal Computer
PCI	Peripheral Component Interconnect

PCIe	Peripheral Component Interconnect Express
PDH	Plesiochronous Digital Hierarchy
PHY	Physical Layer
PNF	Physical Network Function
PoC	Proof of Concept
PoP	Point of Presence
PPP	Public-Private Partnership
PS	Power Supply
Qemu, QEMU	Quick Emulator
QoE	Quality of Experience
QoS	Quality of Service
R/W, r/w	Read/Write
RAID	Redundant Array of Independent Disks
RAM	Random Access Memory
REST	Representational State Transfer
RFC	Request for Comments
RGMII	Reduced gigabit media-independent interface
RIA	Research and Innovation Action
RJ	Registered Jack
SAN	Storage Area Network
SATA	Serial Advanced Technology Attachment
SBC	Single-Board Computer
SC	Small Cell
SC-C-VNF	Small Cell-Common-VNF
SD	Secure Digital
SDK	Software Development Kit
SDN	Software-Defined Networking
SDXC	Synchronous Digital Extended Capacity
SFP	Small Form-Factor Pluggable
SLA	Service Level Agreement
SMMU	System Memory Management Unit
SoC	System on Chip
SONET	Synchronous Optical NETworking
SOTA	State-of-the-Art
SPI	Serial Peripheral Interface
SRIOV	Single Root Input Output Virtualization
STM	Synchronous Transport Module
SW	Software
ToR	Top of the Rack
TU	Transcoding Unit
UC	Use Case
UEFI	Unified Extensible Firmware Interface
UP	User Plane
URL, url	Uniform Resource Locator
USB	Universal Serial Bus
VGA	Video Graphics Array
VIM	Virtualised Infrastructure Manager
VLAN	Virtual Local Area Network
VLUT	Volume Look-up Table
VM	Virtual Machine
VNF	Virtual Network Function
vTU	virtual Transcoding Unit
VTU	Video Transcoding Unit
VxLAN	Virtual Extensible LAN
WAN	Wide Area Network

WiFi	IEEE 802.11 protocols family
WiMax	Worldwide Interoperability for Microwave Access
WP	Work Package
xDSL	DSL technologies
ZFS	Z File System

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Deliverable outline

The present SESAME deliverable covers the Light DC prototype HW and SW integration, and includes the following sections:

- *Section 1* offers a brief introductory overview.
- *Section 2* recaps the architecture of the Light DC designed to provide computing, storage and networking resources to CESCs.
- *Section 3* describes the Light DC physical architecture chosen for the PoC, in particular: the different types and characteristics of the micro-server nodes (ARM and x86-*based*), as well as the interfaces, networking and storage resources.
- *Section 4* describes all the virtualization-*related* activities performed on platform prototype, in order to get a service VNF running on the NXP micro-server and correctly managed by OpenStack.
- *Section 5* reports the results achieved in the integration of a service VNF on the virtualized ARMv8-*based* micro-server managed by OpenStack. Some preliminary KPI regarding performance, power consumption and cost are also presented.
- Finally, *Section 5* summarises the "key topics" discussed in the document and concludes the deliverable.

1.2 Definitions of Terms and SESAME concepts

At this point, it is useful to provide definitions of terms and processes which will be used later in this document to describe the SESAME main concepts.

- **Small Cell (SC):** Does not change in the context of SESAME.
- **Execution infrastructure, micro-server:** Specific hardware that provides processing power, memory, storage and networking capabilities to the Small Cell.
- **CESC (Cloud Enabled Small Cell):** An intelligent entity composed of two, optionally co-located, network-connected physical devices: a Small Cell Physical Network Function (PNF) and a micro-server (µS) platform, which form a node in the distributed Light Data Centre (Light DC).
- **Cluster of CESCs:** A group of CESCs that are locally connected together, exchange information and are properly coordinated.
- **Light Data Centre (Light DC):** A micro-scale virtualised execution infrastructure that provides computational, networking and storage resources to the Small Cell.
- **VIM:** Manager of the HW and networking resources (lifecycle, provision, placement, operation) comprising a cluster of micro-servers, namely the Light DC, and the networking nodes and links (virtual and physical).

2 Light DC Architecture

SESAME envisages combining the MEC (Mobile Edge Computing) and NFV (Network Functions Virtualization) concepts with Small Cell virtualisation in 5G networks and enhancing them for supporting multi-tenancy. The purpose of the Light DC is to provide Cloud services within the network infrastructure and to offer facilities by promoting and assisting the exploitation of network resource information. To this end, all the normally hardware located modules of the Light DC will be delivered as resources using novel virtualisation techniques. Both networking and computing virtualisation extensions are developed by using open frameworks such as OPNFV[1]. The combination of the proposed Light DC architecture with the concepts of NFV and SDN will facilitate achieving higher levels of flexibility and scalability.

As seen in the detailed architecture in [1], the Light DC will be able to execute different Small Cell and Service VNFs under the control of the CESCM. In particular, the Light DC hosts the Small Cell Common VNFs (SC-C-VNFs) which performs the fan-in fan-out of traffic towards EPC, the Small Cell VNF for 5G functional splitting, as well as the Service VNFs.

To support the use cases (UCs) of SESAME[2], the requirements imposed on responsiveness and performance require novel approaches to the deployed Light DC infrastructure, compared to current state-of- the-art (SOTA). As network conditions, traffic loads, and QoS/QoE constraints change rapidly, the CESCs on the network edge must be able to react to those dynamics. This requires, *for example*, a high responsiveness of the system regarding spinning up or shutting down VNF instances providing network functions on-demand. These requirements are exacerbated by the fact that at the network edge, cost-effectiveness is paramount, so device costs must be kept low, which practically limits their available resources. These devices typically come in the form of Single-Board Computers (SBCs). To adapt to the ever-shifting workloads and make efficient use of the available resources, virtualisation is a “key” solution.

However, due to the limit on available resources, providing a highly responsive system is especially challenging. While containers are a common lightweight approach to reduce overhead, they generally provide less isolation between the tenants of a system compared to virtual machines, which can be problematic in multi-tenant scenarios, such as in the cloud-*enabled* infrastructure, where isolation is “vital” to prevent information leakage and undesirable cross-effects.

To increase the performance of Virtual Machines (VMs), both the VMs themselves and the VM manager (hypervisor) have been investigated. This includes work into minimising the operating systems running as virtual machines, making them as lightweight as possible. Furthermore, techniques to increase the performance of the hypervisor itself, with special focus on virtualisation on single-board computers have been investigated and realised. Finally, especially in network-heavy scenarios as in SESAME, the virtualisation overhead of network communications is considerable, especially in the case of VNF chaining, where each step in the chain incurs a latency penalty. Within the scope of SESAME, acceleration techniques for high-

[1] Open Platform for NFV (OPNFV) facilitates the development and evolution of NFV components across various open source ecosystems. Through system level integration, deployment and testing, OPNFV creates a reference NFV platform to accelerate the transformation of enterprise and service provider networks. Participation is open to anyone, whether the interesting person is an employee of a member company or a third party intending to learn more about network transformation. For more details see: *https://www.opnfv.org/*

[2] For more details about the SESAME-*based* proposed use case also see: SESAME Deliverable D2.1: “*System Use Cases and Requirements*”. Available at: *http://www.sesame-h2020-5g-ppp.eu/Deliverables.aspx*

performance virtual networking and especially virtual switching between VMs have been investigated, realised, and tested for their efficiency.

3 Light DC PoC Physical Architecture

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 have been taken into account during the development of the ST Barcelona board, provided by STM[3]. However, for more demanding applications in terms of computing resources, the NXP[4] LS2085A[5] (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, lead to the decision to include the NVIDIA GPU M4000[6] in the micro-server for supporting the computing requirements needed in case of running video transcoding VNFs at the edge.

At the beginning of the SESAME project, the NVIDIA CUDA[7] developer's toolkit was supporting the software development on this device. The toolkit supported different operating systems (Linux, Mac OS and Windows) on different architectures. However, since version 7.5 NVIDIA[8] decided to stop supporting the vast majority of devices based on ARM architecture, which lead to the decision on including x86[9]-*based* nodes in the Light DC, thus making the computing platform heterogeneous.

The system proposed for the PoC provides the use of four different blocks acting as micro-servers:

1. STM board running a virtualization layer to host Common SC-VNF and SC-VNF (at least one).
2. Raspberry Pi 3[10] board.
3. NXP board running a virtualization layer to host SC-VNF, Service VNFs (SW only) and storage.
4. INTEL node (Xeon v3[11]) equipped with a NVIDIA GPU M4000 for VTU HW acceleration and storage.

[3] For more details see: *http://www.st.com/en/evaluation-tools.html*
[4] More details about NXP can be found at: *http://www.nxp.com/*
[5] More details about this specific NXP board can be found at: *http://www.nxp.com/products/microcontrollers-and-processors/arm-processors/qoriq-arm-processors/qoriq-ls2085a-rdb-reference-design-board:LS2085A-RDB*
[6] For more detailed information also see: *http://www.pny.com/nvidia-quadro-m4000*
[7] See: *https://developer.nvidia.com/cuda-toolkit*
[8] For more details also see: *https://developer.nvidia.com/cuda-75-downloads-archive*
[9] x86 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*
[10] For more details also see: *https://www.raspberrypi.org/products/raspberry-pi-3-model-b/*
[11] *https://software.intel.com/en-us/articles/intel-xeon-processor-e5-2600-v3-product-family-technical-overview*

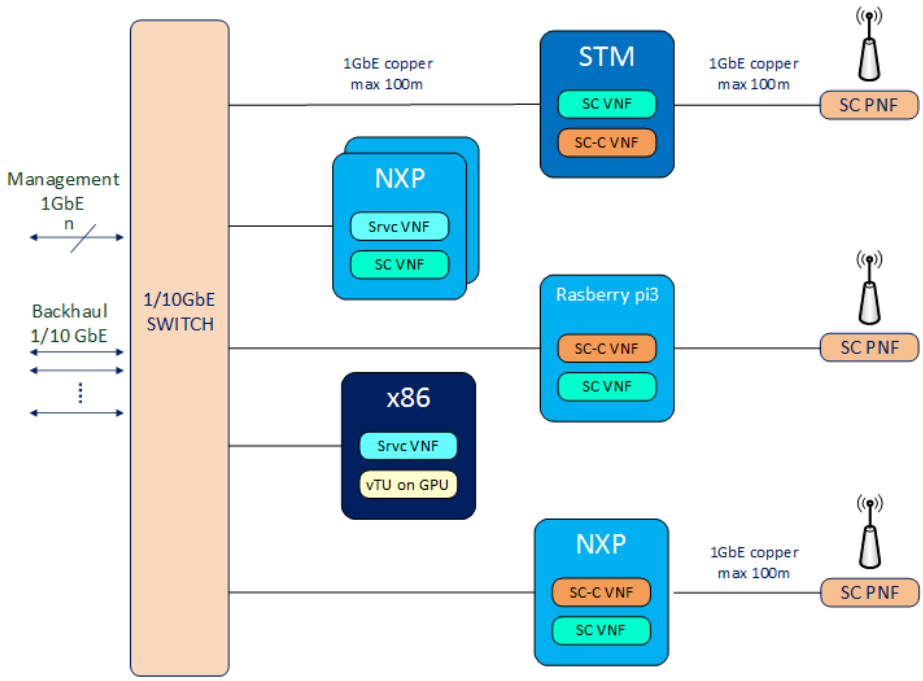


Figure 1: Light DC architecture for the PoC

Micro-server	Architecture	Cores	RAM	Storage	PCI-e[12] Acceleration
NXP LS2085A	ARMv8[13], A57[14]	8	16 GB	500 GB	no
GOMA[15] (FlexPAC)	Xeon E5-2630v3[16]	8	64 GB	4+2 TB	GPU
STM board	ARMv8, A53[17]	4+1	1 GB	xx GB (SATA[18] disk)	no
Raspberry pi 3	ARMv8, A53	4	1 GB	xx GB (microSD[19])	no

Table 1: Micro-server: Main characteristics

[12] For more relevant information also see, *for example*: https://en.wikipedia.org/wiki/PCI_Express
[13] 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
[14] For more details see: https://www.arm.com/products/processors/cortex-a/cortex-a57-processor.php
[15] For more details see: https://www.gomaelettronica.it/en/server-and-workstation-portables-high-density-storage-server-intel-i7-i5-i3-series
[16] http://ark.intel.com/products/83356/Intel-Xeon-Processor-E5-2630-v3-20M-Cache-2_40-GHz
[17] For more details see: https://www.arm.com/products/processors/cortex-a/cortex-a53-processor.php
[18] More related informative details can be found, for example, at: https://en.wikipedia.org/wiki/Serial_ATA
[19] Also see: https://en.wikipedia.org/wiki/Secure_Digital#Microhttps://en.wikipedia.org/wiki/Secure_Digital#Micro

3.1 NXP micro-server

The NXP based micro-server is a commercial evaluation board (based on NXP LS2085A SoC) that in SESAME PoC should host Small Cell VNFs and service VNFs.

Figure 2: NXP LS2085A evaluation board

Main features:

- LS2085A processor (with 8x64-bit up to 1.8 GHz ARM A57 cores[20])
- 16GB DDR4[21] system memory (in SESAME PoC) distributed on two 8GB DIMM modules[22] operating at 2.133GT/s (72-bit System Memory). Four DIMM slots available.
- 8GB DDR4 (one DIMM module) for datapath connected to one port of 40-bits DDR4 (including ECC[23]) up to 1.67GT/s
- Four RJ45 connectors[24] for 10/1GE support
- Four SFP+ cages[25] for XFI support[26]
- Two PCIe connectors supporting
- PCIe card (x4/x8 Gen3)
- PCIe card (x4 Gen3)
- Two SATA 3.0 connectors (one connected to a 500GB Hard disk)
- Two USB 3.0 ports
- One SD[27]/MMC[28] card slot

[20] For more details also see: *https://www.arm.com/products/processors/cortex-a/cortex-a57-processor.php*
[21] For more relevant information also see, *inter-alia*: *https://en.wikipedia.org/wiki/DDR4_SDRAM*
[22] *https://en.wikipedia.org/wiki/DIMM*
[23] For more information also see: *https://en.wikipedia.org/wiki/ECC_memory*
[24] See: *https://en.wikipedia.org/wiki/Registered_jack#RJ45*
[25] See: *https://en.wikipedia.org/wiki/Small_form-factor_pluggable_transceiver*
[26] See: *https://en.wikipedia.org/wiki/E-mu_20K* Also see: *https://en.wikipedia.org/wiki/Sound_Blaster_X-Fi#X-Fi_USB_products*
[27] See: *https://en.wikipedia.org/wiki/Secure_Digital*
[28] See: *https://en.wikipedia.org/wiki/MultiMediaCard*

- NOR/NAND flash interface[29]
- 64MB high speed flash with SPI interface[30].

More in detail, the following figure (Figure 3) depicts the block diagram of the LS2085A processor.

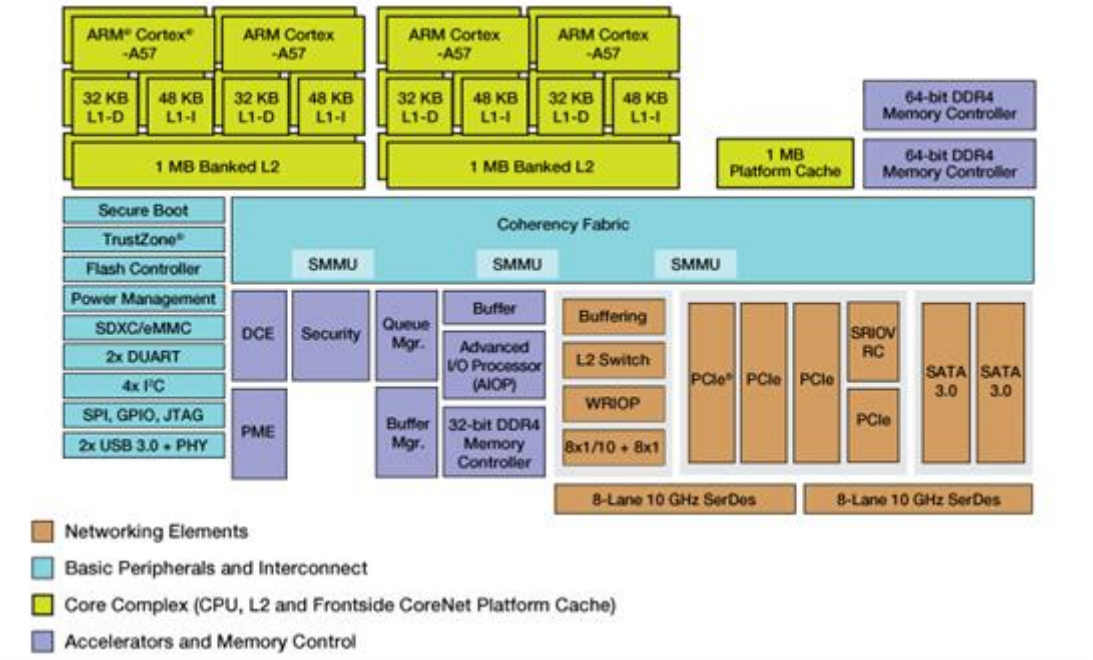


Figure 3: LS2085A block diagram

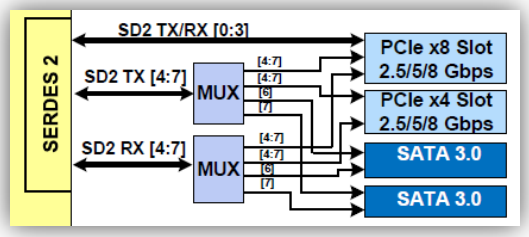
Limitations to consider in SESAME deployment:

PCIe Slots and SATA utilization constraints

Not all the interfaces are available at the same time, in particular using PCIe Slots and SATA disks you have to be aware that there are 3 possible configurations, that can be defined statically (before booting up the micro server) selecting two DIP-Switches[31] on the board:

1. One PCIe Slot x1/x4/x8 (NO SATA)
2. Two* PCIe Slots x1/x4 (NO SATA)
3. One PCIe Slot x1/x4 + Two SATA

* if you need to use a riser is not possible to use two PCIe Slots



PCIe Add-in Cards Form Factor

Low Profile cards (half height, half-length, max. 25W)

[29] For more related information also see: *https://en.wikipedia.org/wiki/Flash_memory*
[30] See: *http://www.acmeportable.com/products/flexpac*
[31] For more details see, *inter-alia*: *https://en.wikipedia.org/wiki/DIP_switch*

Mechanical constraints

Being a 1U micro-server, you are forced to plug a riser card in the x8 PCIe Slot and plug the PCIe Add-in card in the riser. Only one PCIe Card can be plugged in the riser.

Micro-server Power Supply

The external ATX12V[32]/EPS12V PS is a standard primary power supply (PC ATX-PS < 300W) with:

- Vin = 90 - 264Vac
- Temperature Range Operating = 0° ~ 50°C on full load
- Relative Humidity = 20~ 80%

[32] For more information about Advance Technology Extender (ATX) see: *https://en.wikipedia.org/wiki/ATX*

3.2 Intel micro-server

The INTEL based micro server is a commercial platform (GOMA[33] FlexPAC[34] Industrial portable workstation) that in SESAME PoC should host service VNFs and storage.

Figure 4: Intel based micro-server

Main features (platform used for the SESAME PoC):

The FlexPAC Industrial portable workstation is configured with:

- Aluminium Chassis including 17.3" LCD Display (Resolution 1920*1080)
- US Keyboard with integrated touchpad

CPU:

- CPU Intel **Xeon** E5-2630v3 2.4GHz, 8 Core, **16 Threads** (CPU 2 not present)
- **64 GB DDR4** RAM (max. 256 GB)
- Intel® C612 chipset[35]
- Liquid cooling for CPU Heatsink

Free Slots:

- 2x PCI-E 3.0 x16, (one reserved for **GPU**, see related section)
- 1x PCI-E 3.0 x8
- 1x PCI-E 3.0 x4
- 1x PCI-E 2.0 x4 (When CPU 2 is installed)

I/O:

[33] For more details see: *http://www.gomaelettronica.it/it/*
[34] For more details see: *http://www.acmeportable.com/products/flexpac*
[35] For more details see: *https://ark.intel.com/products/81759/Intel-C612-Chipset*

- 10x SATA3 (internal),
- 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

STORAGE:

- Two SATA disks with 2TB and 4TB capacity
- Dimension ca. 433 x 347 x 229 mm
- ATX Power Supply 650W
- Caddy Trolley included
- Weight: around 15Kg

One PCI-E 3.0 x16 slot is equipped with an NVIDIA® QUADRO® M4000 GPU.

NVIDIA® QUADRO® M4000, accelerated by NVIDIA Maxwell™ GPU architecture:

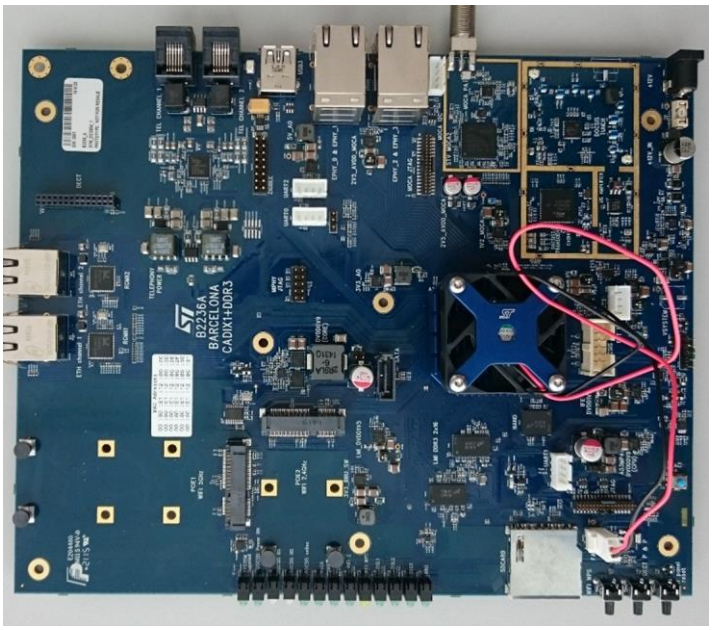
Figure 5: NVIDIA QUADRO M4000

GPU Memory	8 GB GDDR5
Memory Interface	256-bit
Memory Bandwidth	192 GB/s
NVIDIA CUDA® Cores	1664
System Interface	PCI Express 3.0 x16
Max Power Consumption	120 W
Thermal Solution	Active
Form Factor	4.4” H × 9.5” L (Single Slot, Full Height)
Display Connectors	4x DP 1.2
Max Simultaneous Displays	4 direct, 4 DP 1.2 Multi-Stream
Max DP 1.2 Resolution	4096 × 2160 at 60 Hz
Max VGA Resolution	2048 × 1536 at 85 Hz
Graphics APIs	Shader Model 5.0, OpenGL 4.54 DirectX 12.05 Vulkan 1.04
Compute APIs	CUDA,

DirectCompute,
OpenCL™

3.3 STM micro-server

The micro server based on the ST SoC is implemented using the evaluation board provided by ST and it is shown in Figure 6.



- 4 + (1) A53 cores 15K Dmips
- DDR : 32-bit DDR3 – 1GB
- Ethernet : 4x Gigabit Ethernet
- 2 mPCIe slots
- SATA: HDD device supported
- SPI NOR 16MB
- NAND Flash 128MB
- eMMC
- USB: 1 USB 3.0 DRD mode

Figure 6: STM evaluation board

3.3.1 ST SoC features

The ST Light DC SoC implements hardware-based full virtualization support with ARM extensions and System MMU complemented by ST hardened security mechanisms that ensures a complete decoupling of VNFs from the computing, storage and networking resources.

The ST Light DC SoC includes:

- Four 64-bit ARM® Cortex™-A53 cores offering up to 15 K DMIPS and supporting cache coherency;
    - Quad-core ARM Cortex-A53 up to 1.3 GHz, 1 Mbyte L2 cache;
- Support for ARM virtualization and cryptographic extensions;
- Support for of NAS applications;
- Comprehensive integrated networking solution;
    - Hardware-accelerated L3/L4 router and L2 switch, combined with ARM Cortex-A53 processing, up to 16 Gb/s networking and 4096 accelerated sessions;
    - Rich number of Ethernet interfaces:

  - 3 x RGMII ports (including one LAN/WAN port to may be used for backhaul);
  - 4 x Gigabit Ethernet ports with integrated Ethernet PHYs (including one LAN/WAN port to may be used for backhaul);
  - switching/routing line-rate throughput on every port;
- PCIe acceleration;
- Supports I/O coherency with high-speed I/O interfaces (PCIe and Ethernet);
- One 32-bit LMI[36] supporting DDR4 up to 2400 Mbit/s, DDR3 up to 2133 Mbit/s;
- memory obfuscation support;
- Flash memory interfaces: SLC NAND, Serial NOR and two eMMCs (one used for SD card);
- Secure boot;
- Hardware-accelerated cryptographic engine (AES, HASH) and security hardening;
- Connectivity;
  - 2 x PCIe
  - 1 x SATA
  - Optional third PCIe multiplexed on SATA port
  - Optional SGMII port multiplexed on SATA port
  - 1 x USB 3.0
- 28 nm FD-SOI silicon technology process for optimum power/performance ratio;
- Package: FCBGA 27 x 27.

In order to reduce power consumption, the ST Light DC SoC implements the following power management features:

- Cortex-A53-MP4 DVFS;
- Dedicated power island for Cortex-A53-MP4[37];
- Clock gating on various hardware subsystems with no leakage, owing to FD-SOI silicon technology;
- High network availability mode support with network wake-up triggers;
- An ARM® Cortex™-M4 processor is integrated:
  - To manage watchdog functions.
  - To control wake-up of the ARM® Cortex™-A53-MP4.

[36] See: *https://en.wikipedia.org/wiki/Lisp_Machines*

[37] For more details see, for example: *https://www.arm.com/files/event/A2_Advanced_Implementing_ARM_Cortex-A57_and_Cortex-A53_based_big.LITTLE_SoCs_in_16nm_FinFET_Technology.pdf*

3.3.2 ST evaluation board SW

Regarding the software, ST started with a custom porting of Linux kernel 3.10[38], followed by a porting of the Linux kernel 4.1[39] to be compliant with the requirements of SESAME.

Initially the Linux kernel was mounting a custom filesystem from ST exported via NFS. Since the SoC includes a CORTEX-A53 processor with ARMv8 architecture, the ARM TRUSTED FIRMWARE (ATF[40]) has been required for the boot. Both the ATF and the kernel are compiled using a 64bit toolchain. After that the ST-ATF, the kernel image and the DTB are loaded in DDR and gdb start the execution from the ST-ATF entrypoint; at the end of the ST-ATF the execution jumps to the kernel and executes it.

Since the ST Filesystem was when trying to update and install standard packages using utilities like yum or apt-get, ST decided to pass to a standard Ubuntu 16.04 base file system[41]. To increase portability ST also decided to store this file system on an SD card and mount it from there instead of from the network.

The file system has been configured so as to have two users (Ubuntu and root) and the most useful packages have been installed. To enable the use of KVM, the bootloader has been modified to increase the security level after boot to EL2[42] (Hypervisor) and the KVM kernel options have been enabled. Qemu[43] has been installed installed; thanks to it we are capable to "run" virtual machines. In addition, the Docker has been installed by using pre-compiled docker binaries. All the required essential kernel options for the docker daemon execution have been enabled and the daemon runs correctly.

[38] See: *https://kernelnewbies.org/Linux_3.10*
[39] See: *https://kernelnewbies.org/Linux_4.1*
[40] *http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0928e/CJHIDGJF.html*
[41] See: *https://ubuntuforums.org/showthread.php?t=2326404*
[42] See: *http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0488d/CIHJHAAG.html*
[43] For more details see: http://www.qemu.org/

3.4 Rasberry Pi 3 model B microserver

The Raspberry Pi 3 is the third generation Raspberry Pi, as shown in Figure 7.

Figure 7: Raspberry Pi 3 Model B

It replaced the Raspberry Pi 2 Model B in *February 2016*. Compared to the Raspberry Pi 2 it has:

- A 1.2GHz 64-bit quad-core ARMv8 CPU, Cortex™-A53
- 802.11n Wireless LAN
- Bluetooth 4.1

Like the Pi 2, it also has:

- 1GB RAM
- USB ports
- 40 GPIO pins
- Full HDMI port
- Ethernet port
- Combined 3.5mm audio jack and composite video
- Camera interface (CSI[44])
- Display interface (DSI[45])
- Micro SD card slot

[44] More informative details can be found at: *https://en.wikipedia.org/wiki/Camera_Serial_Interface*
[45] More informative details can be found at: *https://en.wikipedia.org/wiki/Display_Serial_Interface*

- Video Core IV 3D graphics core

The Raspberry Pi 3 has an identical form factor to the previous Pi 2[46] (and Pi 1 Model B+[47]) and has complete compatibility with Raspberry Pi 1 and 2.

It can be used to implement a very low cost microserver node as development platform to be used for testing lightweight virtualization solutions, especially for the case of virtualization of small cell- *related* functions.

[46] See: *https://www.raspberrypi.org/products/raspberry-pi-2-model-b/*
[47] See: *https://www.raspberrypi.org/products/raspberry-pi-1-model-b/*

3.5 Interfaces and Networking

As stated in [1], like any data centre, the SESAME Light DC is a pool of resources (computational, storage, network) interconnected, by using a communication network. Of course, the network topology plays a pivotal role on determining the scalability, robustness, performance (e.g. throughput and latency), efficiency and, in simple words, the viability of Light DC. In [1] we saw that any famous topologies such as three-tier, fat tree, DCell, etc., might be adopted for SESAME Light DC, depending on the respective use case and the targeted scenarios. In this section, we will focus upon the nature of interfaces and networking elements used to "form" such topologies.

In principle, there exist several options, among wired and wireless technologies, which can be used for the networking purposes. In general, wired technologies include solutions such as:

- DSL variants, such as ADSL and SHDSL.
- PDH and ADH/SONET interfaces, such as (fractional) E1/T1, E3, T3, STM-1/OC-3, etc.
- Ethernet over copper or optical fibre.

Besides, the wireless technologies (over the licensed or unlicensed spectrum) can be classified as follows:

- Point-to-point microwave radio relay transmission (terrestrial or, in some cases, by satellite);
- Point-to-multipoint microwave-access technologies, such as LMDS, Wi-Fi, WiMAX, etc.;
- Free space optics (FSO).

Depending on the use case and the real market needs, the adapted solution to form light DC may vary from a wired solution (e.g. in a noisy area with lot of radio interference or a place with long distance communications) to a wireless solution.

For the SESAME PoC, some potential wired/wireless solutions are discussed below. More details about the selected solution over the actual SESAME PoC testbed will be presented on the future deliverables of WP7.

The SESAME wired solution candidate will be deployed over the OTE testbed. This testbed consists of three nodes: Control, Compute and Neutron, [10], as part of the OpenStack infrastructure. The fourth node is a PC with four network cards, each connected to the OpenStack nodes and the fourth connected to the management network. This node is used as physical switch and it has also installed OpenvSwitch[48] (OVS) to act as SDN switch.

By installing OpenvSwitch on all nodes in the testbed, we make sure that they are all connected to the switch and SDN-*enabled* via the OVS virtual environment. For that, we used a dedicated data network for all the OVS nodes, assigning a second interface from the nodes as a port to the OVS bridge. This was the essential physical interconnection and OVS configuration in order to obtain a starting environment for the integration OpenStack-ODL.

SESAME's wireless backhaul uses radio transceivers that follow the IEEE 802.11 standard[49] to connect SCs to form a mesh network that serves to carry the S1 tunnels. In order to satisfy the needs of high capacity connections between the SCs of the network, IEEE 802.11ac is used. This specification offers up to several hundred Mbit/s of maximum throughput and can reach more than 1Gbit/s when using MIMO (up to four spatial streams).

[48] For more details see: *http://openvswitch.org/*
[49] For more details see, *inter-alia*: *https://en.wikipedia.org/wiki/IEEE_802.11*

From cheap and high performance wireless transceivers available on the market that support this specification, the Compex WLE900VX[50] transceivers have been chosen. They offer a cheap and flexible solution, as they are connected to a PCIe bus, a standard that is supported by a multitude of hardware platforms. Tests performed in indoor and outdoor environments yielded throughput of up to approximately 350 Mbit/s per wireless link. In order to improve the stability and yield of the wireless links, as well as to reduce cross-channel interference, the use of directive antennas is advised.

[50] *https://www.compex.com.sg/product/wle900vx/*

3.6 Storage

To deliver upon the needs and requirements that the SESAME system poses to storage, a storage system has been developed that allows local storage on the CESC and a remote storage on the CESCM that is available from any CESC. The keystone to the storage system is the fact that OpenStack is the technological foundation inside the SESAME architecture. This allows the use of Cinder [11] , OpenStack's own block storage system. By using Cinder and leveraging it (using filters, scheduler hints and running the Cinder volume service on each CESC), the system is given the mature functionality. The Cinder scheduler, on the CESCM, together with scheduling hints, filters and different volume types (LVM[51] and Hera) enables the VNF developer to place storage volumes that are needed by the VNFs on the correct location. This gives them the ability to use both local storage (LVM, [8]) on the CESC and remote storage (Hera). The way how Hera integrates into Cinder is by having developed a Cinder driver for Hera and the correct deployment configuration. This enables the lifecycle management of volume types through Cinder.

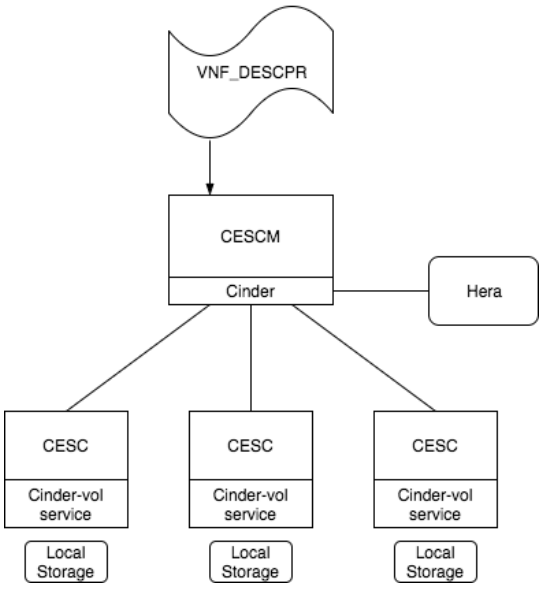


Figure 8: Cinder inside the SESAME architecture

[51] For more information on LVM see: https://www.howtoforge.com/linux_lvm

3.6.1 Hera

Hera is a distributed storage system with ZFS[52] being the base of the lowest component. Hera is managed centrally while the storage is distributed. The connected nodes, which are going to store data on the system, will expose their disks over iSCSI[53] to the Hera frontend node. On the frontend node, these disks from the backend are pooled as ZFS zpools. With the different raidz levels[54] that ZFS offers, the provider is able to create a different redundancy- and efficiency-level per pool. By spanning pools over multiple nodes the storage is more resilient to data loss, it can allow, in some configurations, up to three nodes to die without any data loss (for more in depth analysis see 3.6.3.

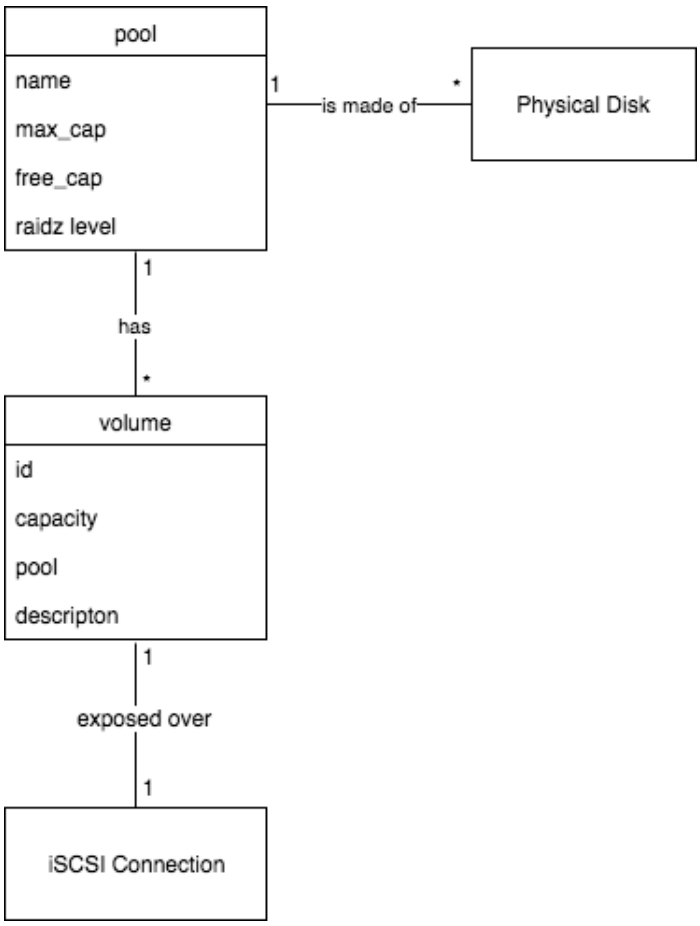


Figure 9: Data model of the storage component in Hera

A pool on the frontend is made up from multiple disks that are connected from the frontend. A pool inside Hera is a zpool that has its own name and information about the maximum capacity and the free capacity on it, as well as its own raidz level. Volumes are created on the pools, while a volume is only on one pool but the pool can store as many volumes as there is storage

[52] For more information on ZFS see: *http://docs.oracle.com/cd/E19253-01/819-5461/zfsover-2/*
[53] For more information on iSCSI see: *http://searchstorage.techtarget.com/definition/iSCSI*
[54] More information on raidz levels: *https://calomel.org/zfs_raid_speed_capacity.html*.
Also see: *https://en.wikipedia.org/wiki/Non-standard_RAID_levels#RAID-Z*

capacity left. Each volume can be attached and therefore is exposed over iSCSI and has an iSCSI connection URL. The Hera frontend can have multiple zpools of different configurations to allow different levels of resiliency and storage efficiency.

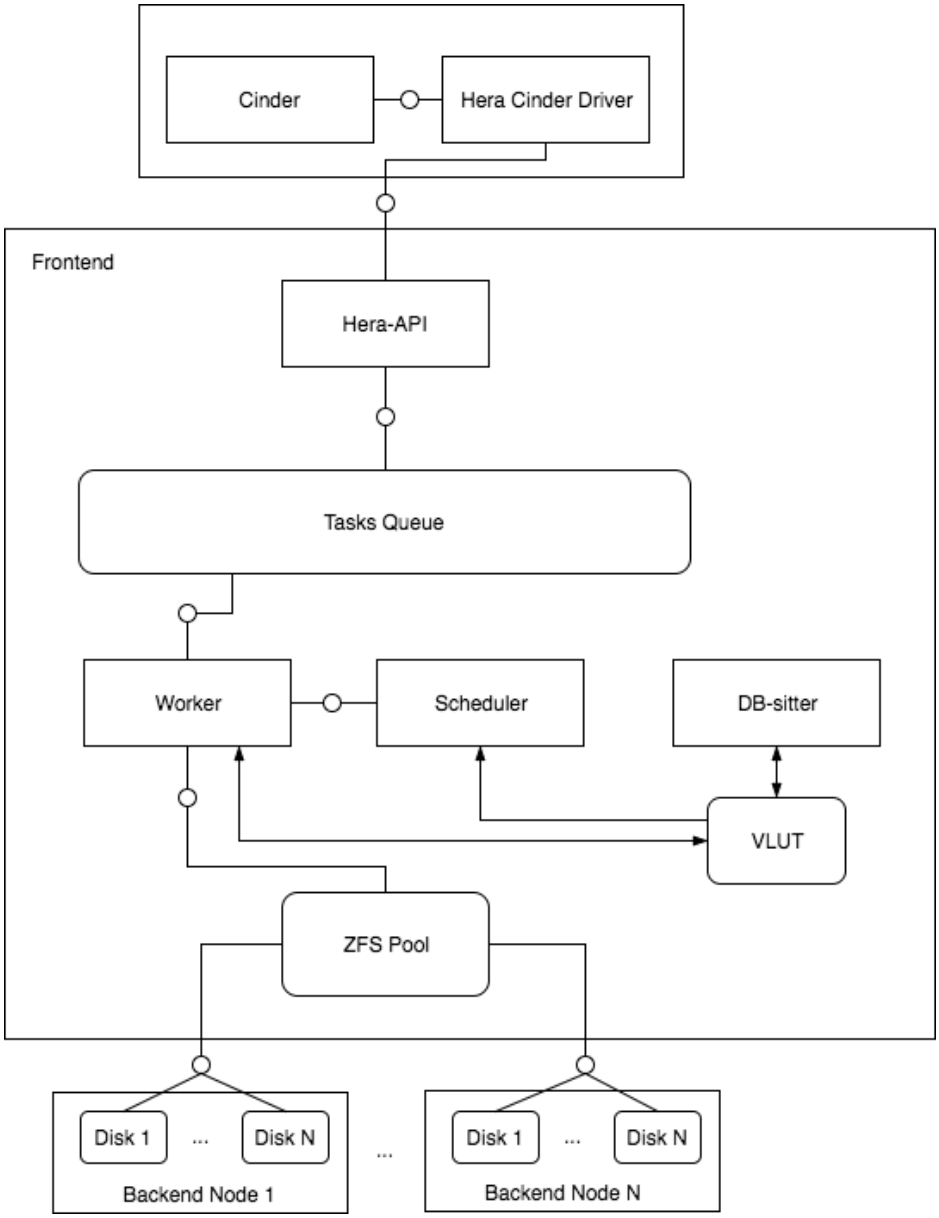


Figure 10: Overview of the Hera architecture, using FMC notation

- **Cinder**: "Cinder is a Block Storage service for OpenStack. It is designed to present storage resources to end-users that can be consumed by the OpenStack Compute Project (Nova, [9])."[55]

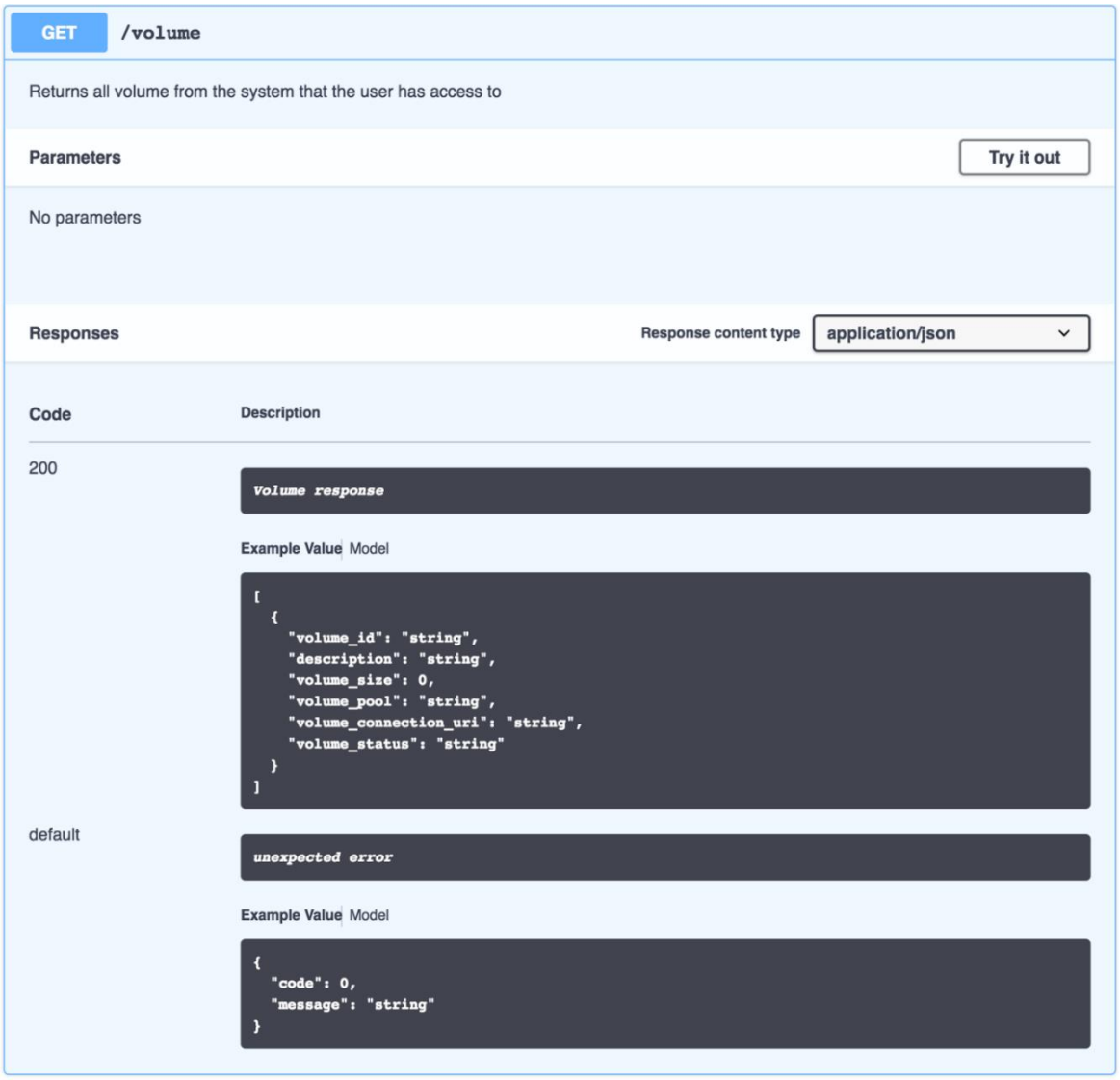
[55] *https://wiki.openstack.org/wiki/Cinder*

- **Hera Cinder Driver**: The storage driver for the Hera system, so that Hera can be used through Cinder. Essentially adds Hera as a backend to Cinder.
- **Hera API**: The restful API of Hera, that receives requests from either its Cinder driver or through curl commands and passes them on to the Task Queue.
- **Task Queue**: AMQP[56] task queue.
- **Worker**: A Celery[57] worker that is mainly responsible for the execution of commands directly on the storage.
- **Hera Scheduler**: The Hera scheduler receives requests whenever a command has to be executed on a volume or a pool. It is responsible to return the location of volumes (the pool on which a volume is) or to choose a pool for a new volume.
- **DB-sitter[58]**: On start-up, the DB-sitter initially checks the system for any available zpools and keeps track of them in the VLUT. It periodically checks if new pools have been added or pools have been removed.
- **VLUT**: The Volume Look-up table (VLUT) keeps track of the Hera volumes and the ZFS pools that are in the Hera system. The VLUT has a table for pools in which their name, storage capacity and free capacity is stored and a table for volumes, that keeps track of their location, size and description. The DB-sitter also checks for inconsistency in the VLUT and fixes these if necessary.
- **ZFS Pool**: On the frontend of Hera the available disks from the backend are pooled into zpools to ensure the resiliency and possibility of restoration of data.
- **Disks**: The backend nodes of Hera expose their disks over an iSCSI connection to the frontend. These disks are the actual physical storage of the Hera system.

[56] See: *https://www.rabbitmq.com/protocol.html*
[57] For more details see: *http://www.celeryproject.org/*
[58] Also see: *http://www.academia.edu/18403648/DBSitter_An_Intelligent_Tool_for_Database_Administration*

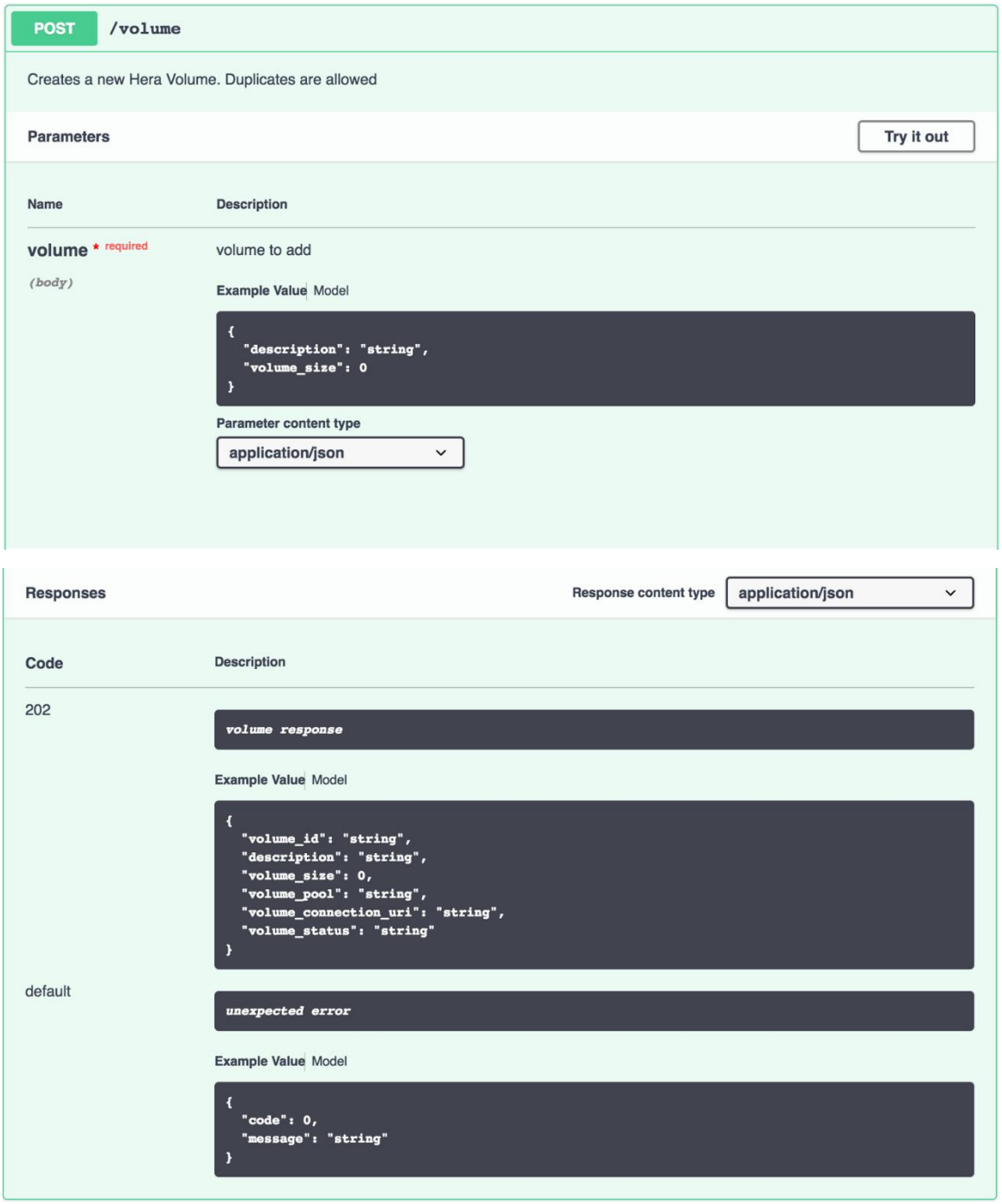
3.6.1.1 Hera-API

For Hera a restful API has been implemented with Swagger[59], since it offers a standardised model and format for RESTful APIs and forces the API to be consistently implemented. Following you can see all the commands that are accepted by the API with a brief explanation to each of them.

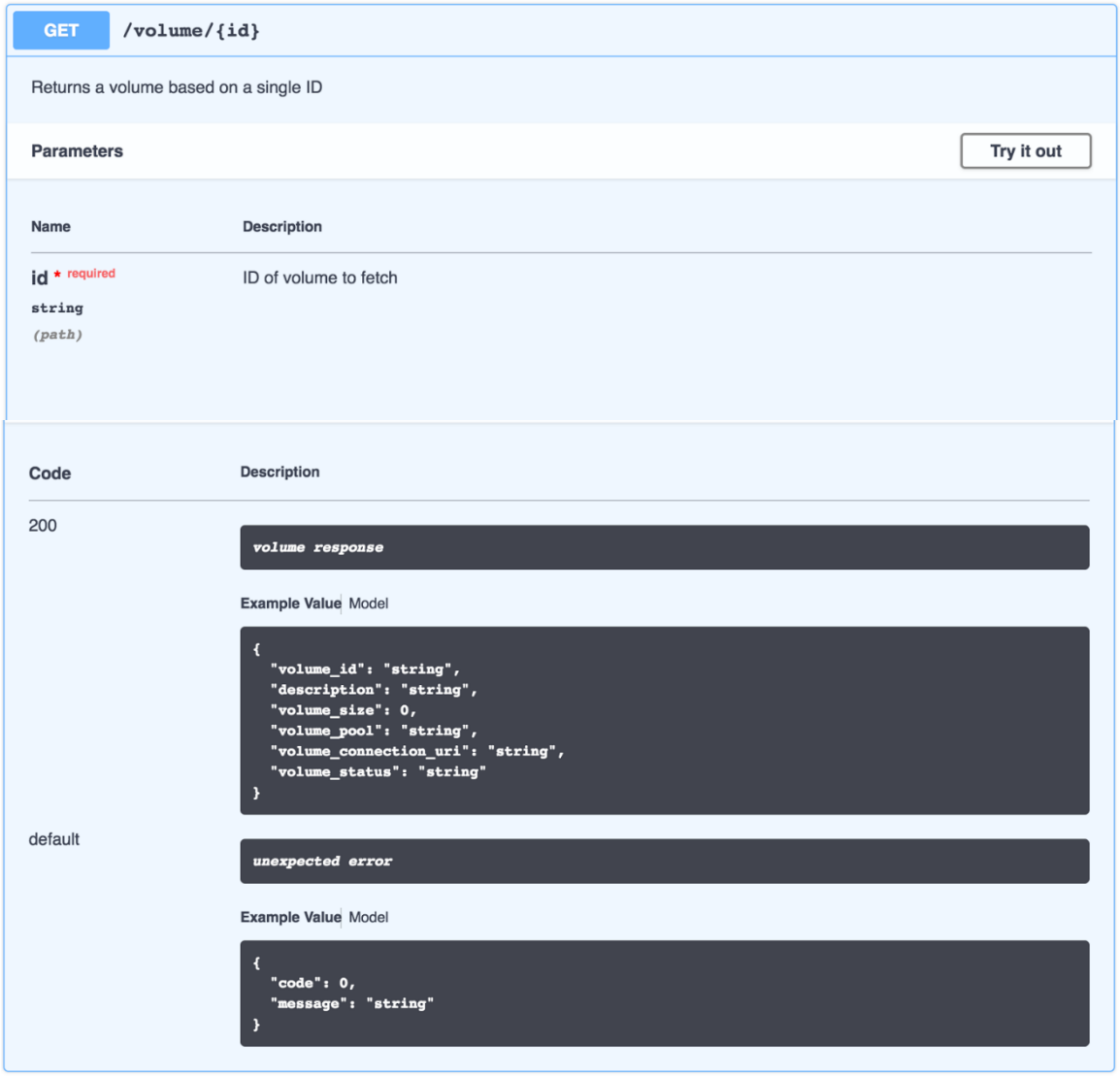


The get on /volume is a simple command to receive a list of all block volumes that are managed by the Hera system. The request will be sent from the API to the worker, which then will return a list of the pools with their id and their storage capacity.

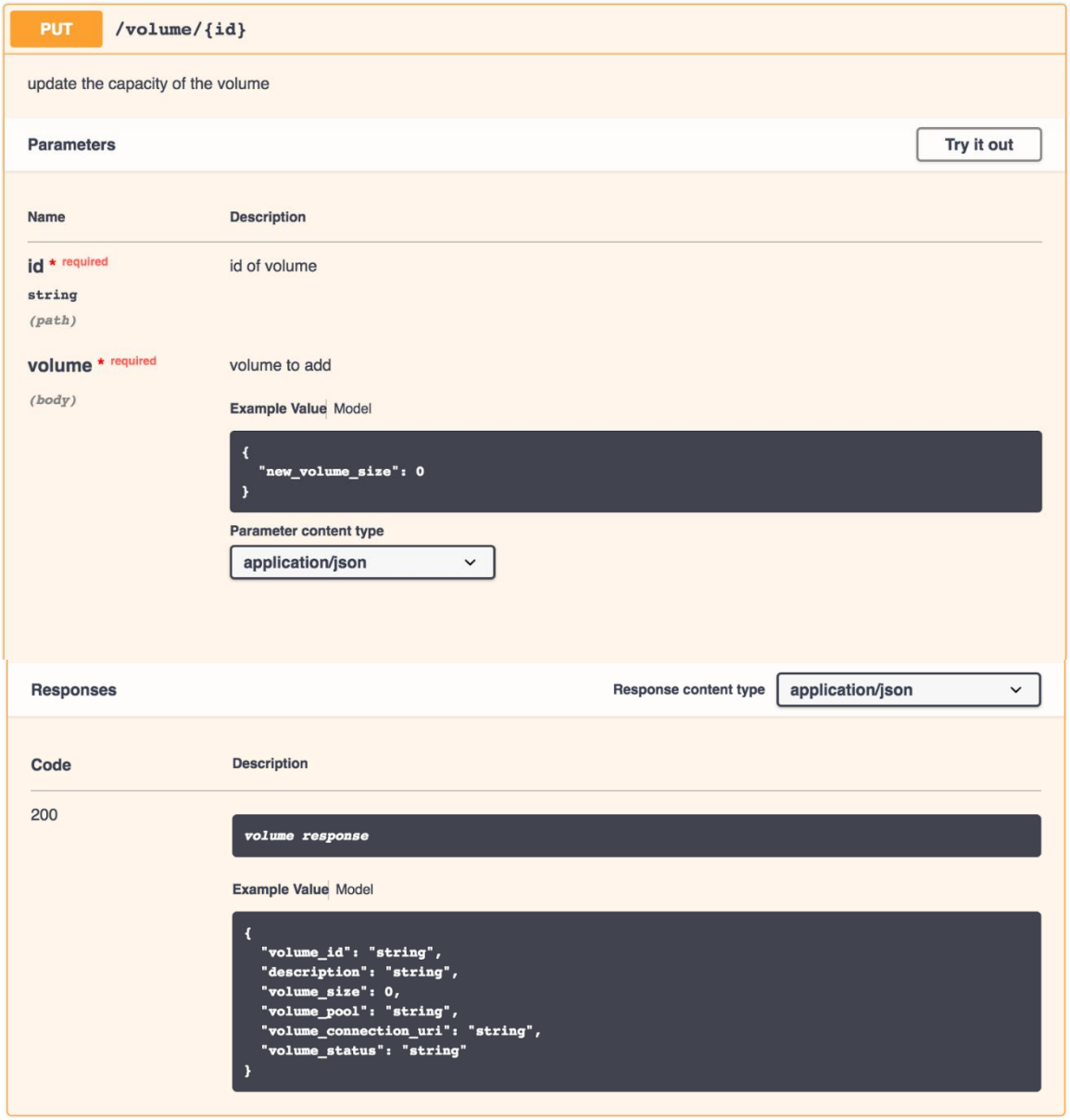
[59] http://swagger.io/



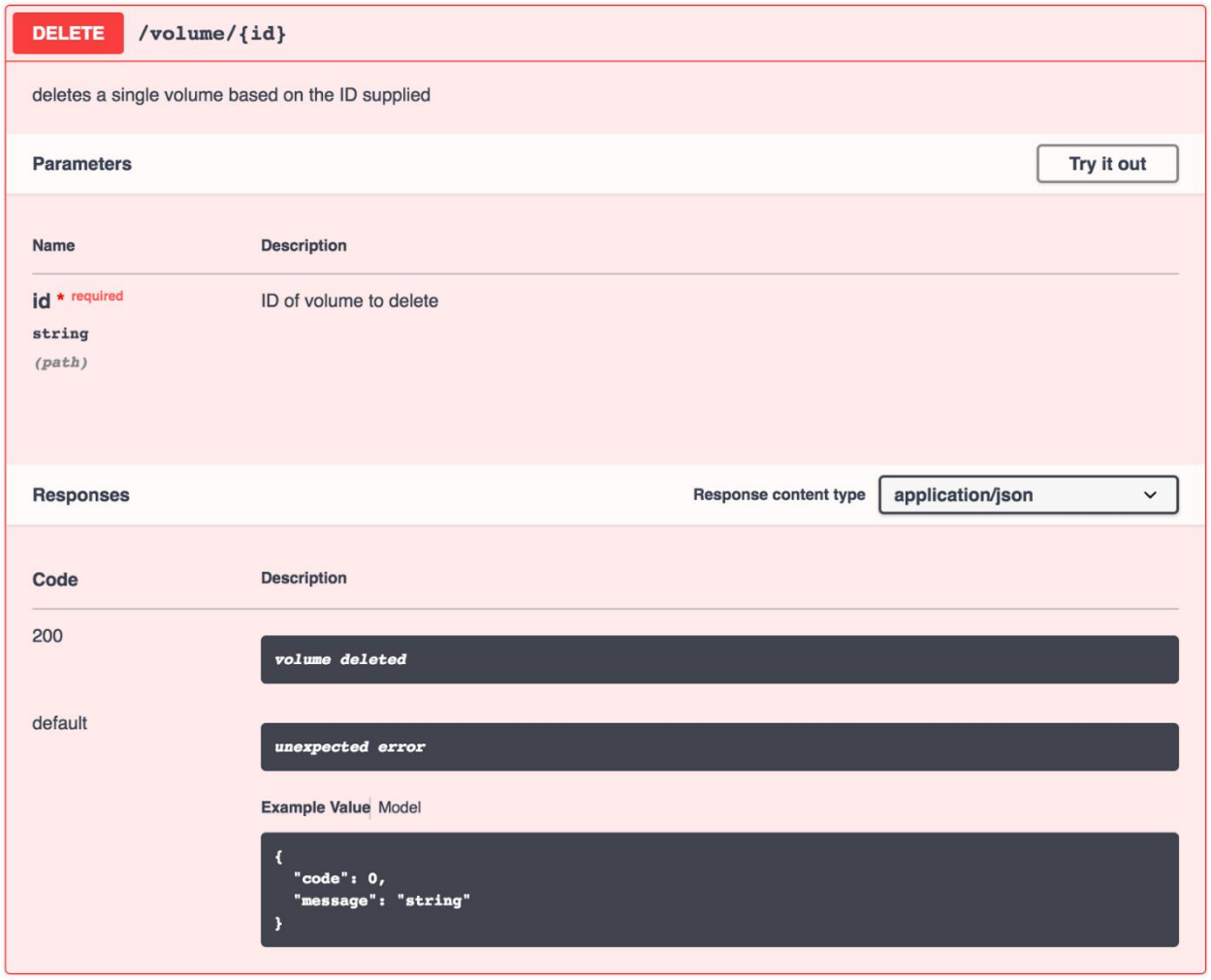
To create a new volume one has to use a post on '/volume' together with a JSON object containing the required parameters. The parameters that need to be passed for the creation of a volume are the size of the storage and a description for the model. After a successful creation the API will return an iSCSI URL to have an attachable volume.



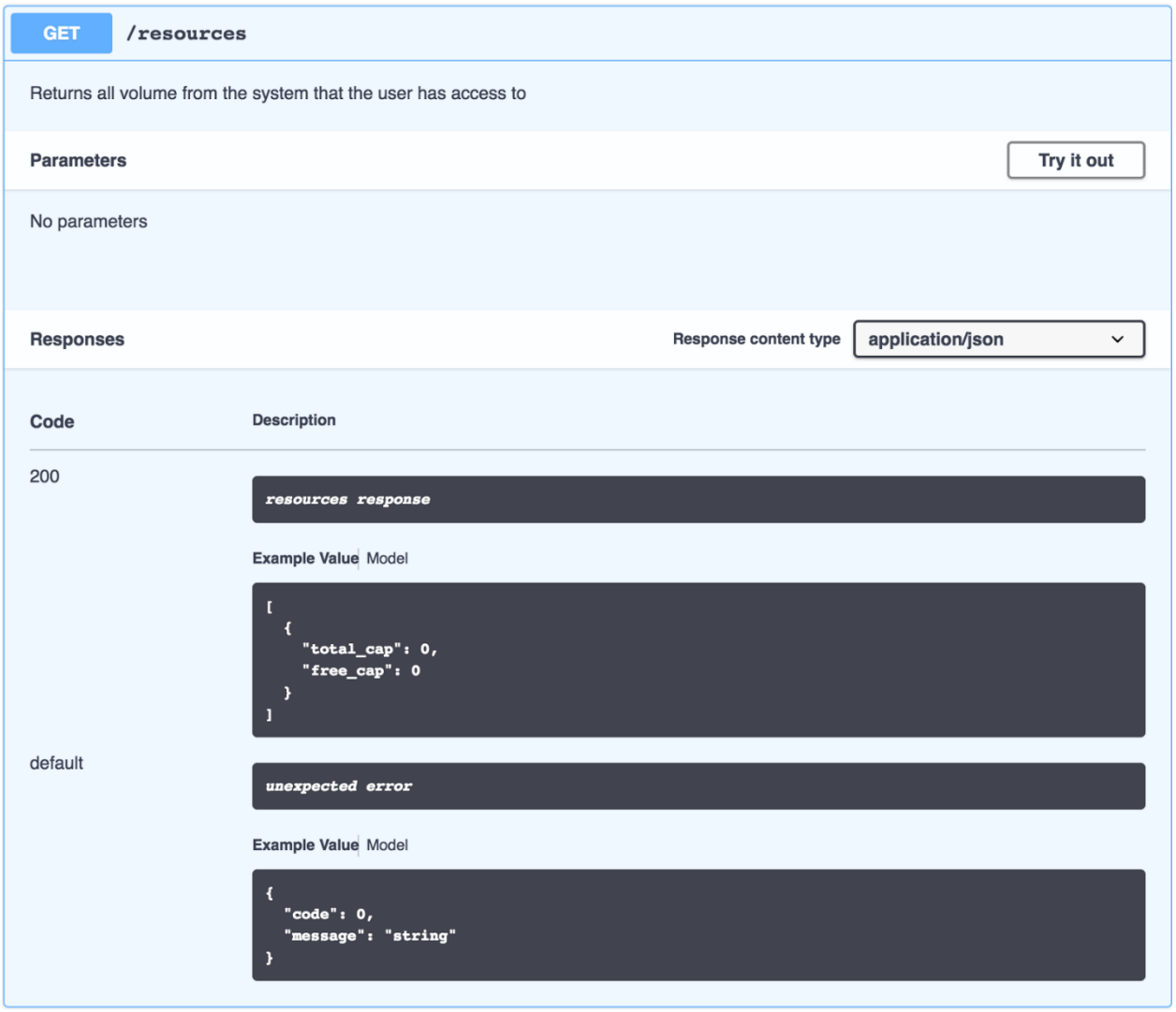
To receive the information of a single volume, one can do so by adding the id of the requested volume at the end of the URL. This will return the storage capacity and information about in which pool the volume is.



To resize a volume through the Hera API a put on said object has to be called with a JSON object containing the new size for the volume. The new size has to be bigger than the old one, because to achieve the resizing in ZFS provisioning has to be used instead of directly allocating the requested storage capacity. Through this API call, the provisioned space is going to be updated.



The delete on a specific volume removes the volume from the Hera system.



The Cinder driver is responsible to give current data to the Cinder scheduler so the scheduler is constantly updated and knows the resources available by all of the backends configured in Cinder. The API has an option for the driver to request the resources of Hera, which will return the free storage capacity still left on Hera and the maximum capacity.

3.6.1.2 Cinder driver

As SESAME has decided to use OpenStack, the decision to use Cinder as the provider for block storage for the system is rather natural. Cinder is already a main component of OpenStack and provides block storage for the system; additionally Cinder is compatible with multiple storage backends. Any storage system can be used with Cinder, the only thing needed is to implement a Cinder driver for said Backend. The Cinder drivers typically act as proxies between it and the storage Backend, thus, makes Cinder compatible with most storage systems as well as making it able to support different Backends at the same time.

In the case of integrating Hera in Cinder and, therefore, integrating it into the world of OpenStack, a driver was implemented to communicate with the Hera API. The driver is required to make calls to the Hera API for the respective operations issued by Cinder.

The Cinder scheduler also needs to be kept updated about the resources available in the backend, so the scheduler knows when no more space is left or the backend is down. This is implemented in the driver by calling a get on the resources and the API will return the necessary Information to the driver. The scheduler will request this information from the driver periodically, and therefore is always current.

The following figures (from Figure 11 to Figure 14) shows the interaction of Cinder and Hera through the driver.

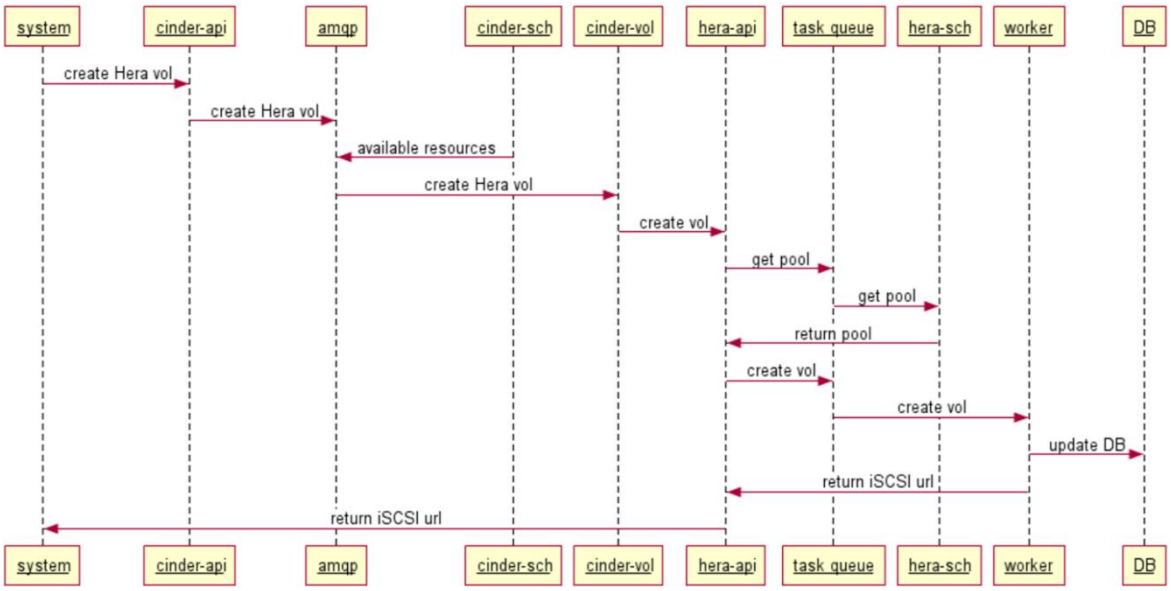


Figure 11: Create a volume

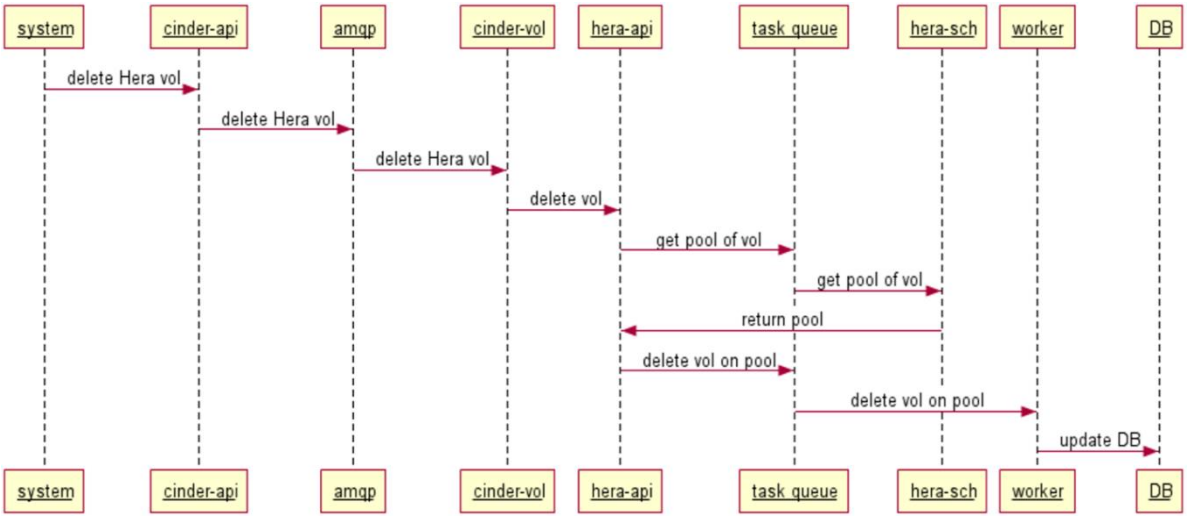


Figure 12: Delete a volume

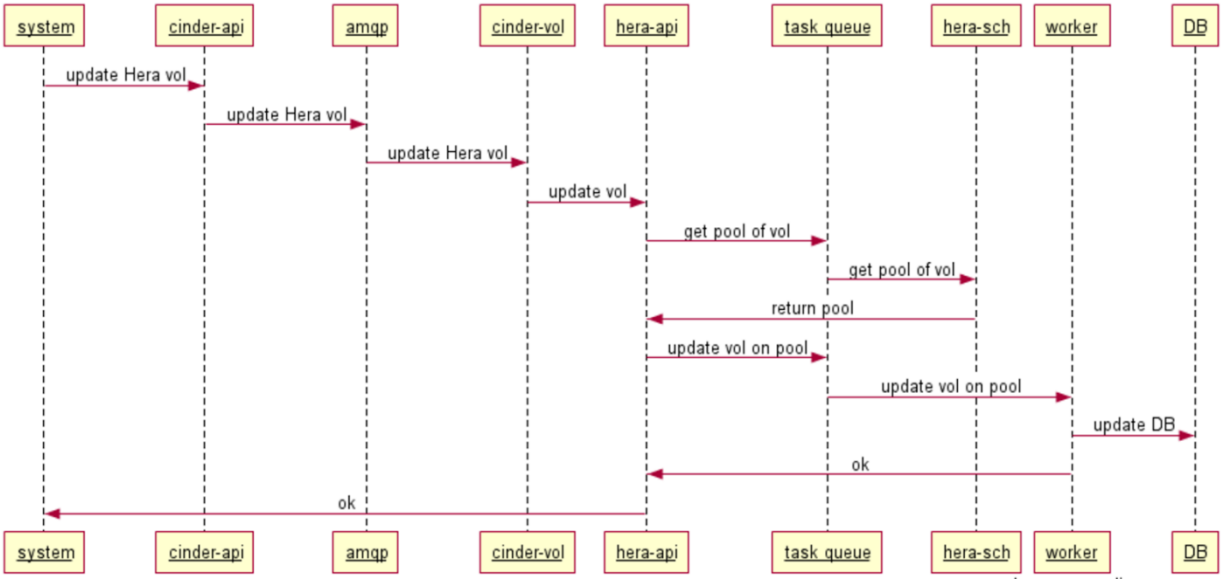


Figure 13: Update a volume

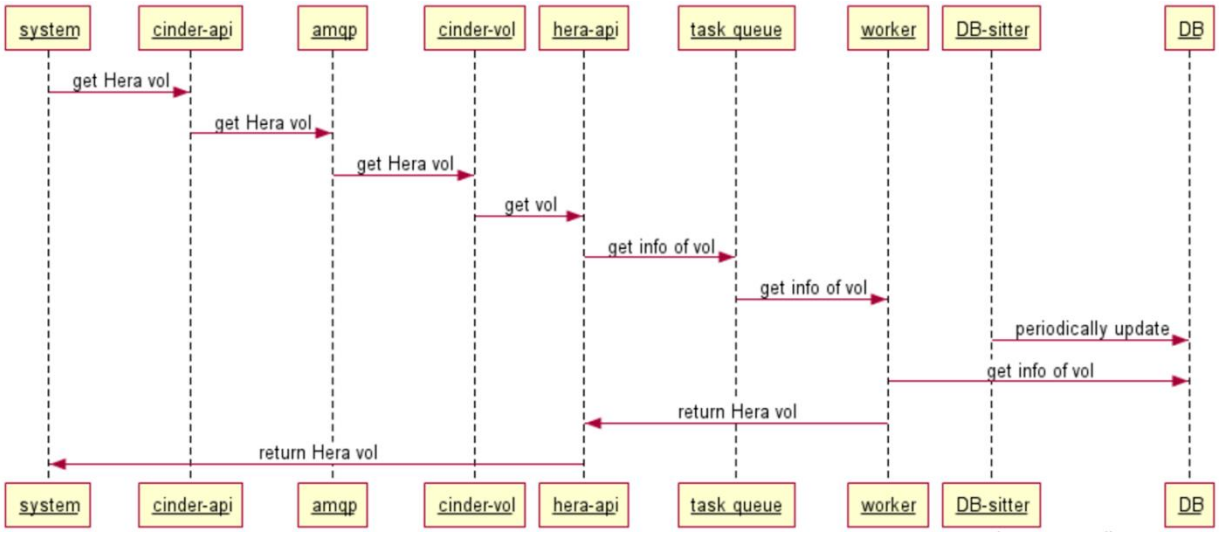


Figure 14: Get a volume

This integration of Hera into Cinder allows Hera to be seamlessly integrated with OpenStack and, therefore, "fits well" into the Architecture for SESAME.

The driver has been implemented as a normal cinder volume driver. There are also other different options to implement it, for example as an iSCSI driver or a SAN driver, but the normal volume driver suited the needs for the Hera driver, since it works as a proxy between Cinder and Hera. The driver informs the Cinder-scheduler about the available resources on the backend, which can be updated through the API. The driver can be configured, so that a specified IP/Port is given on which the Hera API is listening, so that the driver talks to the right Hera instance. The driver only needs to be added to the directory in which the Cinder drivers are, configured so that the API-endpoint is correct and activated by Cinder and it is ready to use through Cinder.

To see more detailed information on how to implement a Cinder driver, please see 3.6.5.

During the implementation, we ran into issues with the integration of the driver inside Cinder because the documentation is lacking. OpenStack very well describes the requirements to a driver, but when it comes to integrating the driver so it can work with Cinder, the documentation is lacking explanation.

3.6.2 Placement

Because Hera is placed in the CESCM and runs in the Light DC -and it is therefore placed in the central hub of the SESAME architecture- placement of Hera itself is no real issue. Hera storage clusters are available on the CESCM and it is a more reliable, redundant storage, but brings the penalty of being less performant (given that it is remote storage) than the local storage provided on the CESCs.

Placement of storage for VNFs is completely up to the developer of the VNF. When deploying a VNF the VNF can either use fast, small, local storage on the CESC where it is deployed or can instead use slower, bigger but more reliable Hera storage which is running on the CESCM. Both storage solutions can be used by the same VNF, e.g. a VNF aggregates data on the local disk and pushes it to Hera in a periodic timeframe.

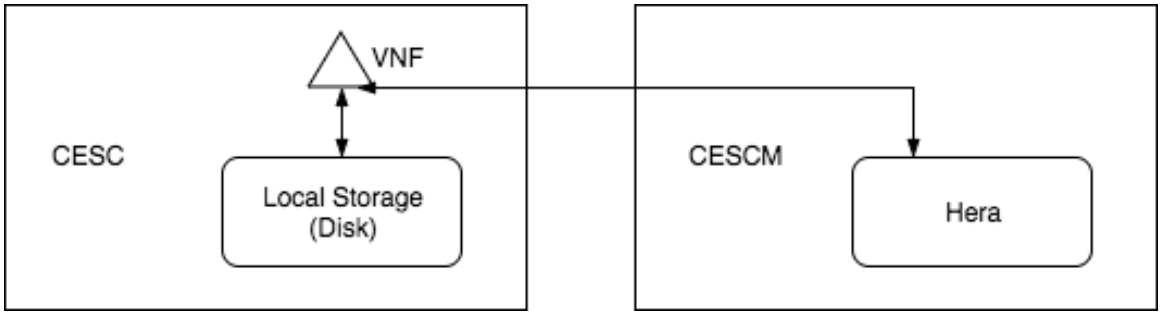


Figure 15: The read/write access of VNFs across the system

Figure 15 shows how a VNF can have read/write access of the local disk on the CESC or the Hera system on the CESCM.

These two available locations for data storage allow for more flexibility when writing VNFs and allows the developer not to be worried about limited storage on the CESC. Hera will allow the developer to use Hera storage if the VNF is considered as using a lot of disk space and based on their expertise and knowledge of their VNFs. A VNF developer is free to choose either Hera storage, which is remote to the CESC, or lvm, which runs on the CESC, or even both. This has to be set in the deployment configuration of the VNF as Cinder volume types.

The Placement of storage inside Hera is handled by the scheduler. The Hera scheduler decides the pool on which a volume for a VNF should be placed, matching the needed storage capacity and the requested resiliency of the storage that should be used for the VNF storage.

When the VNF is deployed it will have information about what storage configurations should be used for the VNF. Said configuration will contain the volume types and the size of the volumes to be created.

According to the storage type which should be used, Cinder will react differently, this is enabled by using scheduler hints[60] and the instance locality filter. Scheduler hints are a key/value map that allow more parameters for the creation of a volume. The instance locality Filter[61] forces the location of a new volume to be local to the compute instance.

[60] See: *https://specs.openstack.org/openstack/heat-specs/specs/kilo/cinder-scheduler-hints.html*
[61] See: *https://docs.openstack.org/developer/cinder/scheduler-filters.html#instancelocalityfilter*

When the volume that is to be created is of type Hera no scheduler hint has to be set and the instance locality filter is not triggered to force a Hera volume on the CESC, where no Hera deployment is running. If the volume should be a local volume to the instance on the CESC, then the scheduler hint for “same host” has to be set and the locality filter will know that the volume will be deployed on the host where the instance is to run.

For more details on configuring Hera and LVM for the CESCM and CESCs, please refer to 3.6.5.

3.6.3 Resiliency

ZFS offers three types of datasets, that is: (i) file systems; (ii) volumes (virtual raw block devices), and; (iii) snapshots (either of a file system or of a volume) which are all defined on top of virtual storage pools (zpools) made of virtual devices (vdevs). The vdevs are made of block devices and, when a zpool aggregates them, it can be configured with different redundancy settings: no redundancy, mirrored configuration, raidz1, raidz2 or raidz3 software raid (in each raidz configuration, a zpool can sustain the loss of 1, 2 or 3 vdevs, *respectively*, without losing data).

The basic principle for the implementation of data resiliency in Hera is the aggregation of vdevs into erasure-coded zpools that use the available raidz levels. Such zpools can then be reused to allocate block devices or to instantiate vdevs to be made part of other zpools.

Following two main resiliency configurations are presented that are based on different strategies: (i) aggregation of redundant iSCSI LUNs defined at the backend into zpools at the frontend, and; (ii) aggregation of "plain" iSCSI LUNs coming from the backend into zpools defined at the frontend.

The variables used for the quantities involved in the definition of the two strategies are described as such:

- **N**: Number of disks per backend node that are used in the storage cluster.
- **M**: Number of backend nodes.
- **K**: Raidz redundancy level at backend nodes. $1 \leq K \leq 3$, where each possible value corresponds to the ZFS raidz1, raidz2 and raidz3 modes.
- **H**: Raidz redundancy level at frontend node. $1 \leq K \leq 3$, where each possible value corresponds to the ZFS raidz1, raidz2 and raidz3 modes.

3.6.3.1 Zpools of redundant LUNs

With this redundancy configuration, the N available disks at each backend node are aggregated to form zpools with a certain raidz level K (i.e., raidz1, raidz2 or raidz3). A single ZFS volume is then defined over the pool and exported as an iSCSI target to the frontend nodes.

Therefore, the iSCSI LUNs attached at the frontend nodes already include a level of redundancy.

At the frontend, a new zpool is defined using the iSCSI targets from the backend with an additional redundancy level H.

With reference to the architecture of Hera, the following two requirements can be easily met with this layout:

- Resist to the failure of up to K disks in each backend node without affecting the operations of the storage cluster. This level of resiliency is achieved with the definition of the local zpools to the backend nodes, that allows the creation of redundant volumes to be exported to the frontend. Each volume is backed-up by N physical disks and can remain operational when more than $N - K - 1$ disks are still active.

- Resist to the failure of more than K disks in no more than H backend nodes. In general, up to H backend nodes can completely fail without affecting the operations of the storage cluster. This requirement is satisfied because we export one resilient volume from each backend node and then collect them into a zpool at the frontend, with an additional redundancy factor of H. With M backend nodes (and thus M vdevs building the zpool at the frontend node), the system will remain operational when more than $M - H - 1$ backend nodes are still active.

The requirements that this layout is able to satisfy demonstrate that many flexible redundancy policies can be implemented by combining different values of the K and H variables. In particular, the storage efficiency ε (usable vs. raw storage) that results from this allocation scheme is given by the following formula:

$$\varepsilon = \frac{N - K}{N} \cdot \frac{M - H}{M}$$

In a cluster with 24 disks per backend node, 5 backend nodes, backend redundancy level of 3 and frontend redundancy level of 1, the resulting efficiency is $\varepsilon = 70\%$.

Note that with this layout, both K and H can obtain a value of 0, indicating that no raidz redundancy is added to the backend or frontend zpools respectively. In addition, it is possible to use different backend redundancy levels at different backend nodes, to further increase the flexibility of this scheme or to homogenize different configurations of backend nodes (e.g., different number of available disks) and present a uniform set of virtual volumes to the frontend.

While this approach allows a high degree of flexibility, creating two layers of redundant zpools introduces complexity. A simpler scheme, presented below, allows the achievement of similar requirements with a less sophisticated layout.

3.6.3.2 Zpools of plain LUNs

This is a simplified approach than the one described in 3.6.3.1. All the storage devices of each backend node are individually made available as iSCSI LUNs to the frontend without introducing any redundancy. At the frontend node, the LUNs are collected into a set of zpools, which are formed by taking into account the number of devices per pool and their origin (the backend node at which they are physically attached). In particular, each frontend pool includes a constant number of LUNs exposed by a subset of the backend nodes (the pool is said to span over those nodes).

The composition of the frontend zpools must allow for the satisfaction of the following requirements:

- Let the cluster tolerate the failure of no more than K disks across all the backend nodes.
- Let the cluster tolerate the failure of no more than one (entire) backend node.

With respect to the layout described in 3.6.3.1, the requirements are relaxed resulting in a cluster that can tolerate an inferior degree of failures. Whereas in 3.6.3.1, the tuning of the K and H variables allows the system to remain operational in front of the loss of up to H backend nodes and K disks for each backend node, with this approach we sacrifice flexibility in favour of a simpler design.

The constraints in the design of this solution impose that each zpool defined at the frontend cannot include more than H disks from each backend node. If this would be violated, the failure of a single backend node would make the whole pool non-operational. Consequently, the number of pools to be defined at the frontend node is equal to $\left\lceil \frac{N}{H} \right\rceil$ and the size of each pool (in terms of number of disks) has an upper bound of $(N \cdot H)$.

These constraints make the storage efficiency ε of this layout independent from H and its definition is as follows:

$$\varepsilon = \frac{M-1}{M}$$

The rationale in preferring this solution over the one in 3.6.3.1 is that the storage efficiency (ε) can still be defined within a wide range of possible values by changing the number of backend nodes over which a pool defined at the frontend spans (i.e., backend nodes offering devices that are included in a certain zpool at the frontend). This can create a logical separation of the backend servers, which can be aggregated again at the frontend by using nested vdevs in a single, global, zpool.

In a cluster with 24 disks per backend node and each frontend pool spanning over five backend nodes, the resulting efficiency is $\varepsilon = 80\%$ and eight pools would be defined at the frontend with 15 disks for each pool.

3.6.4 Measurements

The evaluation focused on demonstrating the performance in terms of reachable bandwidth, in comparison to another storage system, in this case the popular software Ceph [6] as comparison point with a mature suite. We have designed our system to be resilient up to the designed amount of disk failures, which we do not specifically evaluate here, rather focusing on the verification that performance can still be maintained.

The evaluation of the Hera setup has been performed using four servers. Three, used as backends, are equipped with two Xeon 4 Cores 2.67GHz CPUs, 64GB RAM and four 1TB spinning drives and a fourth one, used as frontend, equipped with one Xeon 4 Cores 2.13GHz CPU and 6GB RAM. The Ceph setup uses the same servers: the node used as frontend in the Hera setup is a Ceph Monitor, and the three others are Ceph OSDs. The servers are all attached to the same 1Gbps network.

The frontend pool is made of directly exported LUNs from the backend, as described in section 3.6.3.2 and achieves 66% efficiency.

Bandwidth benchmarks have been performed with fio[62], a customizable micro benchmarking tool which can generate I/O load by reading and writing to files. It proposes many options for the actual load, such as sequential read, sequential write, random read and random write among others. As our architecture has not been designed with a specific type of workload in mind, we performed benchmarks using both sequential (i.e., reading or writing sequential blocks) and random workloads (i.e. writing random blocks over a large file).

The results are summarized in Table 2 and Table 3. The first table presents the results of sequential benchmarks while second table presents the results of random benchmarks, with different block sizes for the requests. Each result is presented as the average out of ten benchmarks, with the standard deviations also reported.

5GB file MB/s		**Hera**		**Ceph**	
Mode	bs	Average	Dev	Average	Dev
3*Read Sequential	4K	112,7	0,6	114,5	0,9
	16K	110,3	1,2	114,4	0,4
	64K	113,6	1,4	113,4	0,8
3*Write Sequential	4K	33,7	6,2	113,5	0,7
	16K	65,6	1,7	121,5	1,3
	64K	66,7	2,4	107,8	0,2

Table 2: fio benchmarking – sequential mode

[62] http://git.kernel.dk/?p=fio.git

5GB file MB/s		Hera		Ceph	
Mode	bs	Average	Dev	Average	Dev
3*Read Random	4K	15,8	4,3	10,7	1,5
	16K	41,8	7,4	18,1	0,7
	64K	105,6	5,3	43,9	1,5
3*Write Random	4K	1,7	0,4	1,8	0,1
	16K	8,5	0,2	9,9	0,3
	64K	33,2	0,6	61,6	2,1

Table 3: fio benchmarking – random mode

Different patterns can be observed when comparing the benchmarks:

- In sequential read mode, the resulting bandwidth is very similar between the two architectures, which we interpret as reaching the maximum hardware bandwidth in both cases.

  Additionally, the bandwidth barely changes with the block size of the I/O units. This is expected as the next block is always read whatever its size and many requests are queued, which means the block size has little impact on the overall performance.

- In random read mode, the Hera architecture displays a better performance for all three tested block sizes, with the biggest difference with the 64k block size. We attribute this difference to the fact that data is striped over multiple drives, and that access to data might be parallelized over a number of drives if the data request is large enough to be chunked into multiple access requests at the file system level.

- In both write and random write, Ceph performs better, with a more important margin in sequential write than in random write. Indeed, the nature of raidz3 ZFS pools means that each file system block is striped over potentially all devices comprising the RAID-Z vdev, meaning that each write operation has to wait until all disks complete the write. Nevertheless, the block size has a positive impact on performance of the Hera architecture bandwidth-wise as larger blocks may be split at the disk level and written in parallel across the pool individual disks.

3.6.5 How to write a Cinder driver

In order to achieve an integration of Hera into SESAME, which uses OpenStack, it is necessary to write a Cinder driver.

First of all, we have the Hera storage system with a RESTful API, all the logic and functionality is already available. We position the driver as a proxy between Cinder and Hera. To implement the driver methods, one does not have to look very far, there is a page on the OpenStack Cinder docs that explain which methods need to be implemented and what they do. For a basic Cinder Driver skeleton check out this repository: Cinder Driver Example[63].

We have decided for a normal volume driver, but you may also decide for another driver that you want to write, then you need to inherit from another base driver, e.g.: write your Driver for SAN volumes (SanDriver) or for iSCSI volumes (ISCSIDriver). Also we have always looked at other drivers (mainly the LVM driver) for some guidance during the implementation.

These methods are necessary for a complete driver, while implementing it we wanted to try single methods after implementing them. Once the mandatory methods were implemented, and we attempted to execute the driver's code, nothing was happening! We quickly realised, that the get_volume_stats method returns crucial information of the storage system to the Cinder scheduler. The scheduler will not know anything of the driver if values are not returned, so to quickly test we had this dict hardcoded and the scheduler stopped complaining.

```
{
    'volume_backend_name': 'foo',
    'vendor_name': 'bar',
    'driver_version': '3.0.0',
    'storage_protocol': 'foobar',
    'total_capacity_gb': 42,
    'free_capacity_gb': 42
}
```

In order to provide parameters to your driver, you can also add them in the following way, as part of the driver implementation. Here, we add a REST endpoint as a configuration option to the `volume_opts` part.

```
volume_opts = [
    cfg.StrOpt('foo_api_endpoint',
               default='http://0.0.0.0:12345',
               help='the api endpoint at which the foo storage system
sits')
]
```

[63] See: *https://github.com/icclab/Cinder-Driver-Example*

All of the options that are defined, can be overwritten by putting them inside the `/etc/cinder/cinder.conf` file under the configuration of our own driver.

In order to understand what values Cinder will give to a driver the volume parameter can be used. When you get to implement the functionality of the driver, you will want to know what is passed to the driver by cinder, especially the `volume` dict parameter is of great interest, and it will have these values:

```
size
host
user_id
project_id
status
display_name
display_description
attach_status
availability_zone
// and if any of the following are set
migration_status
consistencygroup_id
group_id
volume_type_id
replication_extended_status
replication_driver_data
previous_status
```

To test your methods quickly and easily, it is very important that the driver is in the correct directory, in which all the Cinder drivers are installed, otherwise Cinder will, naturally, not find the driver. This can differ on how OpenStack has been installed on your machine. With devstack the drivers are on: `/opt/stack/cinder/cinder/volume/drivers`. With packstack they will be on: `/usr/lib/python2.7/site-packages/cinder/volume/drivers`.

There was one last headache that needed to be resolved to allow full integration of our cinder driver. When the driver is placed the correct directory, we proceed to add the necessary options (as shown below) to the `/etc/cinder/cinder.conf` file.

```
# first we need to enable the backend (lvm is already set by default)
enabled_backends = lvmdriver-1,foo
# then add these options to your driver configuration at the end of the
file
[foo]
volume_backend_name = foo # this is super important!!!
```

```
volume_driver = cinder.volume.drivers.foo.FooDriver # path to your driver
# also add the options that you can set freely (volume_opts)
foo_api_endpoint = 'http://127.0.0.1:12956'
```

You must set the `volume_backend_name` because it links Cinder to the correct backend, without it nothing will ever work (NOTHING!).

Finally, when you want to execute operations on it, you must create the volume type for your Cinder driver:

```
cinder type-create foo
cinder type-key foo set volume_backend_name=foo
```

Now restart the Cinder services (c-vol, c-sch, c-api) and you should be able to use your own storage system through Cinder.

4 Virtualization platform for ARMv8

In this section, the virtualization-related activities performed as a contribution to the SESAME platform prototype are depicted.

4.1 Accelerated computing

The SESAME virtualization computing acceleration is mostly based on FPGA. The project FPGA virtualization technology has been developed by VOSYS starting from the design phase, targeting PCI-Express FPGA boards such as the Virtex UltraScale XCVU095[64]. The testing and development platform used has been Intel x86, mainly because of the higher maturity of SR-IOV[65] and IOMMU[66] support on this platform.

The design phase has been reported, together with the SESAME FPGA virtualization architecture, in [3]. After the design phase, a first implementation of the FPGA virtualization infrastructure has been completed, including SR-IOV support and a DMA. This configuration is able to allocate a number of virtual machines directly to hardware accelerators, minimizing the performance overhead. In this way, the VNFs are able to offload the computation of the data processing, significantly improving system performance and power consumption.

The integration of such technology with OpenStack enables automatic booting and creation of VMs directly from the dashboard.

4.2 Accelerated virtual networking

As already introduced in [3], VOSYSwitch was chosen in the SESAME project for the purpose of provisioning the network for the VNFs running within the Light DC. As such, it has to cover the specific architecture requirements of the platform.

The hardware chosen to implement the edge computing node within the project is NXP's LS2085A-RDB (see Section 3.1) which is based on an SoC from the Layerscape 2 family. It provides an advanced embedded network dataplane (DPAA2) and a set of eight 64-bit ARM cores – Cortex A57. The latter is an implementation of ARM's server type CPU architecture and provides advanced features such as out of order execution, encryption instructions, etc.

The OpenDataPlane initiative started by ARM within the Linaro consortium[67], provides a set of highly effective user-space drivers that are specifically targeting portable dataplane (DP) implementations. Its API also exposes hardware accelerated packet processing functionalities like packet classification, scheduling, queue management and encryption. NXPs LS2085A-RDB SDK comes with a set of drivers that implement the stable "Monarch" 1.11 release of the ODP[68] API. VOSYSwitch, as a dataplane that provisions the VM networking within chosen hardware

[64] For more relevant information also see, *inter-alia*: *https://www.xilinx.com/support/documentation/selection-guides/ultrascale-fpga-product-selection-guide.pdf*

[65] For more related information see, *inter-alia*: *https://en.wikipedia.org/wiki/Single-root_input/output_virtualization*

[66] Also see: *https://en.wikipedia.org/wiki/Input%E2%80%93output_memory_management_unit*

[67] *https://www.linaro.org/*

[68] See: *https://www.opendataplane.org/*

platform, leverages these drivers as a fast and accelerated way to access the network hardware in the user space and bypass the Linux Kernel networking stack.

Within the Light DC deployment, VOSYSwitch dataplane is used as a virtual switch connecting the virtual network functions with the physical network. The OpenStack's network management facility Neutron, implements several different network isolation methods; VLAN and GRE being the most popular amongst them. These two methods cover respectively the two major distinguishing network connectivity approaches on Layer 2 (VLAN) and Layer 3 (GRE). The physical network equipment setup, like the top of the rack (ToR) switches depends on the selected Neutron network isolation method. Since VLAN management requires more specific equipment, and possibly manual configuration, GRE tunneling was used as the isolation method in SESAME.

The Generic Routing Encapsulation (GRE) protocol is described in RFC 2784[69]. Figure 16 shows the schematic representation of its header format. The default Linux IP stack implementation uses only the "Key" optional field, and this is what OpenStack with Open vSwitch employs in a default GRE-*enabled* installation. It is important to note here, that the Linux Kernel supplied with the NXP SDK does not include the needed module to handle these tunnel interfaces, which has put even stronger requirements on VOSYSwitch to provide an efficient way to handle this type of traffic efficiently.

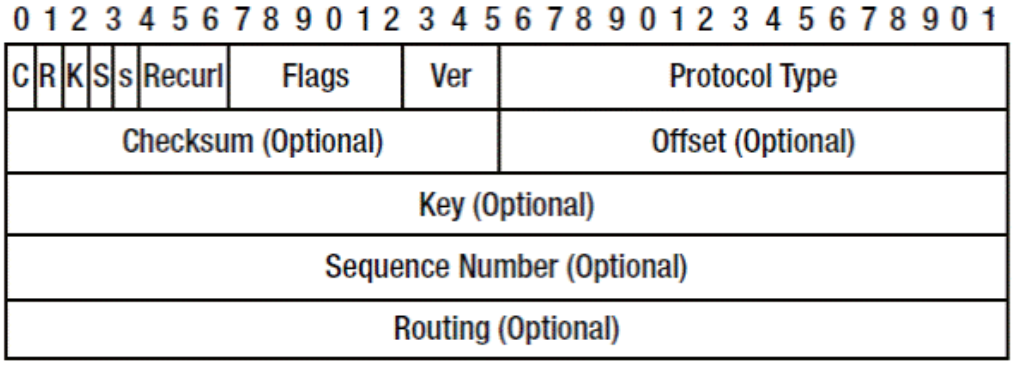


Figure 16: GRE tunneling header format

VOSYSwitch, as it was introduced in SESAME, was not implementing GRE header encapsulation/decapsulation capabilities. The full support was done completely within the project. The component that provides the function to (de)multiplex the GRE tunnels is called GREMux (Figure 17). It has an uplink port, which is connected to the physical network that carries the GRE encapsulated traffic. Once the packet is received it is tripped from its outer Ethernet, IP and GRE header, and depending on the "key" field of the latter, it is redirected to the corresponding tenant VMs.

[69] See: https://www.ietf.org/rfc/rfc2784.txt

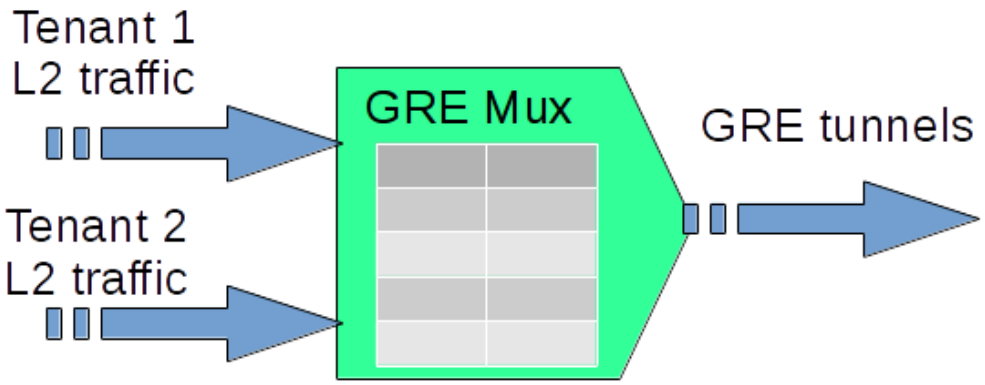


Figure 17: GREMux

The nature of the GRE tunnels is that each endpoint (or GRE Mux component in the VOSYSwitch), needs to establish a separate connection to each of the other participating termination points. This means that within middle-sized datacentre, each VOSYSwitch has to maintain hundreds of active connections, where each connection can potentially carry thousands of isolated tenant tunnels. Managing all those tunnels and connections can be a very challenging task. The solution implemented within the SESAME project, can do this very effectively, due to the leveraging of the high performance hash tables, which are part of the LuaJIT[70] and the specifically designed MAC table lookup implementation (see Figure 18).

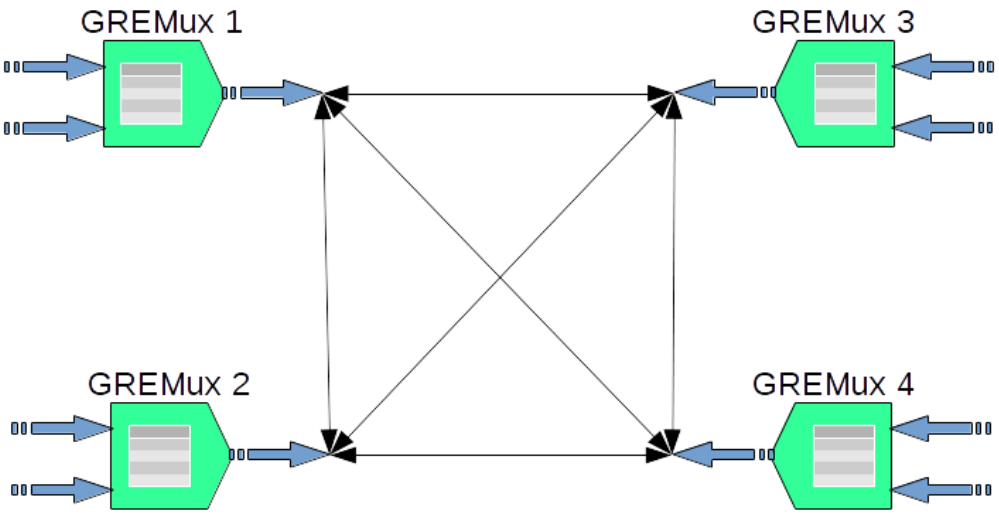


Figure 18: GRE tunnel full-mesh of four GREMux

[70] See: http://luajit.org/

4.3 OpenStack agents for ARMv8

The Ligh DC within SESAME is managed with the open source OpenStack virtual infrastructure management software. The demonstration setup was mainly tackling the problems of effectively sharing the compute and network resources, leaving the storage management as a subject of the other tasks within the project. In such a scenario, the main actors in a compute node, from the point of view of the VIM, are the Nova [9], and Neutron [10], agents. The first one controls the management of the virtualization of the CPU and memory resources, while the second manages the establishment of the network related entities. Examples of the latter are virtual network, subnet and port.

Due to the specifics of the platform SDK, distributed by NXP with the development board LS2085A-RDB, which was used within SESAME, the OpenStack version deployed is, the now obsolete, Kilo[71]. As of the moment of writing this document the minimum supported stable release is Newton[72], i.e. two versions forward after Kilo. As we will discuss later, using this old version raised a certain number of problems that were resolved during the course of establishing the demo.

4.3.1 Accelerated host to guest communication

VOSYS is the company that initated and upstreamed the switch to guest connectivity approach called vhost-user. Today this is the de-facto standard for connecting user-space network dataplanes and KVM guests. It leverages the OASIS[73] standartised paravirtualisation paradigm called VIRTIO[74] and the pre-existing Linux in-kernel VIRTIO offload processing called vhost.

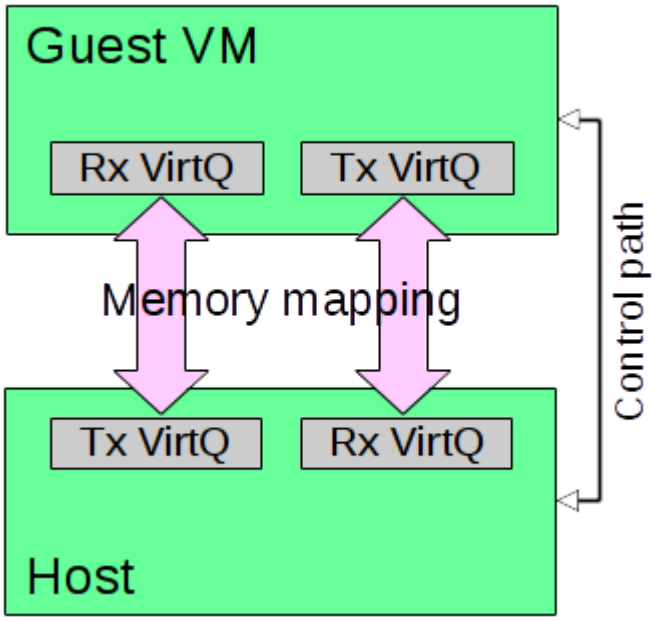


Figure 19: vhost and vhost-user

[71] For more details see: *https://releases.openstack.org/kilo/*
[72] For more details see: *https://releases.openstack.org/newton/*
[73] See: *https://en.wikipedia.org/wiki/OASIS_(organization)*
[74] For more details see: *http://docs.oasis-open.org/virtio/virtio/v1.0/virtio-v1.0.html*

As Figure 19 shows, the essential concept in both the vhost (in-kernel) and vhost-user (the user space implementation) is the same and relies on two components: share/map the Rx/Tx virtual queue memory between the host and the guest, and a control path, which assists the setup of that memory sharing. While the former is self-explanatory and has the clear purpose of passing the packet data between the host and the guest world, the latter concept is used to provide a control communication channel in parallel to the Rx/Tx data path.

VOSYSwitch, being a user-space data path, implements the vhost-user protocol to achieve an efficient, paravirtualized network solution, based on VIRTIO. The pre-requisites for this protocol are a huge-page backed guest memory allocation and a UNIX socket to pass the control messages.

4.3.2 Nova huge page allocation

When allocating the resources for a new VM, Nova chooses a suitable compute host by inspecting its hardware properties. In a common QEMU/KVM based compute node those properties are gathered by the Nova compute agent through API calls to the libvirtd[75] virtualisation management daemon. Its responsibility is to enumerate the CPU and RAM resources amongst others. In a modern multi-socket system, those resources are organised in a hierarchy. Such hierarchy is called NUMA[76] topology, and describes the relations (distance) between the CPU, Memory and PCI resources on a host.

In order to allocate a huge-page backed memory block for the guest VM's RAM, Nova tries to find a compute node that has available the requested CPU cores and the amount of needed memory attached to the same NUMA node by inspecting the aforementioned topology of the available compute nodes.

From the NUMA point of view, LS2085A[77] is a SoC that consist of a single node, equipped with eight CPU cores, and all the system memory attached to it. However, the default libvirt library installed by the supplied NXP SDK is not producing an adequate NUMA topology description, which resulted in an inability to create a VM using huge pages. As previously described, Nova needs the correct description of the HW resources in order to choose it as a target to allocate the resources and bring up the VM on it. Therefore, this libvirt shortcoming on the LS2085A makes the whole compute node unable to be used by the Nova in the specific setup.

To overcome this problem, it was necessary to produce a set of patches as follows:

- Nova compute agent enforces the NUMA topology, such that reflects the real LS2085A, ignoring the libvirt inaccurate result.
- Qemu enforces the sharing of the allocated huge-pages memory.

In the recent versions, Neutron ML2[78] mechanism drivers and their counterpart Neutron agents are provide as and external plugins. However, OpenStack Kilo pre-dates this concept, which

[75] See: *https://libvirt.org/*
[76] For more details see, *for example*: *https://docs.openstack.org/admin-guide/cli-nova-numa-libvirt.html*
[77] Also see: *http://www.nxp.com/products/microcontrollers-and-processors/arm-processors/qoriq-layerscape-arm-processors/qoriq-layerscape-2085a-and-2045a-multicore-communications-processors:LS2085A*
[78] See: *https://wiki.openstack.org/wiki/Neutron/ML2*

requires an intrusive approach to enable external add-on network provisioning implementations. This is done on the controller node, where VOSYSwitch installation requires inserting ML2 Mechanism driver entry points in the Neutron's "entry_points.txt" as follows:

In the [neutron.ml2.mechanism_drivers] section add:

vosyswitch =
neutron.plugins.ml2.drivers.vosyswitch.mech_driver.mech_vosyswitch:VosyswitchMechanismDriver

In the [console_scripts] section add:

neutron-vosyswitch-agent =
neutron.plugins.ml2.drivers.vosyswitch.agent.vosyswitch_neutron_agent:main

4.3.3 VOSYSwitch Neutron agent with GRE

As described in the previous section, a full GRE support in VOSYSwitch was developed for the purpose of the demonstration lab setup. However, to put this functionality in the context of the OpenStack deployment, an extension of the VOSYSwitch Neutron agent was needed. Originally, this agent was supporting only VLAN network isolation. Even if it is efficient and enough for real world production environments, where the number of tenants is restricted to few thousands, it lack a substantial functionality when it needs to drive the GRE based networking.

The default agent configuration implements learning. That is, any unknown unicast destination, multicast or broadcast traffic is flooded out tunnels to all other compute nodes. Any incoming traffic is used for its source MAC address. That MAC address is added to a learning table, so future traffic to that MAC address is not flooded but sent directly to the hosting node. There are several inefficiencies here:

- The MAC addresses are not initially known by the agents, but the Neutron service has full knowledge of the topology.
- Broadcast ARP requests are send to the whole network, which can influence scaling with large number of VMs.
- The broadcast messages will be received by hosts that do not have VMs in the designated network.

To tackle these problems, Neutron introduces the L2population module, which forms a large forwarding database (FDB) of the whole network that it manages. The Neutron agents receive updates to the FDB, and do not need to generate broadcast traffic as the information is already provided by the central Neutron server.

This module is specificaly bound to the overlay networking in Neutron – GRE and VxLAN[79], and therefore was explicitly needed in the VOSYSwitch Neutron agent.

[79] https://en.wikipedia.org/wiki/Virtual_Extensible_LAN

5 Light DC Integration result

5.1 Virtualized infrastructure in OpenStack environment

The testbed environment provided at Italtel labs was been used to integrate and verify several aspects of the Light DC development:

1. Hypervisor, Hardware acceleration layer and Accelerated Virtual networking (VOSYSwitch) integration on the NXP platform.
2. OpenStack integration (the Kilo Release was selected): computing and networking managed by OpenStack controller in case of Hybrid node (Intel/ARM).
3. VNFs deployment in compute node based on:
    a. Intel, CPU-only;
    b. Intel, CPU+GPU;
    c. Arm, CPU-only.
4. VTU (Video Transcoding Unit) performance characterization in case of video transcoding service for CESCs.

The testbed is described in Figure 20.

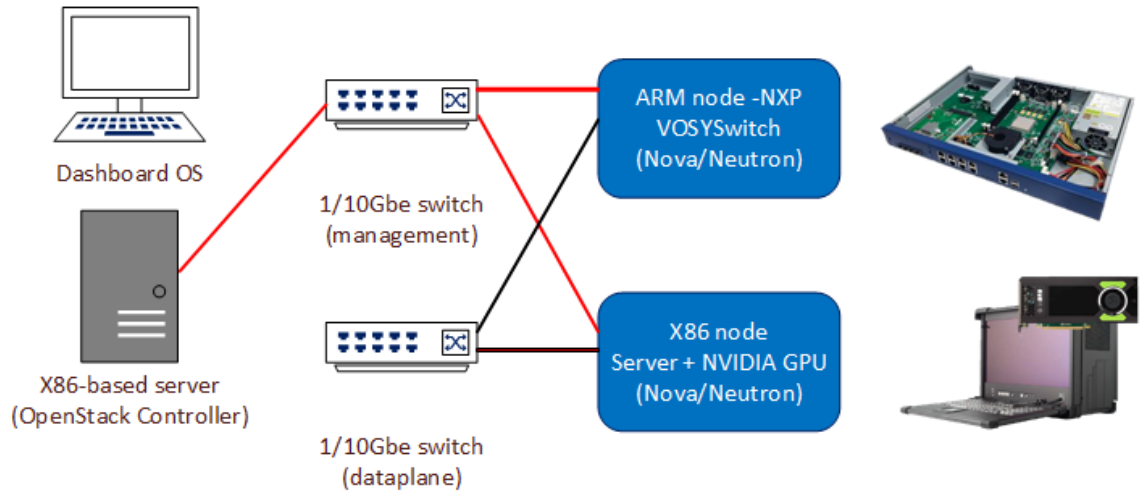


Figure 20: Light DC architecture for the PoC

5.2 VNF integration

The video transcoding feature is selected as first development of a service Virtual Network Function to be hosted on an ARM-based micro server.

The starting point for the VNF is an application running on Linux that includes FFmpeg[80].

FFmpeg is a free software package that provides libraries and programs for handling multimedia data; the main features of FFmpeg used by the transcoding VNF are:

- video and audio converter;
- sample rate converter;
- video resizing.

The application developed for the video transcoding feature has three main blocks (see Figure 21):

- Video Transcoding engine;
- Web server (GUI interface);
- Controller (CLI interface).

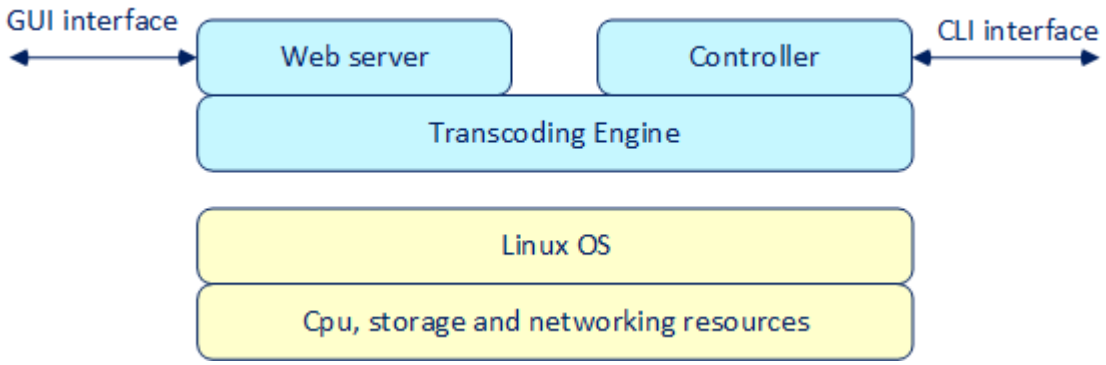


Figure 21: Transcoding Application Architecture

5.2.1 Porting to x86

The first version of the service VNF for video transcoding (named VTU, Video Transcoding Unit) has been developed and tested on an x86 server (waiting for an ARM-*based* virtualization platform). It runs on a x86_64 multicore server with support to virtualization (Intel® Virtualization Technology, Intel® VT) and which provides the PCI pass-through[81] feature for guaranteeing GPU acceleration on the PCIe bus.

[80] For more details see: *https://ffmpeg.org/*
[81] For further information see: *https://en.wikipedia.org/wiki/X86_virtualization*

The operating system chosen for integrating the NVDIA CUDA toolkit (release 7.5), needed for managing the GPU M4000, is Linux (kernel version 3.10, CentOS 7[82] distribution).

The VTU image developed for x86 is in the "qcow2" format[83], OpenStack compatible. It was tested and validated on the testbed in two different scenarios:

- CPU-only.
- Use of HW acceleration via GPU.

5.2.2 Porting to ARMv8

As soon as the NXP LS2085A reference board was chosen as the micro-server for hosting the service VNFs developed in SESAME, the porting to this platform has started.

In particular, several steps was necessary:

- Porting of Linux using the NXP BSP/SDK (yocto, LS2085A SDK 2016 0304, poky[84]), kernel version: 4.1.8 (the KVM hypervisor is natively integrated).
- Integration of QEMU, VOSYSwitch, Nova and Neutron agents for building the virtualization platform in the OpenStack environment. Several patches were needed to run the Armv8 based compute node in OpenStack (Kilo release).
- The VTU qcow2 ARM image needs the UEFI[85] boot loader in order to be launched by KVM, but UEFI boot is not managed by the OpenStack Kilo release. To overcome this issue three different images were built:
    1. Kernel image
    2. Initrd image
    3. a raw image with linux file system
- The first kernel image was based on *aarch64ArchLinux*[86], but because it had an issue with the management of the MAC addresses between the host and guest element, it was necessary to switch to the *AltArch* CentOS 7.3[87] for Arm64 (kernel 4.5.0).
- This choice would imply the porting of audio and video codecs inside the distribution, as they are not available natively. At the end, the chosen solution makes use of a container that includes the whole FFmpeg package.

5.2.3 Outcomes

A hybrid two nodes Light DC has been implemented. It is visible by OpenStack (Kilo release), with two VTU images:

- One for x86 with or without GPU HW acceleration.

[82] See: *https://www.centos.org/*
[83] For more details see: *http://git.qemu.org/?p=qemu.git;a=blob;f=docs/specs/qcow2.txt*
[84] More information can be found at: *https://www.yoctoproject.org/tools-resources/projects/poky*
[85] See: *https://en.wikipedia.org/wiki/Unified_Extensible_Firmware_Interface*
[86] For further details see: *https://archlinuxarm.org/platforms/armv8/generic*
[87] Download from: *http://mirror.centos.org/altarch/7/isos/aarch64/*

- One for ARM without GPU HW acceleration (as the NVIDIA CUDA Toolkit does not support, starting from version 7.5, the ARM/Linux platform).

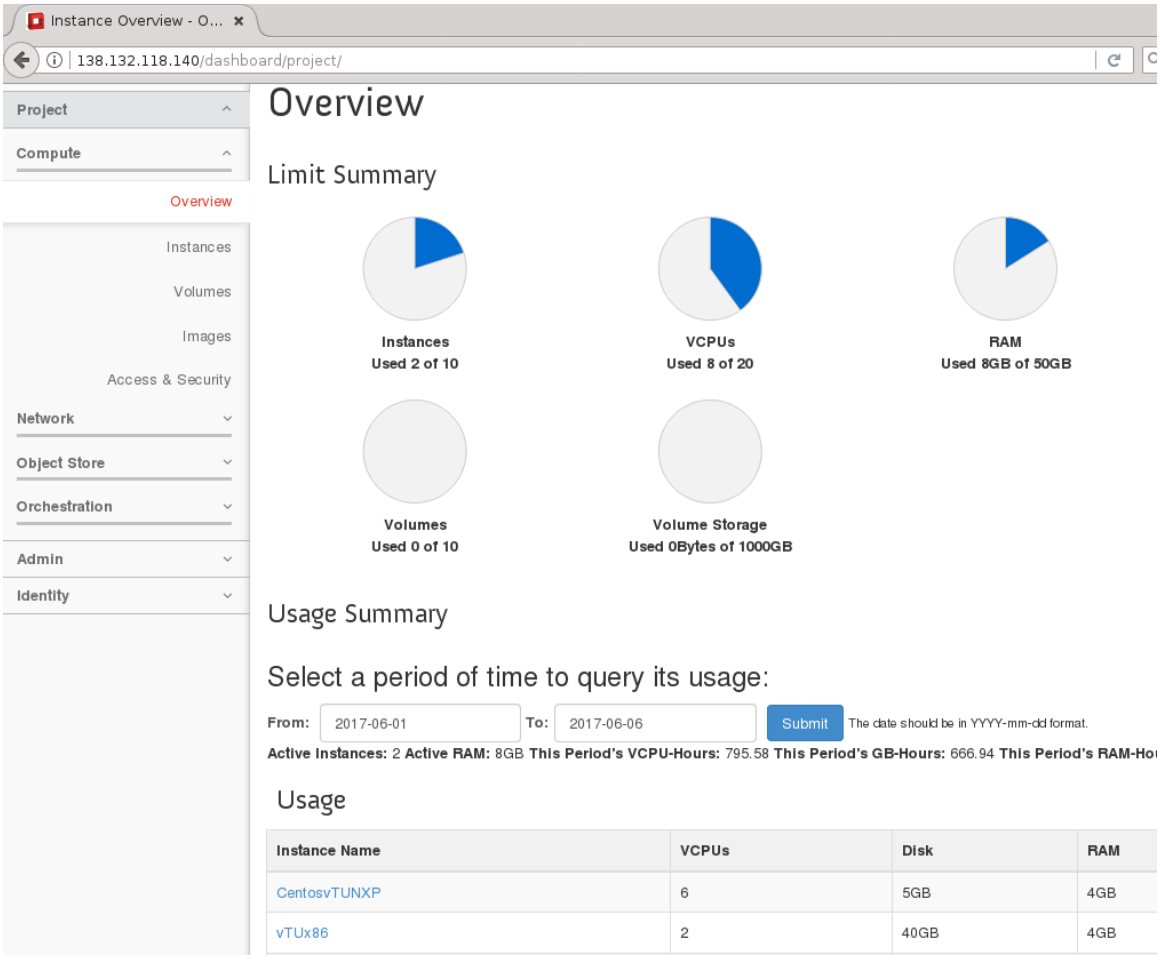


Figure 22: Virtualized resources available on the Light DC

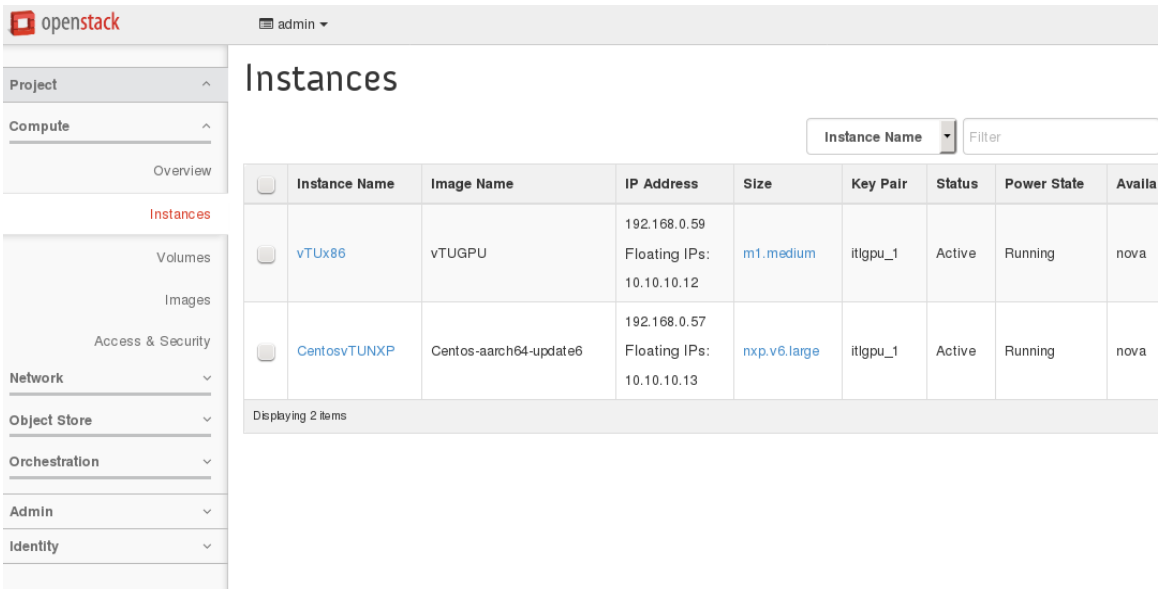


Figure 23: Active VNF Instances on the Light DC

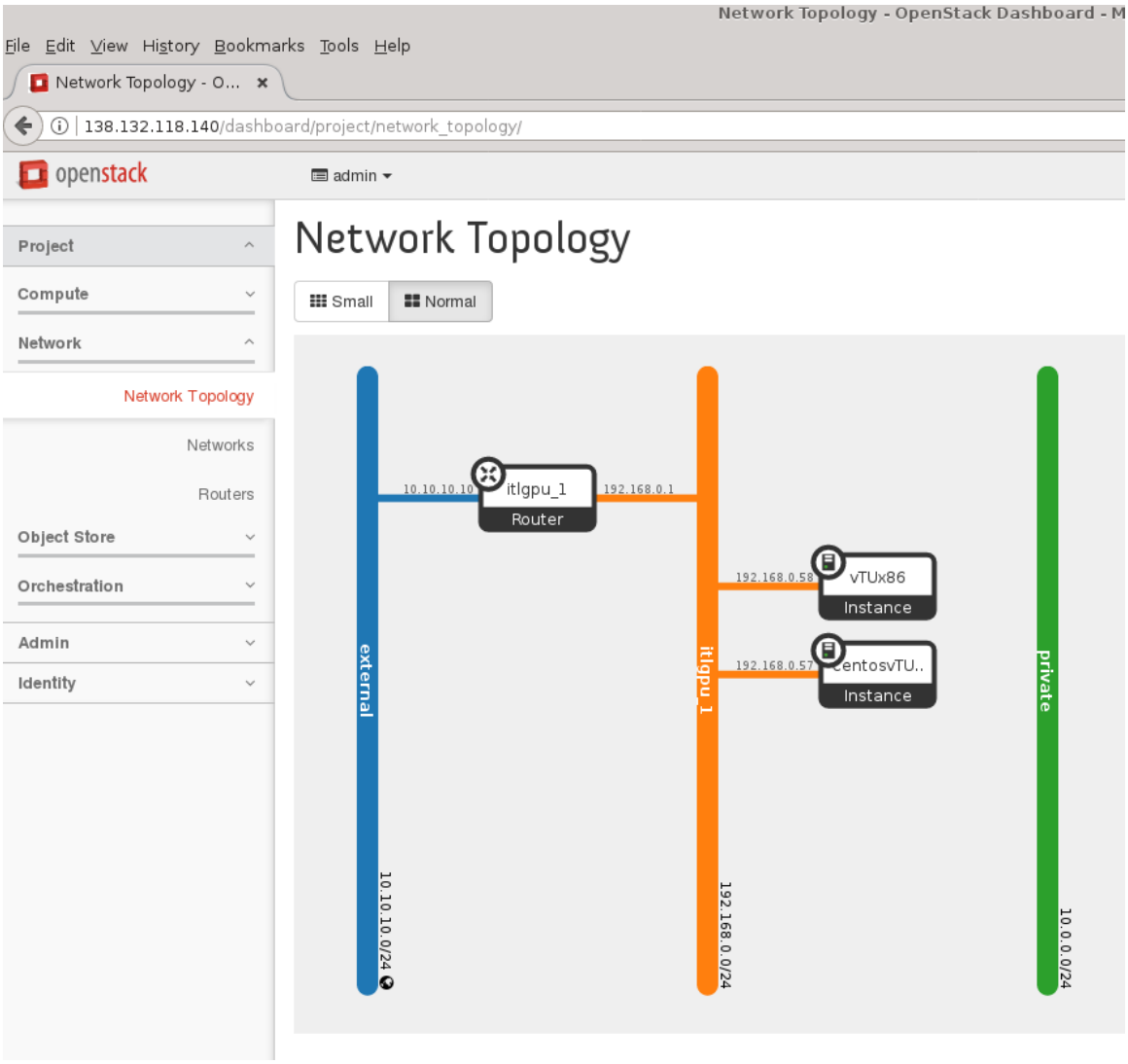


Figure 24: Light DC network topology (control and data on the same network)

5.3 Preliminary KPI

In this paragraph we consider some figures related to a specific service VNF, named VTU, to make some consideration about ARM versus x86, and hardware accelerated (using GPU) versus software-only platforms.

The VTU (Video Transcoding Unit) is an application that can convert video streams from one video format to another. Either it can run on bare metal environments, or it can be implemented as a VNF providing optimized video transcoding function, for the benefit of many other VNFs, to create enhanced services.

Depending on the type of application that should be provided, the source video stream could originate from a file within a storage facility, as well as coming in form of packetized network stream from another VNF. Moreover, the requested transcoding service could be mono-directional, as in video stream distribution-like applications, or bi-directional, like in videoconferencing.

The VTU VNF is implemented as SW module running on a virtual machine and can be installed in one or multiple physical servers clustered together through a local communication fabric. The VTU can support a large set of video codecs, and in particular the most recent and popular ones, such as H.264[88], H.265[89] and VP8[90]/VP9[91] ([14], [15]).

Although software-only functions can give acceptable performance in many applications, when compute-intensive workloads running at the data plane are of interest, such as those based on video data processing, quite poor results can be obtained. In these cases, to reach the expected performance, it is often necessary to consider a slightly different approach that involves the use of Hardware accelerators. In general, managing HW accelerators and making them transparently available to every VNF goes against the assumption of every virtualized environment, of having a uniform HW platform made of CPU-only computing elements. The presence of HW accelerators bound to a virtual function implies the use of a SW layer that must be HW-aware, thus significantly complicating system management operations and scalability, [12], [15]. Though, the advantages of HW acceleration can be so preponderant, in particular when performance, latency or Service Level Agreement (SLA) requirements are challenging, that not considering them can "push" a commercial product out of the market. In fact, acceleration is not just related to performance, but also to the reduction of the number of physical servers, footprint, network appliances and power consumption. In short, it can make the difference in the commercial proposition of a product.

A distinctive feature of VTU is the possibility not only to run on general purpose CPUs but also to exploit the Hardware acceleration provided by a GPU, to improve the compute performance of video codecs. To this end, two different architectural approaches can be used. The first one, also known as "cooperative CPU- GPU" makes use of a GPU to offload the most compute-intensive functions of the video codec (usually, the Motion Estimation block), while the main algorithm is kept running on the CPU. The second approach, conversely, uses full HW implementation of video codecs. Today, various HW versions of the most popular encoding schemes, such as H.264, HEVC, VP8 and VP9, are available ([13], [16]). The fully HW approach can provide higher

[88] See, *for example*: https://en.wikipedia.org/wiki/H.264/MPEG-4_AVC
[89] For more details see, *inter-alia*: https://en.wikipedia.org/wiki/High_Efficiency_Video_Coding
[90] For more details see, *inter-alia*: https://en.wikipedia.org/wiki/VP8
[91] For more details see, *inter-alia*: https://en.wikipedia.org/wiki/VP9

compute performance than cooperative CPU-GPU algorithms. Though, the HW approach very often lacks the flexibility in service management needed by service operators, thus the cooperative approach is still preferred in many real-life implementations. The VTU can adopt both GPU-accelerated approaches. In fact, it can use the NVIDIA[92] NVENC[93] encoder for the H.264 and H.265 encoding schemes [13]. Also, the CPU-GPU cooperative approach described in **Σφάλμα! Το αρχείο προέλευσης της αναφοράς δεν βρέθηκε.**, [18], can be used for the Google open Source VP8 encoder.

Many tests were carried out to achieve a full performance characterization of the VTU, both for the SW-only version, and the GPU-accelerated one. During all the tests it was verified that the RAM was not completely used, to be sure that the results were not influenced by the different quantity of RAM installed in Intel-based, rather than ARM-*based* micro-servers.

Kind of tests:

- SW-only version of VTU on two different micro servers: NXP (ARM) and GOMA (x86).
- The GPU-accelerated version of VTU on GOMA micro-server (equipped with NVIDIA Quadro M4000 GPU).

  Note: The first SESAME proposal provided for the use of GPU associated with ARM, however, in the meanwhile, NVIDIA decided to change strategy and not support GPU utilization for ARM platforms as NXP. This fact precluded the possibility to test the GPU-accelerated NXP platform.

In the following, a few meaningful results are presented and discussed.

In all tests, the same H.264 Full HD video file (1080x1920 resolution) was used as input.

In particular, Figure 25 shows the results obtained with the VTU featuring the H.264 transcoding (expressed in frames per second) without HW acceleration (SW-only) and with HW acceleration (using a GPU). The processing implies decoding from the input format to the one required as output.

The VTU provided four different video resolutions as output, in four different transcoding tests: VGA (480x640 pixel), HD480 (480x852 pixel), HD720 (720x1280 pixel), HD1080 (1080x1920 pixel).

The vertical axis represents the output resolution, while the horizontal axis indicates the achieved output frame-rate in frames per seconds (fps).

The Encoder used in SW-only VTU for H.264 is X264. The Encoder used by the GPU for H.264 and H.265 is NVIDIA NVENC.

Figure 25 collects the results of the tests achieved with H.264 encoders respectively. Only one session was launched for each test.

[92] NVENC is a technology used by NVIDIA that allows video hardware encoding. Many NVIDIA GPUs support this technology, among others some **GeForce** GPUs used in desktop and mobile computers.

[93] NVidia NVENC is a feature in its graphics cards that performs H.264 video encoding, offloading this compute-intensive task from the CPU. It was introduced with the Kepler-*based* GeForce 600 series in *March 2012*. For more informative details also see, *inter-alia*: https://en.wikipedia.org/wiki/Nvidia_NVENC

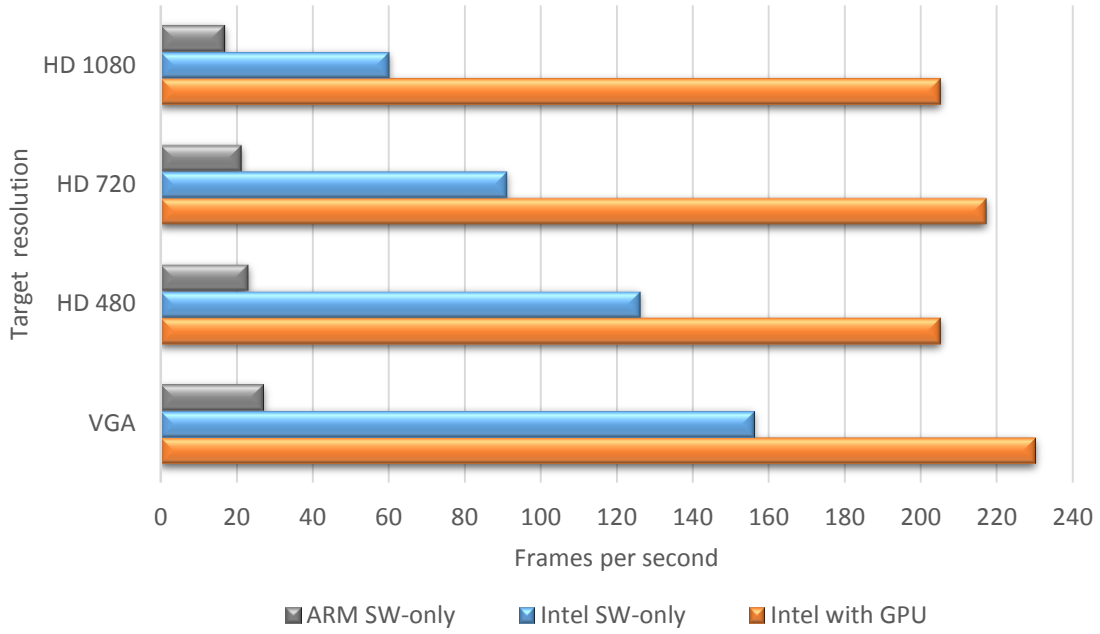


Figure 25: H.264 single session encoding performance (higher is better) measured on three different HW platforms, for different output resolutions

As one can easily see, the performance for the three HW platforms are very different. In particular, ARM performance is very poor, to the point that it is not conceivable a utilization in a real-time scenario. The improvement using the GPU compared to a SW-only solution is remarkable in all cases. This confirms the need of GPU acceleration especially in modern and future scenarios where even higher video resolutions are going to be used.

Another important aspect to emphasize is related to the occupation of compute resources during transcoding. Although in SW-only mode CPU resources were completely occupied, (all the CPU cores were running at 100%), using the GPU, both CPU and GPU resources were only partially used. This fact led us to a second set of tests in which multi-session performance was analysed. In this new set of tests, the focus was on a single case, i.e., H.264 HD1080, launching 2, 4, 8, and 16 concurrent transcoding sessions.

The results are reported in Figure 26 and Figure 27.

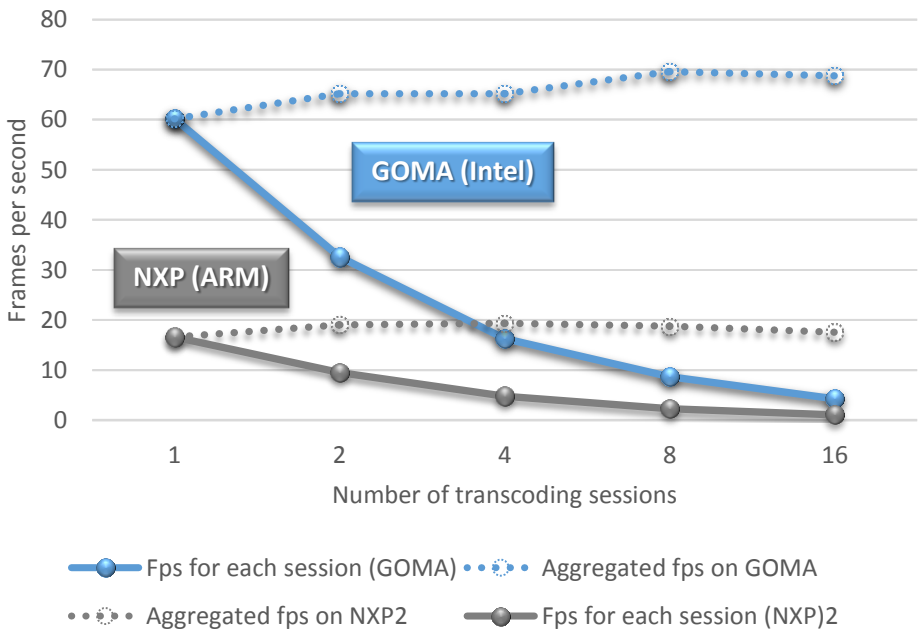


Figure 26: H.264 HD1080 encoding, SW-only, in multi-session transcoding tests (higher is better). Blue lines refer to GOMA, grey to NXP. Performance refer to each single session (solid line) and to aggregated sessions (dotted lines)

Considering the SW-only implementation (Figure 26), the performance of each single session decreases with the total number of executing sessions. Again, the ARM performance appears very low, being almost a fourth of the Intel one.

Comparing the global performance of 1 session to that with 2-4-8-16 concurrent sessions (aggregated fps) the result is very similar as you can see from the dotted lines. This can be easily justified considering that the CPU occupation during the processing is always around 100% also running a single session.

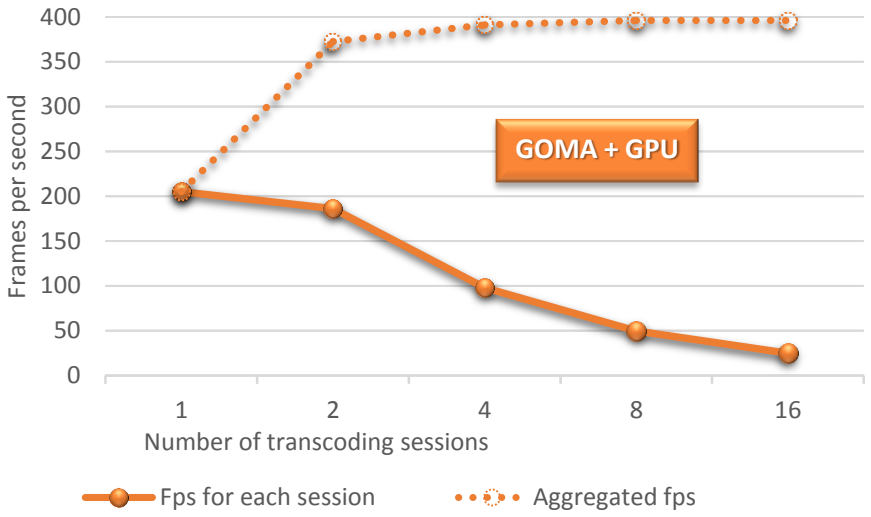


Figure 27: H.264 HD1080 encoding, using a GPU, in multi-session transcoding tests (higher is better). Performance refer to each single session (solid line) and to aggregated sessions (dotted lines).

The same is not true using the GPU (Figure 27). In this case the CPU is only partially used because the workload is mainly offloaded to the GPU whose resources are, in turn not fully used (FIGURE 29). Using the GPU with 16 concurrent sessions we reach 24.75 fps for each session, for a total of 16x24.75=396 aggregated fps. The 396/69 ratio brings to a 5.7x gain in performance using the GPU respect to a SW-only solution. Furthermore, during the GPU test with 16 transcoding sessions the CPU was running at 70% giving it the possibility to run other tasks. This was not possible with SW-only solution, because in such a case the CPU was always 100% occupied.

It is interesting to analyze what are the power figures of the three HW platforms used to run the tests (multi-session performance). Next figure (Figure 28) shows the power consumption, expressed in Watt.

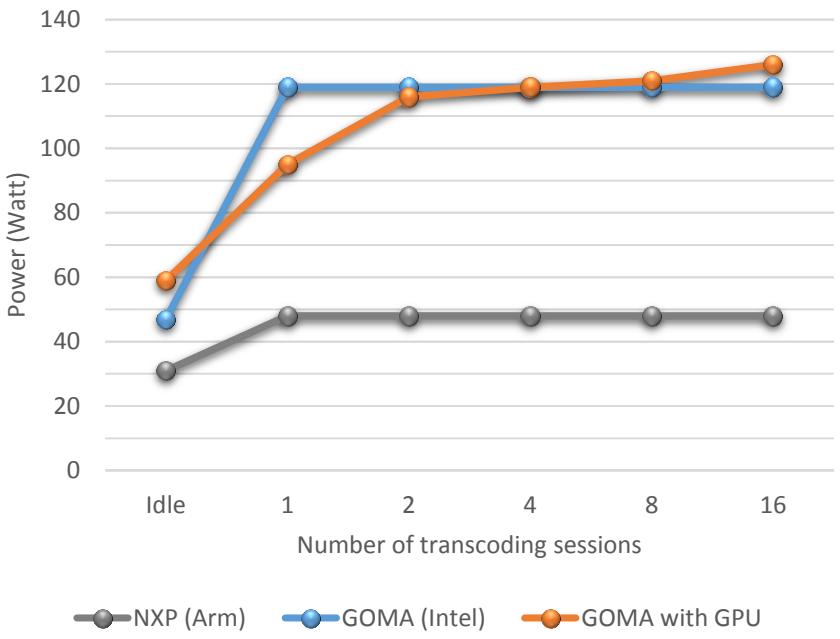


Figure 28: Power consumption (lower is better) of the three HW platforms running the H.264 HD1080 encoding multi-session transcoding tests

As expected, the NXP micro-server exhibits the better results, starting from 31W in idle state and going to 48W when running the VTU application. It is worth noting that 9.5W of the 31W are dedicated to the fans that run always at maximum speed; if the NXP micro server would control appropriately the fan speed, its power consumption would improve significantly.

Regarding the behavior of the Intel platform, with and without GPU, the results could appear unexpected because there are conditions in which the presence of the GPU does not increase the total power consumption, but rather decreases it. This can be explained considering Figure 29, showing the percentage of CPU and GPU resources occupied during transcoding (dotted lines). Let us consider, for example, the situation with one session. Intel platform consumes 119W without GPU, and 95W with GPU. However, while w/o GPU the CPU is running at 100% (not reported in the figures), with GPU the transcoding requires only 20% of CPU and GPU resources (as in Figure 29). This is the reason why the power consumption with GPU is in some case even better (lower) than the ones w/o GPU.

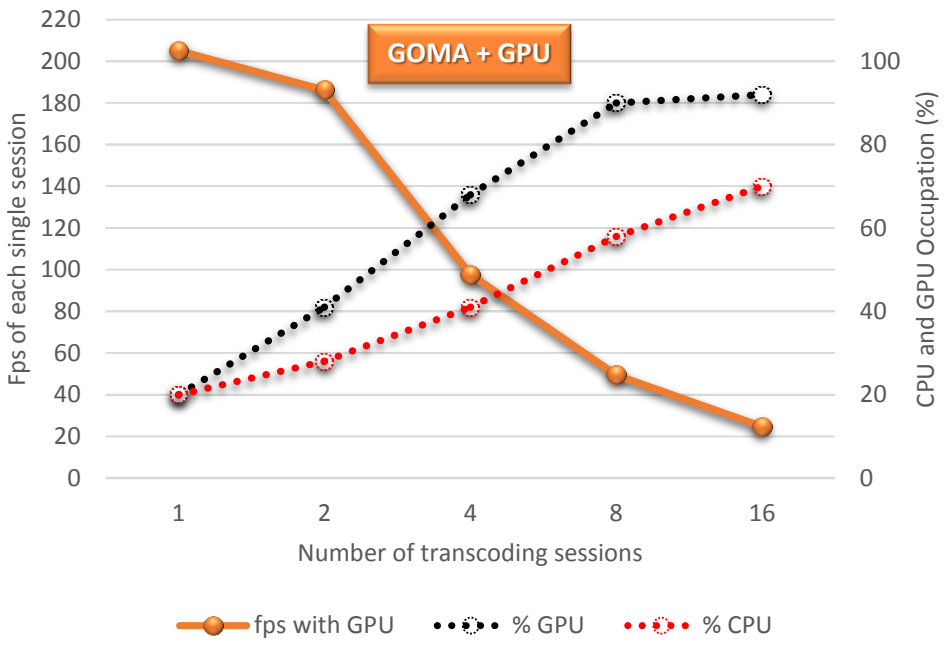


Figure 29: H.264 HD1080 encoding with GPU in multi-session transcoding tests (performance related to each single session) with percentage of CPU and GPU resources utilization.

Comparing the efficiency of the two solutions in term of performance/watt (Figure 30), we see another important advantage of using the GPU. In fact, in case of 16 concurrent sessions (H.264-HD1080), the gain in efficiency using Intel + GPU is 5.4x compared to Intel (SW-only). This could be, in some way, expected. Less obvious is the greater efficiency of Intel respect to Arm (for this particular application).

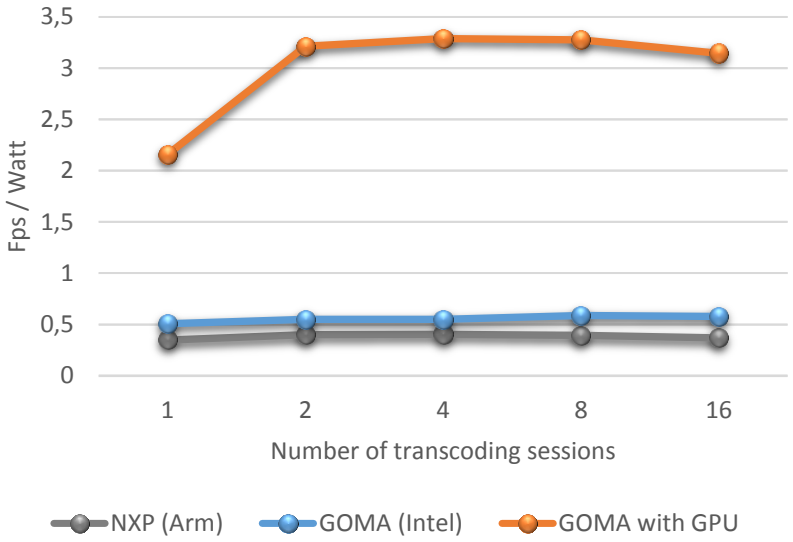


Figure 30: Efficiency of the three HW platforms (expressed in performance/power) for H.264 HD1080 encoding in multi-session transcoding tests (higher is better)

Finally, Figure 31 shows the performance of the VTU when the transcoding is made starting from the same input file used so far, but converting it in a H.265 format. This transcoding operation is much heavier than the previous one (H.264), as you can see from the reduced performance respect to Figure 25. We did not report the Arm performance because are too low to be considered. Intel performance are very low too, and the reduction in performance (and efficiency) respect to the GPU reaches 10x.

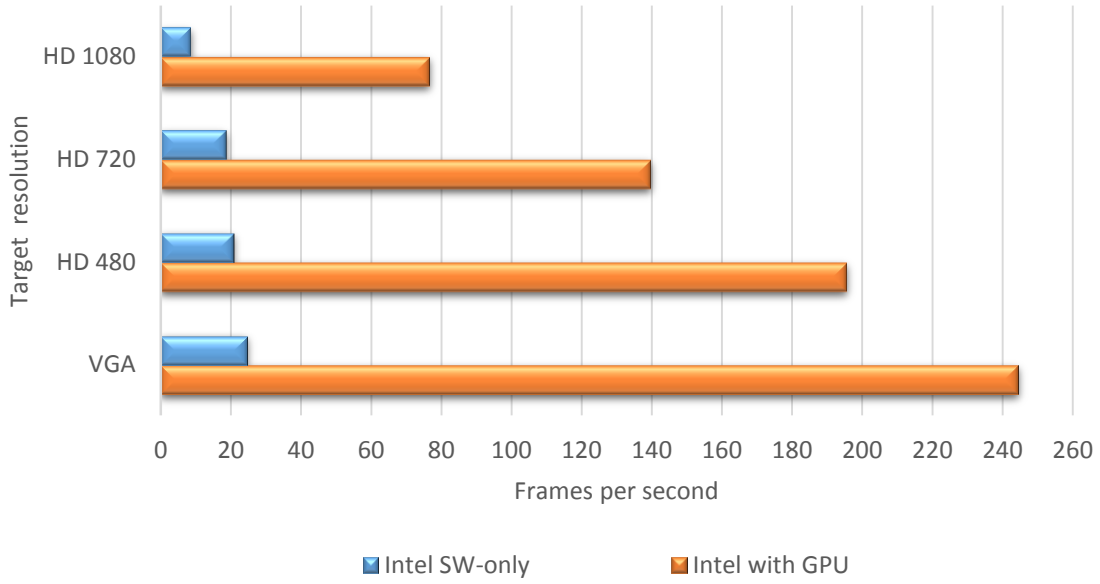


Figure 31: H.265 single session encoding performance (higher is better) measured on two different HW platforms, for different output resolution

The costs of the GOMA and NXP micro servers for various reasons cannot be taken as a reference. In fact, NXP micro server is not commercially available as a product, but only as a test platform. On the other hand, GOMA server is a rugged platform, suitable for transportation, integrating screen and keyboard, that uses advanced cooling strategy (liquid cooling) not usually provided in standard server. For these reasons, speaking of costs, it seems more correct to refer to equivalent standard servers equipped with the same CPU, RAM, etc.

We can then estimate a cost around 2.500,00 Euros for Intel platform and 2.000,00 Euros for ARM. The GPU is around 1.000,00 Euros. Considering the performance/cost ratio, the presence of a GPU appears always economically convenient (for the VTU application).

Summarizing, the reported results led us to the following conclusions:

- For computing intensive VNFs, as VTU, the Intel CPU (Xeon E5-2630v3 2.4GHz, 8 Core) performs much better than ARM (LS2085A ARM A57 1.8 GHz, 8 Core).
- The ARM platform is not able to run VTU in real-time scenarios.

- In general the ARM platform does not appear suited to running VTU due to poor performance.
- The addition of a GPU (NVIDIA Quadro M4000) to Intel CPU results in a boosts of performance in a range between 5 and 10 times (5x -10x).
- The NXP (ARM) micro server provides lower power consumption (and requires much lower space) but this is not enough to guarantee efficiency. Actually, for VTU application, Intel is more efficient than ARM.
- The best efficiency (performance/watt) is achieved by far using the GPU.
- Considering the performance/cost ratio, the presence of a GPU appears always economically convenient (for the VTU application).

6 Conclusion

This deliverable, starting from the specification of the Light DC architecture as reported in the deliverables D4.1 [2], and D4.2 [3], describes the Light DC prototype implemented for hosting the SW virtualization platform, the VNFs related to the virtualized small cells, as well as the service VNFs to be hosted in the SESAME PoC.

The physical architecture of the selected PoC, the different types of micro server nodes (ARM- and x86-*based*), as well as the interfaces, networking and storage resources are shown.

Furthermore, the virtualization platform related activities, specifically developed for the NXP micro-server prototype are depicted, as well as the activities performed for enabling the ARM-*based* node to be managed by OpenStack.

The integration activities include the development and testing of a service Virtual Network Function that provides video transcoding service at the edge (VTU) on the hybrid (ARM/x86) Light DC.

The VTU service VNF is used to get some preliminary KPIs regarding performance, power consumption and cost, considering the Arm and x86 platforms; the case of HW acceleration (GPU) usage is also investigated.

The contents of this deliverable will be used as “input” to the WP7, where will be included the outcomes of the WP3, WP5 and WP6 development activities and prototypes.

7 References

[1] Deliverable D2.2: "Overall System architecture and Interfaces – First Iteration", H2020 SESAME project, *April 2016*.

[2] Deliverable D4.1: "Light DC architecture design", H2020 SESAME Project, *June 2016*.

[3] Deliverable D4.2: "Virtualization extensions for acceleration of Light DC capabilities", H2020 SESAME project, *December 2016*.

[4] NXP web page: *http://www.nxp.com/products/microcontrollers-and-processors/arm-processors/qoriq-arm-processors:QORIQ-ARM*.

[5] ARM web page: *https://www.arm.com/products/system-ip/amba-specifications.php*.

[6] Ceph web page: *http://ceph.com/*.

[7] OpenStack wiki for Cinder: *https://wiki.openstack.org/wiki/Cinder*.

[8] Wiki page: *https://en.wikipedia.org/wiki/Logical_Volume_Manager_(Linux)*.

[9] OpenStack wiki for Nova component: *https://wiki.openstack.org/wiki/Nova*

[10] OpenStack wiki for Neutron component: *https://wiki.openstack.org/wiki/Neutron*.

[11] OpenStack page for Cinder: *http://docs.openstack.org/developer/cinder/devref/architecture.html*.

[12] European Telecommunications Standards Institute (ETSI) (2014): "NFV Management and Orchestration - An Overview", GS NFV-MAN 001 v1.1.1. European Telecommunications Standards Institute, Sophia-Antipolis, France.

[13] NVIDIA NVENC Programming Guide [Online]. Available from: *https://developer.nvidia.com/nvenc-programming-guide 2017.03.27*

[14] N.M. Cheung, X.Fan, O.C. Au, and M.C. Kung (2010, March): Video Coding on Multicore Graphics Processors. *IEEE Signal Processing Magazine*, *27*(2), pp.79-89.

[15] P. Paglierani (2015): "High Performance Computing and Network Function Virtualization: A major challenge towards network programmability". In Proceedings of the 2015 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom), Constanta, pp.137-141.

[16] WebM Video Hardware RTLs [Online]. Available from: *https://www.webmproject.org/hardware/ 2017.03.27* .

[17] Comi, P., Secondo-Crosta, P. Beccari, M., et *al.* (2016): "Hardware-accelerated high-resolution video coding in Virtual Network Functions". In Proceedings of the European Conference on Networks and Communications 2016 (EuCNC-2016), pp.32-36, Athens, Greece, *June 27-30, 2016*.

[18] Paglierani, P., Grossi, G., Pedersini, F., and Petrini, A. (2016): "GPU-based VP8 encoding: Performance in native and virtualized environments". In Proceedings of the International Conference on Telecommunications and Multimedia 2016 (TEMU-2016), pp.1-5, Heraklion, Greece, *July 25-27, 2016*.

.