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 D4.3

Techniques for efficient VNF Deployment with relevant VIM extensions, Evaluation framework

Document Number:	H2020-5GPPP-GA No.671596/*WP4/D4.3/30.06.2017*
Contractual Date of Delivery:	*30.06.2017* (M24)
Editor:	Dr. Ioannis Giannoulakis – National Centre for Scientific Research "Demokritos" (NCSRD)
Work-package:	*WP4*
Distribution / Type:	Public (PU) / Report (R)
Version:	1.0
Total Number of Pages:	60
File:	*SESAME_Deliverable 4.3_v1.0_Final*

Abstract

The SESAME EU-*funded* project targets innovations around three central elements in the wider 5G context: (i) The placement of network intelligence and applications in the network edge through Network Functions Virtualization (NFV) and Edge Cloud Computing; (ii) the substantial evolution of the Small Cell concept, already mainstream in 4G but expected to deliver its full potential in the challenging high dense 5G scenarios, *and*; (iii) the consolidation of multi-tenancy in communications infrastructures, allowing several operators/service providers to engage in new sharing models of both access capacity and edge computing capabilities.

Deliverable D4.3 describes an overview and the VNFs applied in the SESAME platform in further details. The aim has been to "highlight" the specific features of the SESAME testbed by fulfilling most key Small Cell- and NFV-*related* requirements and, *at the same time*, allowing implementation in a relatively short timeframe with reasonable technical complexity.

For NFV to take full advantage of cloud technologies and realize the cost savings plus agile delivery models that NFV promises, the VNFs need to be architected for a cloud infrastructure — simply porting the functions to a virtual machine is not enough. Even if a VNF has been properly architected for the cloud, the orchestration and management layers need to have the capabilities to effectively manage the full life cycle activities of the VNF without incurring avoidable ongoing development costs at the orchestration layer. The support of a VNF adapter simplifies how VNF life cycle actions are triggered, and provides great flexibility in how VNF suppliers and service providers implement their NFV cloud management and orchestration.

5G-PPP Disclaimer:
This *Deliverable* has been prepared by the 5G Initiative, via an inter 5G-PPP project collaboration. As such, the contents represent the consensus achieved between the contributors to the report and do not claim to be the opinion of any specific participant organisation in the 5G-PPP initiative or any individual member organisation of the 5G-Infrastructure Association.

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	24.05.2017	Initial draft by NCSRD	I. Giannoulakis
	24.05.2017	Initial contributions by NCSRD	I. Giannoulakis
0.2	16.06.2017	Contributions by i2CAT	P. Khodashenas
		Contributions by ATOS	E. Jimeno
		Contributions by ITL	A. Albanese
		Contributions by FLE	M. Wilson
		Contributions by ORION	E. Kafetzakis
0.3	16.06.2017	Merged and circulated for review	I. Giannoulakis
0.4	26.06.2017	Overall review of the document by i2CAT	P. Khodashenas
0.5	27.06.2017	Overall review of the document by SMNET	A. Dardamanis
1.0	29.06.2017	Full conceptual and editorial review by OTE – Document ready for submission to the Commission	I. Chochliouros

Contributors

First Name	Last Name	Partner	Email
Ioannis	Giannoulakis	NCRSD	giannoul@iit.demokritos.gr
Pouria	Sayyad Khodashenas	i2CAT	pouria.khodashenas@i2cat.net
Shuaib	Siddiqui	i2CAT	shuaib.siddiqui@i2cat.net
August	Betzler	i2CAT	august.betzler@i2cat.net
Daniel	Camps	i2CAT	daniel.camps@i2cat.net
Antonino	Albanese	ITL	antonino.albanese@italtel.com
Elisa	Jimeno	ATOS	elisa.jimeno@atos.net
Emmanouil	Kafetzakis	ORION	mkafetz@orioninnovations.gr
Mick	Wilson	FLE	mick.wilson@uk.fujitsu.com
Hui (Julie)	Xiao	FLE	Hui.Xiao@uk.fujitsu.com
Athanassios	Dardamanis	SMNET	adardamanis@smart.net.gr
Ioannis	Chochliouros	OTE	ichochliouros@oteresearch.gr

Glossary

Acronym	Explanation
4G	Fourth Generation of Mobile Communications
5G	Fifth Generation of Mobile Communications
aka	also known as
AOMedia	Alliance for Open Media
AP	Application Protocol
API	Application Programming Interface
AR	Augmented Reality
AV1	AOMedia Video 1
AVC	Advanced Video Coding
BS	Base Station
BSD	Berkeley Software Distribution
BSS	Business Support System
CCTV	Closed Circuit Television
CE	Crowded Event
CESC	Cloud-enabled Small Cell
CESCM	Cloud Enabled Small Cell Manager
CN	Core Network
CPE	Customer Premises Equipment
CPU	Central Processing Unit
DB	Database
DBaaS	Database as a Service
DC	Data Centre
DDoS	Distributed Denial of Service
DHCP	Dynamic Host Configuration Protocol
DHS	Department of Homeland Security
DHS	Dynamic Host Support
DNAT	Distribution Network Address Translator
DNS	Domain Name Server
DNS	Domain Name System
DoS	Denial of Service
DPI	Deep Packet Inspection
DynDNS	Dynamic DNS
E2E	End-to-End
EMS	Element Management System
eNB	Evolved Node B
EPC	Evolved Packet Core
ETSI	European Telecommunications Standards Institute
EU	European Union
EuCNC	European Conference on Networks and Communications
FE	Functional Entity
FF	Fast Forward
FFMPEG, ffmpeg	Fast forward Moving Pictures Expert Group
FG	Forwarding Graph
FP7	7th Framework Programme
FPGA	Field Programmable Gate Array
FW	FireWall

FWaaS	FireWall as a Service
GA	Grant Agreement
GPL	General Public License
GPU	Graphics Processing Unit
GS	Group Specification
GUI	Graphical User Interface
GW	Gateway
H2020	Horizon 2020
HDD	Hard Disk Drive
HeNB	Home eNodeB
HEVC	High Efficiency Video Coding
HOT	Heat Orchestration Template
HTP	Hyper-text Transfer Protocol
HTTP	Hyper-text Transmission Protocol
HW	Hardware
HWA	Hardware Accelerator
I/O, i/o	Input/Output
IaaS	Infrastructure-as-a-Service
ICSI	International Computer Science Institute
ICT	Information and Communication Technology
ID, id	Identifier
IDS	Intrusion Detection System
IETF	Internet Engineering Task Force
IoT	Internet of Things
IP	Internet Protocol
IPS	Intrusion Prevention System
IPsec	Internet Protocol Security
IPv4	IP version 4
IPv6	IP version 6
IT	Information Technology
ITU	International Telecommunication Union
ITU-T	International Telecommunication Union - Telecommunication Standardization Sector
JSON	JavaScript Object Notation
KNN, kNN	k-Nearest Neighbors
KPI	Key Performance Indicator
LAN	Local Area Network
LB	Load Balancer
LBaaS	Load Balancer as a Service
Light DC	Light Data Centre
LTE	Long Term Evolution
µs	micro server
MAC	Medium Access Control
MANO	Management and Orchestration
MEC	Mobile Edge Computing
MPEG	Moving Pictures Expert Group
NAT	Network Address Translator
NBAR	Network Based Application Recognition
NE	Network Edge
NETCONF	Network Configuration Protocol

NF	Network Function
NFP	Network Forwarding Path
NFV	Network Functions Virtualisation
NFVI	Network Functions Virtualisation Infrastructure
NFVO	NFV Orchestrator
Nmap	Network Mapper
NMS	Network Management System
NS	Network Service
NSD	NS Descriptor
OASIS	Organization for the Advancement of Structured Information Standards
OISF	Open Information Security Foundation
Open CV	OpenSource Computer Vision
OS	Operating System
OSI	Open Systems Interconnection
OSS	Operations Support System
OVS	Open virtual Switch
P2P, p2p	Peer-to-Peer
PBFS	Packet Based per Flow State
PCI	Peripheral Component Interconnect
PGW	Packet Gateway
PHP	Hypertext Preprocessor
PNF	Physical Network Function
PNFD	PNF Descriptor
PoC	Proof of Concept
PoP	Point of Presence
PPP	Public-Private Partnership
PPTP	Point-to-Point Tunneling Protocol
QoS	Quality of Service
RAM	Random Access Memory
RAN	Radio Access Network
REST	Representational State Transfer
RFC	Request for Comments
RIA	Research and Innovation Action
RPC	Remote Procedure Call
RRD, rrd	Round-Robin Database
RTP	Real-Time Transport Protocol
SC	Small Cell
SDK	Software Development Kit
SDN	Software-defined Networking
SIFT	Scale Invariant Feature Transform
SLA	Service Level Agreement
SME	Small- and Medium-sized Enterprise
SMS	Short Message Service
SNAT	Source Network Address Translator
SOTA	State-of-the-Art
SP	Service Provider
SPI	Stateful Packet Inspection
SSH	Secure Shell
SSL	Secure Sockets Layer
SVG	Scalable Vector Graphics

SVM	Support Vector Machine
SW	Software
SWA	Software Architecture
TCP	Transmission Control Protocol
TEMU	Telecommunications and Multimedia
ToC	Table of Contents
TOSCA	Topology and Orchestration Specification for Cloud Applications
TU	Transcoding Unit
UDP	User Datagram Protocol
UE	User Equipment
UHD	Ultra High Definition
UI	User Interface
URI	Universal Resource Identifier
UTM	Unified Threat Management
VA	Video Analytics
vDPI	virtual Deep Packet Inspection
VIM	Virtual Infrastructure Manager
VM	Virtual Machine
VNF	Virtual Network Function
VNFD	VNF Descriptor
VNFFG	VNF Forwarding Graph
VNFFGD	VNF Forwarding Graph Descriptor
VNFM	VNF Manager
vNIC	virtual Network Interface Controller
vPGW	virtual Packet Gateway
VSF	Varnish security FireWall
vTU	virtual Transcoding Unit
VTU	Video Transcoding Unit
WAN	Wide Area Network
WP	Work Package
XaaS	All-as-a-Service
XML	Extensible Markup Language
XPATH, XPath	XML Path Language
YANG	Yet Another Next Generation

Table of Contents

List of Figures

List of Tables

N/A

1 Introduction

Virtualized Network Functions or VNFs are the software (SW) realisations of the various network functions that can be deployed on a Network Functions Virtualization Infrastructure (NFVI) and are assigned a specific network operation. In this way, a VNF handles a specific network functionality that runs on one -or more- virtual machines on top of the virtualized hardware (HW) infrastructure. If the network functions on which services depend upon are virtualized, they can be deployed very quickly, potentially "anywhere in the network", without the delays associated with physical hardware procurement and installation. Moreover, when VNFs are spun up on top of the cloud infrastructure, they take advantage of the instant availability of virtual resources.

VNFs can be instantiated by using cloud application lifecycle management techniques that automate deployment, thus reducing operational cost and time-to-launch. NFV taps into the agile application lifecycle management opportunity that cloud delivers to achieve operators' service agility goals. Normally, VNFs need further configuration to fulfil customer-specific services. A cloud deployment template can help to turn up a new VNF instance, so that it is operationally ready, but such a template does not have an API that can configure the VNF dynamically, on demand. This is where runtime configuration approaches are needed to complete the fulfilment process. NETCONF[1]/YANG[2] or TOSCA[3] are examples of such approaches[4]

---

[1] The Network Configuration Protocol (NETCONF) is a network management protocol developed and standardized by the IETF. It was developed in the NETCONF working group and published in *December 2006* as RFC 4741 and later revised in *June 2011* and published as RFC 6241. The NETCONF protocol specification is an Internet Standards Track document. NETCONF provides mechanisms to install, manipulate, and delete the configuration of network devices. Its operations are realized on top of a simple Remote Procedure Call (RPC) layer. The NETCONF protocol uses an Extensible Markup Language (XML) based data encoding for the configuration data as well as the protocol messages. The protocol messages are exchanged on top of a secure transport protocol. The NETCONF protocol has been implemented in network devices such as routers and switches by some major equipment vendors. One particular strength of NETCONF is its support for robust configuration change using transactions involving a number of devices. For more details see, *among others*: https://en.wikipedia.org/wiki/NETCONF

[2] YANG is a data modeling language for the definition of data sent over the network configuration protocol (NETCONF). The name is an acronym for "*Yet Another Next Generation*". The YANG data modeling language was developed by the NETMOD working group in the IETF and was published as RFC 6020 in *October 2010*. The data modeling language can be used to model both configuration data as well as state data of network elements. Furthermore, YANG can be used to define the format of event notifications emitted by network elements and it allows data modelers to define the signature of remote procedure calls that can be invoked on network elements via the NETCONF protocol. The language, being protocol independent, can then be converted into any encoding format, e.g. XML or JSON, that the network configuration protocol supports. YANG is a modular language representing data structures in an XML-free format. The data modeling language comes with a number of built-in data types. Additional application specific data types can be derived from the built-in data types. More complex reusable data structures can be represented as groupings. YANG data models can use XPATH expressions to define constraints on the elements of a YANG data model. Also see, *inter alia*: https://en.wikipedia.org/wiki/YANG

[3] Topology and Orchestration Specification for Cloud Applications (TOSCA), is an OASIS standard language to describe a topology of cloud based web services, their components, relationships, and the processes that manage them. The TOSCA standard includes specifications to describe processes that create or modify web services. Also see, *for example*: https://en.wikipedia.org/wiki/OASIS_TOSCA

[4] See, *for example*, the context proposed by Y. Weinreb (*2016, February*) in: "*Combining TOSCA and Netconf/YANG to Deploy Network-Centric Services in NFV and SDN Architectures*" (as proposed in:

that deliver a programmatic configuration experience and that can be harnessed to support the automated lifecycle management goals of NFV.

In the NFV environment of SESAME, a VNF takes on the responsibility of handling specific network functions that run on one -or more- Virtual Machines (VMs) on top of the hardware networking infrastructure, i.e., the Light Data Center (Light DC) which is a Mobile Edge Computing (MEC) enabler in the context of SESAME. The VNFs that support SESAME infrastructure have been selected so as to describe better the potential of the final platform. These network functions have been designed and developed with the mindset that the service is the critical outcome, and it needs to be closely monitored and their usage to be optimized.

In addition, the breakthroughs introduced by the SESAME project as regarding the NFV concepts may cause a paradigm shift for many stakeholders including network operators, solution vendors, service integrators, providers, and service users. However, several issues continue to hinder the management of NFV infrastructure (NFVI) resources. Therefore, software network appliances need to be deployed in the NFVI with various hardware and software requirements, quality of service (QoS) provisioning levels, and/or negotiated service level agreements (SLAs).

A telecommunications infrastructure based on cloud technologies must have virtual network functions (VNFs) whose architecture leverages the specific SESAME cloud capabilities. The readiness and timing of those re-architected VNFs depend on technical, performance and business considerations. This deliverable "outlines" the architectural principles for the VNFs used in the project and it also addresses the issue of how to simplify the interface between the VNF/EMS and the VNFM for lifecycle management.

---

http://cloudify.co/2016/02/17/mwc-cloud-sdn-mano-nfv-netconf-yang-tosca-vnf-openstack.html). Also see the discussion proposed by F. Khan (2016, June) in: "*NETCONF/YANG and TOSCA- Friends or Enemies?"* (http://www.telecomlighthouse.com/netconfyang-and-tosca-friends-or-enemies/).

1.1 Definitions of Terms and SESAME concepts

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

- **Small Cell (SC):** LTE micro eNodeB which its radio functionality does not change fundamentally in the context of SESAME.
- **Execution infrastructure, micro server:** Specific hardware that provides processing power, memory, storage and networking capabilities to the SC.
- **CESC (Cloud Enabled Small Cell):** An intelligent entity composed of two, optionally co-located, network-connected physical devices: a Small Cell Physical Network Function and a micro server (μS) platform, which forms 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 virtualized execution infrastructure that provides computational, networking and storage resources to the Small Cell.
- **CESC Manager (CESCM):** 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). That includes ETSI MANO[5] VIM, VNFM and NFVO.

[5] For more details about the ETSI MANO see, *inter-alia*: *ETSI GS NFV-MAN 001 v1.1.1 (2014-12): Network Functions Virtualization (NFV); Management and Orchestration*. Available at: http://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_NFV-MAN001v010101p.pdf

2 VNF development guideline

Larger service providers "push" for re-architecting network functions, including decomposition, to take full advantage of cloud techniques. Smaller service providers (SPs) are currently concerned with leveraging virtualization and cloud for compact solutions that are more easily deployed and maintained. For them, the fact that the VNFs are re-architected is secondary.

Overall, the feedback is that VNFs need to be re-architected into discrete functions that can be scaled independent of the rest of the VNF and independent of the overall solution. VNFs also need to be agnostic of the underlying hardware and to have the ability to take advantage of emerging cloud technologies and high degrees of automation. The re-architecting of network functions is not an easy task, but there are few key architectural principles that can help VNF suppliers to make network functions future ready, so they can leverage emerging and future cloud technologies.

A key architectural shift in the VNF development is to decompose each network function into small VNF components that are easily instantiated, based on capacity requirements. For appropriate VNFs, such as vDPI, the functional components must be data-less and consist primarily of the VNF's logic. VNF suppliers use standards-*based* interfaces and avoid proprietary interfaces to enable multi-vendor deployments; or do so as soon as relevant standards permit. Similarly, to avoid duplicate development and the associated maintenance costs, they should employ reusable software assets and/or open-source components, *where possible*.

The architecture principles presented in this deliverable provide a solid foundation for the proposed network functions (see e.g., Figure 1).

However, to "capture" the benefits of a multi-supplier cloud and to allow VNF and MANO suppliers to evolve their architectures over time, there also needs to be a simpler way for the VNFM to execute lifecycle activities.

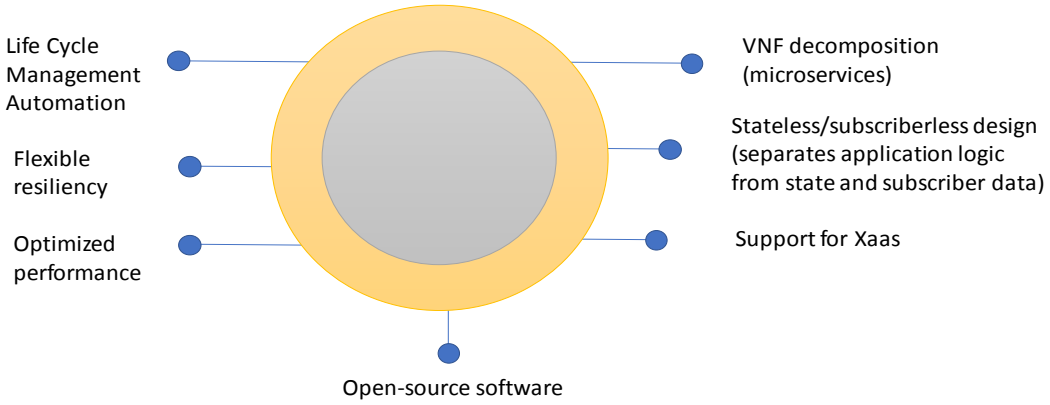

Figure 1: Key architectural items for cloud-ready VNFs

Therefore, a simplified lifecycle management interface between the VNF/EMS and VNFM, which uses RESTful APIs[6] has been studied. It has a common structure or basic primitives that enable the VNFM to more easily execute lifecycle actions when compared to the existing situation with its heavier reliance on pair-wise integration between VNF/EMS and VNFM in multi-supplier networks.

[6] For more details see, *for example*: *http://www.restapitutorial.com/*

2.1 Network functions decomposition

One goal of traditional network functions is to optimize the use of hardware resources; this was accomplished by putting as much functionality as possible on a single host. With advances in processing power, this approach makes it difficult to use available resources efficiently and scaling of network function capacity tends to be at the host level, creating unused capacity.

For cloud deployments, VNF suppliers need to optimize their VNFs by decomposing each function into the right type of building blocks (see Figure 2). Although this deliverable does not explore microservice specifications, decomposition should be guided by the performance and business requirements. The functions may evolve to microservices that can be scaled easily, as network demand increases or decreases. Microservices are a software architecture style in which complex VNFs are composed of small, independent processes communicating with each other by using language-agnostic APIs. These services are small, highly decoupled and focus on performing a small task, thereby facilitating a modular approach to system-building.

The compute resources required each decomposed functions should be kept as small as possible while still providing reasonable capacity increments; this will result in fewer compute resources being used, thus leading to improvement in the overall compute resource utilization and cost reduction.

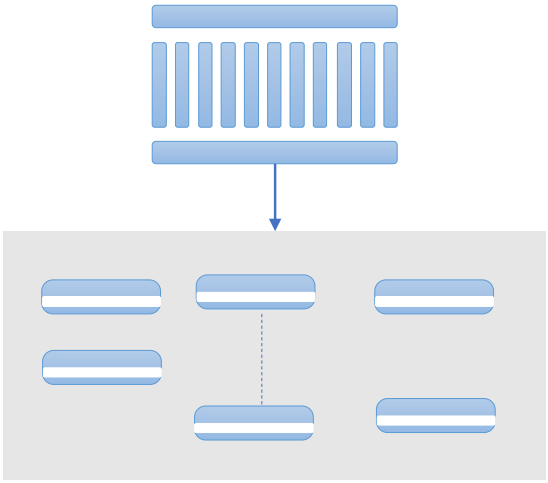

Figure 2: Decomposing network functions into small functional components

2.2 Design for and use commoditized cloud functions

A key concept -and goal- of the cloud infrastructure is to provide a common set of services that are centrally managed and used in the cloud's underlying infrastructure. These services include databases, firewalls and load balancers, and are typically referred to as "X as a service" (XaaS).

Examples include DataBase as a Service[7] (DBaaS), FireWall as a Service[8] (FWaaS) and Load Balancer as a Service[9] (LBaaS).

Because the network functions are architected to "run" in the cloud, VNF programmers should ensure that the interfaces to these types of services are open and clearly defined, so that commoditized functions can be supported. The actual use of commoditized functions needs to be determined on a case-by-case basis to ensure that the performance and capacity characteristics of a network function are maintained, but VNF suppliers should architect the network functions assuming that commoditized functions will be used.

A network function also needs to clearly define the performance characteristics and feature functionality needed by the XaaS, including required throughput, required tail latency, alignment with interface specifications and required feature capabilities. It is also recommended that the network function provides the capability to monitor the quality of service (QoS) that an XaaS is providing, and that it provides alarms if the QoS is not being met.

2.3 Design for media routing and packet processing performance

VNFs that deliver real-time characteristics of media processing have a "tight" coupling between the VNF software and the underlying hardware. Traditionally, media processing has also required dedicated or specialized hardware to "meet" the performance requirements. For VNFs that require real-time -or near-realtime- media routing capabilities, the move to the cloud has been questionable because of the performance impact that virtualization introduces.

For VNFs with critical requirements on the performance optimization, it is necessary to architect them, independently from the way that the cloud infrastructure delivers the packets to the VNFs. The packet delivery method/technology should be defined and configured when a VNF is instantiated in the cloud infrastructure. This requires the VNF to clearly define the technologies required for specific capacity and to indicate the capacity that can be expected, if the technology is not available.

Also, VNFs that deliver IP routing can greatly vary in their data plane (or user plane) and control plane requirements, so when architecting them, VNF suppliers should consider what matters the most for their specific use case.

When it comes to IP routing VNFs, the data plane requirement is about high-performance packet processing: taking the packets from the network into a virtual machine (VM), further processing by the VM, and forwarding of packets to the network. There are different techniques to improve this flow while maintaining full packet processing capabilities and delivering high throughput. Careful design of these VNFs must take all of their critical data and user plane requirements into consideration. VNFs that require data plane performance must be designed and optimized to deliver the necessary performance and scale. VNF suppliers and NFVI and

[7] For more details also see, *inter-alia*: https://www.stratoscale.com/blog/dbaas/what-is-database-as-a-service/. Also see: https://docs.openstack.org/mitaka/config-reference/database-service.html

[8] For more details also see, *inter-alia*: https://docs.openstack.org/newton/networking-guide/fwaas.html

[9] For more details also see, *inter-alia*: https://docs.openstack.org/mitaka/networking-guide/config-lbaas.html

MANO suppliers must recognize these problems and work to deliver solutions with the required performance to support commercial deployment.

2.4 Lifecycle automation for flexible delivery methods

ETSI has defined a MANO architecture that is intended to address how an overall NFV solution is managed. As shown in Figure 3, the MANO architecture consists of three major functional blocks and associated interfaces. At a high level, the NFV orchestrator (NFVO) supports the network services (NSs) lifecycle management. In fact, a NS is a composition of several VNFs working together on a common goal. Next, the VNFM provides lifecycle management of a single VNF instance, VNF general configuration, and event reporting between the NFVI and the EMS/network management system (NMS). The VIM manages the underlying compute, storage and networking resources. While all these implementations are valid solutions, they tend to address a bigger problem (e.g. lifecycle automation and QoS management on wide deployments and over multiple points of presence) rather than interaction with a single VNF.

From a single VNF perspective, a simpler approach is to provide a direct RESTful APIs and/or a basic GUI to interact VNF. The VNF adapter should be a passive function that provides a simpler interface between the VNF/EMS and the user (either a human being or a software) for lifecycle management actions such as instantiation, scaling, migration, termination and software update. These primitives or APIs are delivered by the VNF supplier.

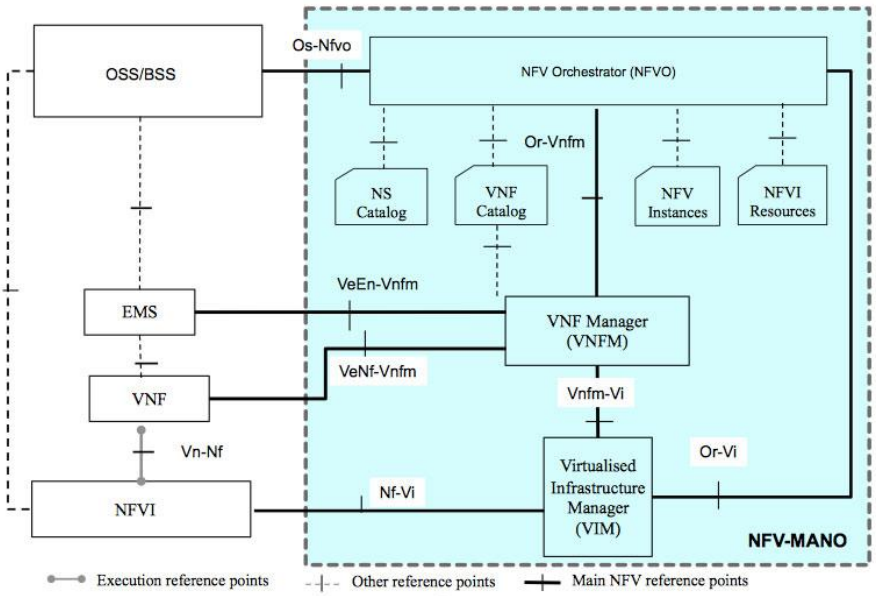

Figure 3: ETSI NFV MANO Architecture

2.5 Deployment and operation of a VNF

Figure 4 shows a typical cycle of deploying and operating steps for a VNF process. The process has a cyclic nature because the task does not end with a successful deployment. The condition of the infrastructure might change over time due to expansion, replacement, or simply aging. The VNF itself is dynamic and changes over time through events such as scaling out, migration and software updates. All these constant changes require the validation and verification processes to be repeated on a regular basis and hence, the need for automation.

Only through automation can the process become repeatable and provide meaningful results. Performing the process by using automatic tools, removes uncertainties by eliminating the human factor. It ensures no aspect is neglected and makes the task effortless, meaning it is more likely to be repeated whenever required, without undue burden.

The techniques and tools described in this deliverable can make the VNF deployment and lifecycle management much more predictable, thus resulting in quicker time-to-market and less wasted effort. Through our deep historical understanding of VNFs and the work of onboarding a wide variety of very different VNFs, we have developed both the expertise and the required tools to successfully repeat this process [1].

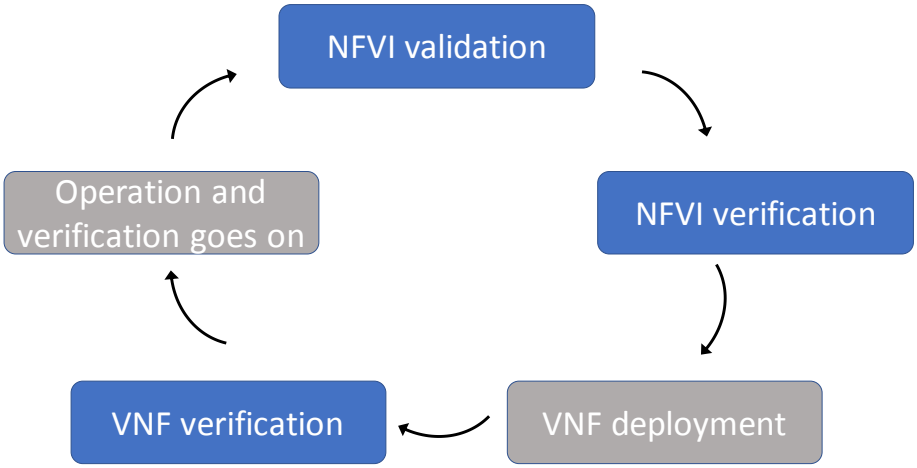

Figure 4: VNF deployment and operation cycle

2.6 VIM architecture

The VIM interacts with the infrastructure layer in order to manage and configure the virtualization infrastructure. It provides an API-*driven* orchestrator layer to manipulate the virtual compute, storage and networking resources, so they can execute appropriately for the different VNFs.

In the last recent years, the architecture of OpenStack[10] has grown in every release, starting with a modular architecture, built with independent components, with services for managing

[10] For more details see: *https://www.openstack.org/*

compute resources pools (Nova[11]), and object storage system (Swift[12]). The IaaS layer has been extended with extra services, based on a flexible framework and API-*driven*, that offers means to integrate both to vendors and solution providers, their compute, network and storage-infrastructure plugins best suited to the environment.

OpenStack implementation has been selected as SESAME VIM software platform, as it provides compelling solution for many challenges in the research computing for delivering flexible infrastructure. It also offers high level of stability, performance, level of support and possible extensions with new features for the NFV support. The real value lies in its APIs that reinforce the trunk code of the implemented functions, so that they provide appropriate execution environments for different VNFs.

The modular collection of cloud services that SESAME project offers are (see e.g., Figure 5):

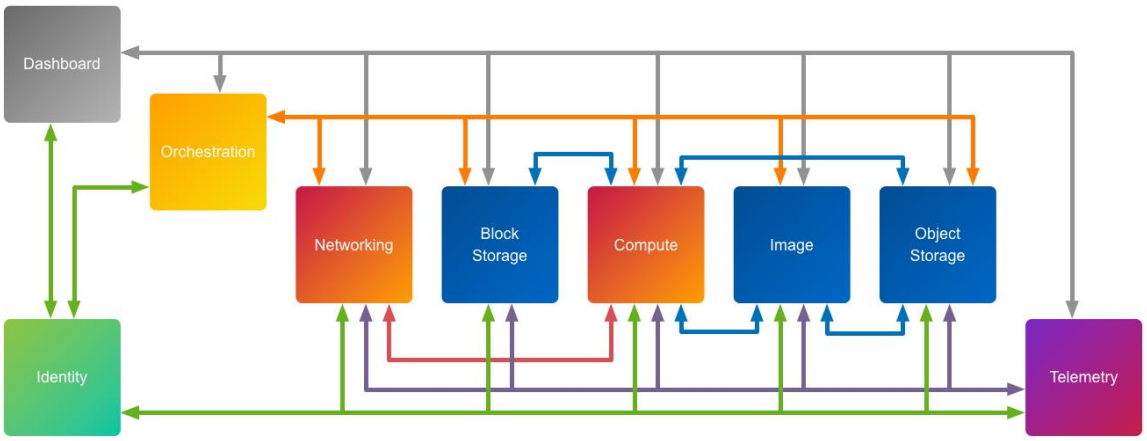

Figure 5: OpenStack Architecture

- Compute: Nova – which is responsible to instantiate and maintain operations for compute services, such as scheduling, and on-demand initiation of VNFs, as well as the removal of these services.
- Object Storage: Swift – this provides multi-tenant object storage with inherent replication and automatic scaling.
- Image Services: Glance[13] – which process, stores and retrieves VM disk images and corresponding metadata for actual image files using Swift.
- Network Management: Neutron[14] – which provides networking mechanism required in an environment as a service to other OpenStack components (such as Nova). It handles network interface by creating and attaching the virtual switch port to the vNIC of the

[11] For more details see: *https://wiki.openstack.org/wiki/Nova#Nova*

[12] The OpenStack Object Store project, known as Swift, offers cloud storage software so that you can store and retrieve lots of data with a simple API. It is built for scale and optimized for durability, availability, and concurrency across the entire data set. Swift is ideal for storing unstructured data that can grow without bound. More details can be found at: *https://wiki.openstack.org/wiki/Swift*

[13] For more details see: *https://wiki.openstack.org/wiki/Glance*

[14] Neutron is an OpenStack project to provide "networking as a service" between interface devices (e.g., vNICs) managed by other OpenStack services (e.g., Nova). For more details see: *https://wiki.openstack.org/wiki/Neutron*

VM, assigning the IP address, configuring network overlay for tenant isolation, and providing network configuration for bare-metal provisioned servers.

- Block Storage: Cinder[15] – which provides persistent block storage for instances, such as a VM, running on the OpenStack platform. It also enables block storage devices and volume snapshot for persistent usage.
- WebUI: Horizon[16] – a web-*based* service portal that provides usability improvements for tenants and administrators to control and configuring access for provisioning services.
- Authentication: Keystone[17] – which provides identity management for OpenStack services tracking and authentication of users, as well as authorization of the services requested.
- Cloud Template: Heat[18] – which provides orchestration of services using Heat Orchestration Templates (HOTs), which describe a given cloud application to build the entire cloud setup of OpenStack.
- Telemetry: Ceilometer[19]– it collects cloud usage and performance measurements. Its targets monitoring and metering, but it can be expandable to collect data for other needs.

The transformation to VNF services and deployment of the SESAME use cases needs an API framework, and this feature makes OpenStack the suitable "candidate". However, to ensure service availability and support for provisioning of NFV services, some extensions to the set of APIs are needed, and the concept "as a whole" needs to be adapted. OpenStack provides a basis to develop the necessary plugins to support the provisioning and deployment of a large number of VNFs; and also to develop extensions to API services, that are needed to support the scenarios.

The SESAME deliverables that are to follow in the context of the WP5 (" *Infrastructure Virtualization and Management*") and of the WP6 ("*Orchestration and service level management*") will describe in more details the implementation of the Virtual Infrastructure Manager.

[15] For more details see: *https://wiki.openstack.org/wiki/Cinder*

[16] Horizon is the canonical implementation of OpenStack's Dashboard, which provides a web based user interface to OpenStack services including Nova, Swift, Keystone, etc. For more details see: *https://wiki.openstack.org/wiki/Horizon*

[17] Keystone is an OpenStack service that provides API client authentication, service discovery, and distributed multi-tenant authorization by implementing OpenStack's Identity API. For more details see: *https://docs.openstack.org/developer/keystone/*

[18] Heat is the main project in the OpenStack Orchestration program. It implements an orchestration engine to launch multiple composite cloud applications based on templates in the form of text files that can be treated like code. Further details can be found at: *https://wiki.openstack.org/wiki/Heat*

[19] For more details see: *https://wiki.openstack.org/wiki/Telemetry*

3 Selected SESAME VNFs

3.1 vDPI

The virtual Deep Packet Inspection[20] (vDPI) is designed to analyze in real-time network traffic, to recognize specific applications and to categorize each traffic flow according to its service. As this operation generates significant workload, it is highly recommended to be deployed in a medium to large VM flavor for efficient traffic processing. In the following, we describe the techniques which were employed.

The Linux network stack, upon which the vDPI has been developed, is commonly used as a basis for cloud networking solutions. Its primary goal is the provisioning of a general-purpose network stack for a fully functional operating system, rather than a stack that is specifically designed for high packet processing performance.

The required packet analysis and flow information processes consume a significant amount of computing resources, as all network flows and most of their packets need to be processed deeper than protocol headers, in many cases at payload level, to provide accurate and precise packet identification of network traffic. The vDPI is primarily used to identify network traffic profiles for various security, or network management purposes. Additionally, vDPI analyzes IP traffic from Layers 2 to 7, including headers and data protocol structures together with the payload of the packet's message. This information is used to identify the various application protocols and traffic flows. Collection of this information can be used to provide a traffic classification of the network being monitored. Various commercial DPI solutions exist from companies such as WindRiver [1], PACE [3] and Qosmos [4]. The Qosmos DPI offers a virtualized solution built on Qosmos' flagship product ixEngine[21], which has established itself as the *de facto* industry-standard DPI engine for developers of telecoms and enterprise solutions. An alternative free open-source solution is nDPI [5]. NDPI is an ntop-maintained superset of the popular OpenDPI library[22]. NDPI has performed better, compared to other open source DPI libraries and Cisco's NBAR[23], in extensive traffic recognition experiments performed in [6].

The DPI solution described here is built using the nDPI library. This approach uses only an indicative, small number of initial packets from each flow, in order to identify the payload

[20] For more information about the more generalized concept of DPI (Deep Packet inspection) see, *for example*: *https://en.wikipedia.org/wiki/Deep_packet_inspection*

[21] Qosmos ixEngine is a Software Development Kit (SDK) composed of software libraries and tools that are easily integrated into new or existing solutions. Developers benefit from market-leading IP flow parsing technology to accelerate the delivery of application aware solutions. Qosmos ixEngine can be used in all environments, that is: physical, virtualized and in SDN architecture. For more details see: *http://www.qosmos.com/products/deep-packet-inspection-engine/*

[22] Also see: *http://www.ntop.org/products/deep-packet-inspection/ndpi/*

[23] Applications in today's enterprise networks require different levels of service based upon business requirements. These requirements can be translated into network policies. Mission critical applications including ERP and workforce optimization applications can be intelligently identified and classified by using Network Based Application Recognition (NBAR). For more details also see: *http://www.cisco.com/c/en/us/products/ios-nx-os-software/network-based-application-recognition-nbar/index.html*

content and does not inspect each packet. In this respect, it follows a Packet Based per Flow State (PBFS). This method uses a table to "track" each session based on the 5-tuples (source address, destination address, source port, destination port, and the transport protocol) that is

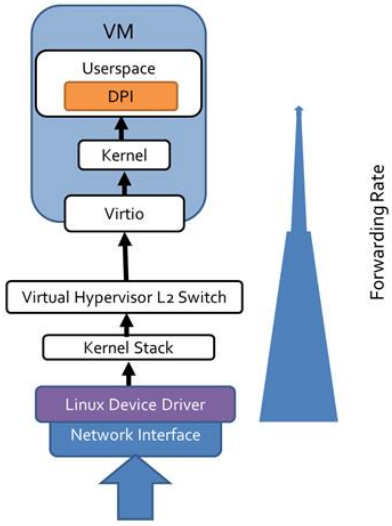

Figure 6: Virtual DPI VNF component architecture

maintained for each flow. The component architecture is shown in Figure 6.

Figure 7 provides an overview of the high-level Traffic Classification VNF architecture. The ORION's VNF comprises four VNF Components (VNFCs); namely the DPI VNFC, the Classification VNFC, the Analytics VNFC and the Repository VNFC. The role of each VNFC is explained briefly below.

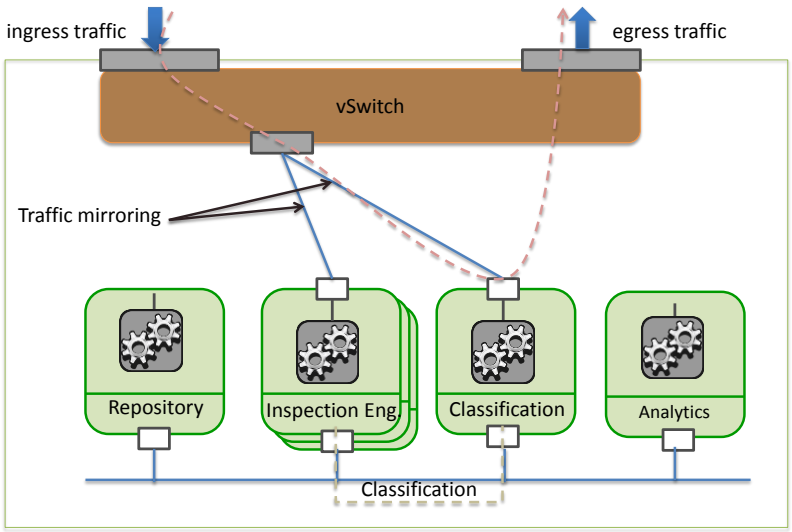

Figure 7: vDPI high level framework

Deep Packet Inspection Engine VNFC this component is responsible for the most processing intensive functionality of the VNF. The component implements the filtering and packet matching algorithms in order to support the traffic classification. This component supports a flow table (exploiting hashing algorithms for fast indexing of flows) and an inspection engine for traffic classification. As soon as a new flow has been identified by the DPI engine, the flow registration will be updated with the appropriate information and communicated to the Classification VNFC.

Classification VNFC - this component is responsible for the routing and data packet forwarding process. The component accepts the incoming traffic, consults the Flow Table for classification information for each incoming flow and then applies QoS policies accordingly. It is noted that data traffic is mirrored to the DPI VNFC in order for DPI and Classification VNFCs to work in parallel. In case that the VNFCs are not executed on the same host, mirroring the traffic might include additional overhead. Thus, other more efficient VNF graphs are implemented.

Repository VNFC this component is responsible for storing the raw information of the monitored and classified flows. This component is implemented as a round robin database[24] that will be possible to store monitoring and classification information for a specific time window.

Analytics VNFC this component is responsible for the analysis and statistical processing of the classified traffic information. The component exposes a Web-*based* API, allowing the access to the information from the VNF tenants. This component fulfils the requirement for passive operation of the VNF, allowing collection of historical traffic information by the customer.

3.2 vTU

The Video Transcoding Unit (vTU) VNF provides some basic functionalities, which can be used to building video content sharing services for the users. In particular, the VTU provides video and audio transcoding functions, together with local storage capabilities of pre-recorded audio/video files.

The services provided by the vTU can be accessed through a web-service-*based* interface, available from any browser. Through the vTU, users can originate or receive live video streams. In particular, they can stream an originated video content to the vTU, or upload it as a pre-recorded file. Other users can receive this live content, or download it at a later time. If the originated video is shared among users in real-time, the VTU concurrently saves it to a distributed storage system, for successive content sharing. In the VTU, a common storage area is also dedicated to system managers that can upload contents to be offered to all the users. Such functionalities can be used to build enhanced video services, well-suited for the highly debated Crowded Event (CE) use cases [1]. In this scenario, a high number of users concentrates in a small area for a short time, typically ranging from few hours to a week. Well-known examples of CEs are sporting matches or concerts at a stadium, congresses or exhibitions hosted by dedicated venues, international events spread over a university campus or even an entire city.

[24] For more informative details see, *inter-alia*: *https://en.wikipedia.org/wiki/RRDtool*

3.2.1 Architecture

In this paragraph, the overall vTU architecture is summarized and briefly discussed.

3.2.1.1 Functional Description

The core task of the Video Transcoding Unit is to convert video streams from one video format to another. Though the VTU can also process audio signals, video encoding processes are by far the most compute intensive workloads it can perform; therefore, in the following, the discussion will be mainly focused on video processing functionalities.

A simplified functional description of the VTU is reported in Figure 8. Depending on the application, the source video stream could originate from a file within a storage facility, as well as coming in form of a packetized network stream from a user or from another VNF. Moreover, the requested transcoding could be mono-directional, as in applications like video stream distribution, or bi-directional, like in videoconferencing applications.

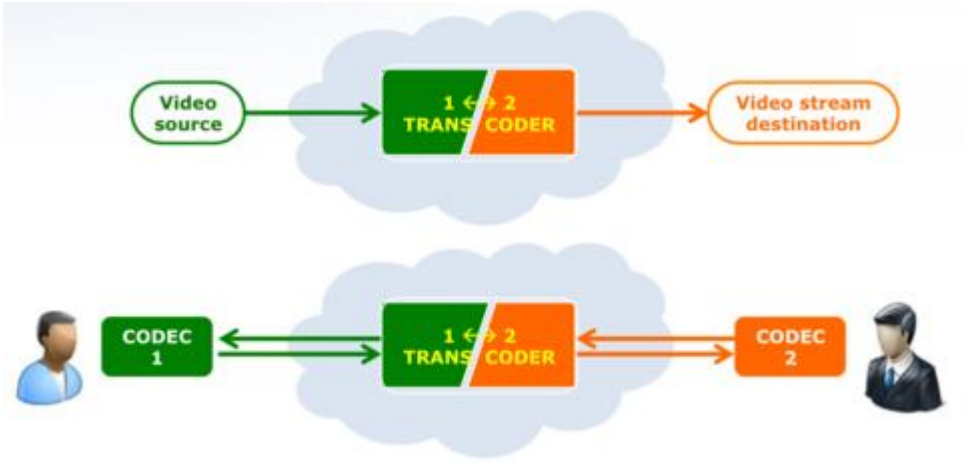

Figure 8: Functional description of the vTU

Differently from traditional video content delivery systems, in which users only have the role of content consumers, the VTU also allows users to originate video contents. Each video file sent by a user and received by the VTU, in real-time or as a pre-recorded file, is firstly temporarily saved in a local storage system; then, it is processed, so as to obtain copies at different resolutions, to "match" the various media capabilities of the users' devices. The output files at the desired resolutions are finally stored in a distributed storage system, available for immediate streaming by any instance of the VTU VNF.

The most convenient architecture for the VTU VNF is a modular architecture, in which the necessary encoding and decoding functionalities are deployed as plug-in within a "container" Unit taking care for all the communication, synchronization and interfacing aspects. With the aim of finding a convenient approach for the development of the VTU architecture, an investigation has been carried out about the state-of-the-art (SOTA) of any available software framework that could be usefully employed as starting point for this architecture. This investigation has identified **ffmpeg**, the open-source audio-video library under Linux environments (*https://ffmpeg.org/*), as the "best choice" for the basic platform for the VTU, as it

is open-source, modular and customizable, and contains most of the encoding/decoding plug-ins that this VNF could need.

To define the functionalities that best fit to the needs of the target applications for the VTU, a survey has been carried out, searching for the most diffused video formats that should be present as encoding/decoding facilities in the vTU. This analysis has shown that the following video formats are of primary interest:

- ITU-T H.264[25] (aka AVC[26])
- ITU-T H.265[27] (aka HEVC[28])
- Google's VP8[29]
- Google's VP9[30].

The ffmpg framework can provide software-*based* encoding and decoding capabilities for the above coding schemes, and of many other algorithms.

However, 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. Though, the advantages of HW acceleration can be so preponderant, in

[25] ITU-T Recommendation H.264: "*Advanced Video Coding for Generic Audio-Visual Services*". For more details also see: *https://www.itu.int/rec/T-REC-H.264*

[26] H.264 or MPEG-4 Part 10, Advanced Video Coding (MPEG-4 AVC) is a block-oriented motion-compensation-*based* video compression standard. As of 2014 it was one of the most commonly used formats for the recording, compression, and distribution of video content. The intent of the H.264/AVC project was to create a standard capable of providing good video quality at substantially lower bit rates than previous standards (i.e., half or less the bit rate of MPEG-2, H.263, or MPEG-4 Part 2), without increasing the complexity of design so much that it would be impractical or excessively expensive to implement. For more details see, *among others*: *https://en.wikipedia.org/wiki/H.264/MPEG-4_AVC*

[27] ITU-T Recommendation H.265: "*High Efficiency Video Coding*". For more details also see*:* *https://www.itu.int/rec/T-REC-H.265*

[28] High Efficiency Video Coding (HEVC), also known as H.265 and MPEG-H Part 2, is a video compression standard, one of several potential successors to the widely used AVC (H.264 or MPEG-4 Part 10). In comparison to AVC, HEVC offers about double the data compression ratio at the same level of video quality, or substantially improved video quality at the same bit rate. It supports resolutions up to 8192×4320, including 8K UHD. For more details see, *for example*: *https://en.wikipedia.org/wiki/High_Efficiency_Video_Coding*

[29] VP8 is an open and royalty free video compression format owned by Google and created by On2 technologies as a successor to VP7. More details can be found, *for example*, at: *https://en.wikipedia.org/wiki/VP8*

[30] VP9 is an open and royalty free video coding format developed by Google. VP9 is a successor to VP8 and competes mainly with MPEG's High Efficiency Coding (HEVC/H.265). At first, VP9 was mainly used on Google's popular video platform *YouTube*. The emergence of the Alliance for Open Media (AOMedia), and its support for the ongoing development of the successor AV1, led to growing interest in the format. In contrast to HEVC, VP9 support is common among web browsers. For more details see, *among others*: *https://en.wikipedia.org/wiki/VP9*

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[31] 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. The fully HW approach can provide higher compute performance than cooperative CPU-GPU algorithms. Though, the HW approach very often lacks the flexibility in service management needed by 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 NVENC[32] encoder[33] for the H.264 and H.265 encoding scheme. Also, the CPU-GPU cooperative approach described in [8], [9] can be used for the Google Open Source VP8 encoder.

Thus, in SESAME three versions of the VTU VNF have been developed:

- SW-only VTU, for ARM processors[34]
- SW-only VTU, for x.86[35] processors
- HW-accelerated VTU, for x.86 processors with NVidia Graphics Processing Unit[36] (GPU).

Based on the needed performance, the power consumption budget and the expected costs, an operator can choose the most appropriate platform for the VTU services to be deployed. Details about Key Performance Indicators (KPIs) that can be for platform selection are briefly discusses

---

[31] Motion estimation is the process of determining motion-vectors that describe the transformation from one 2D image to another; usually from adjacent frames in a video sequence. It is an ill-posed problem as the motion is in three dimensions but the images are a projection of the 3D scene onto a 2D plane. For more details see, *for example*: *https://en.wikipedia.org/wiki/Motion_estimation*

[32] 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.

[33] 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*

[34] ARM is the industry's leading supplier of microprocessor technology, offering the widest range of microprocessor cores to address the performance, power and cost requirements for almost all application markets. Combining a vibrant ecosystem with over 1,000 partners delivering silicon, development tools and software, and with more than 90bn processors shipped, our technology is at the heart of a computing and connectivity revolution that is transforming the way people live and businesses operate. For more details also see: *https://www.arm.com/products/processors*

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

[36] Also see: *http://www.nvidia.com/object/gpu.html*

in the following Part 3.2.2, and analyzed in detail in SESAME Deliverable 4.4 ("*Light DC Prototype*").

3.2.1.2 vTU Interfaces

As described in the pervious sections, the VTU is a VNF that, during its normal operation, receives an input video stream, transcodes it and generates an output video stream in the new format. For each transcoding job, the VTU also needs to receive a proper job description, in which all necessary information are provided, like, *for instance*, information on the video format of the input stream and on the desired video format for the output stream, the identification and definition of the input/output data channels (e.g. IP addresses and ports, in case of network streams, or file ID within a storage facility, for file-generated streams).

For these reasons, the vTU needs, at its inner level, to communicate through three interfaces, as Figure 9 shows:

- **Input** port, receiving the video stream to transcode;
- **Output** port, producing the transcoded video stream;
- **Control** port, receiving the job descriptor and, implicitly, the command to start the job.

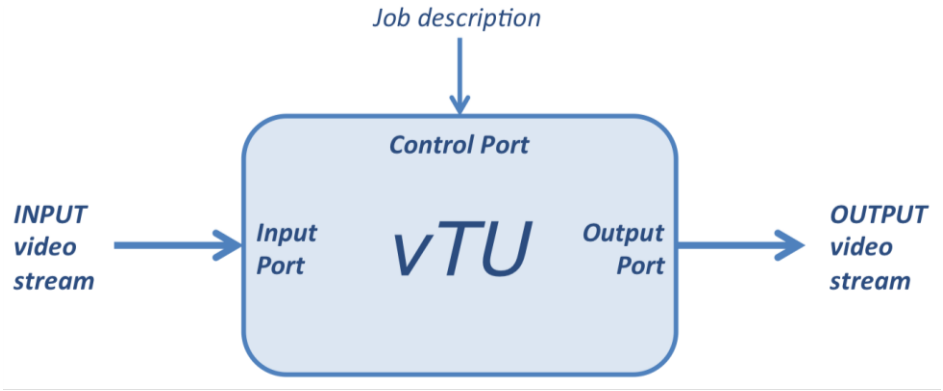

Figure 9: vTU low level interfaces

Through the Control interface, the VTU receives the job descriptor, which contains all necessary information to start the requested transcoding task. The starting command is implicit in the reception of the job description message: That is, when such a message is received on the Control port, the VTU starts listening at the Input port and begins the transcoding task, according to the received description, as soon as the first stream packets are received.

The format of the job description message is XML-based. The general structure of the message is shown in Figure 10. This format allows to define all necessary parameters, such as the desired input and output video formats and the I/O stream channels (files, in this case, but they could as well identify network channels sending/receiving RTP packets).

```
<vTU>
  <in>
    <local>
      <stream>test.y4m</stream>
    </local>
    <rstp>
      <ip/>
      <port/>
      <stream/>
      <timeout/>
    </rstp>
    <codec>
      <vcodec/>
      <acodec/>
    </codec>
  </in>
  <out>
    <local>
      <overwrite>y</overwrite>
      <stream>out_test.h264</stream>
    </local>
    <rstp>
      <ip/>
      <port/>
      <stream/>
      <timeout/>
    </rstp>
    <codec>
      <vcodec>h264</vcodec>
      <acodec/>
    </codec>
  </out>
</vTU>
```

Figure 10: XML structure of the vTU job description message.

To simplify the interactions of the users with the VTU, a web-service-*based* interface has also been developed. Through such an interface, a user can upload or download a pre-recorded audio/video content, or can start mono-directional or multi-directional streaming. A screenshot showing the initial VTU web-service-*based* interface is reported in Figure 11.

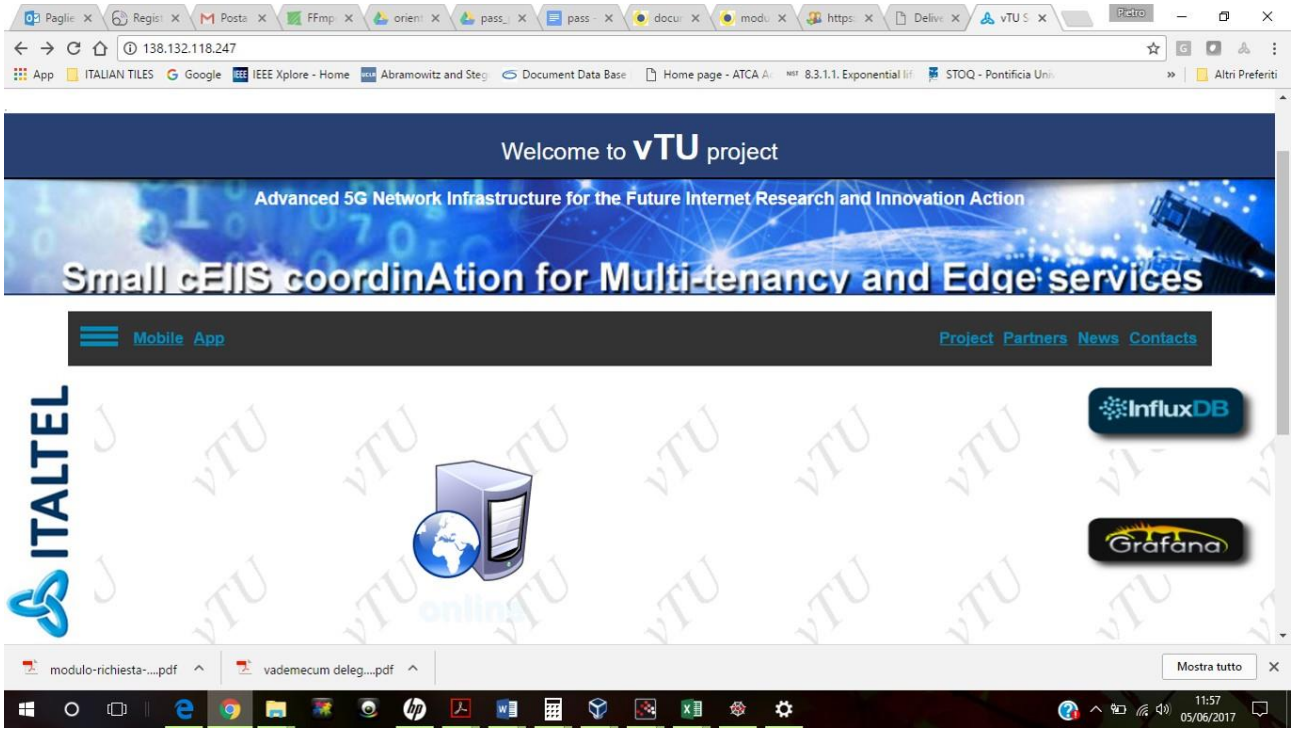

Figure 11: VTU web-service-*based* interface for the users.

For service management purposes, also a monitoring interface is provided. The initial view of such an interface is shown in Figure 12.

Figure 12: vTU monitoring Interface.

Such an interface is available only to system managers, and is accessed through an *ad hoc* web interface. To monitor the vTU status and performance, several metrics are generated and constantly sent at regular and modifiable basis to a user defined network address. Currently, monitoring data generated by the VTU server is redirected to a local instance of InfluxDB database[37], a popular open source database solution, oriented to collecting and managing time-series data [10]; then, such records are graphically arranged and visualized into a browser window by Grafana, a popular dashboard for displaying time series metrics [11]. The collected metrics are related, but not limited, to CPU, GPU (when available), memory, and disk usage; also encoding performances are captured and visualized, such as the number of transcoded frames per second.

3.2.1.3 Technologies

In the virtualization context, the problem of virtualizing a GPU is well-known, and can be stated as follows: a guest Virtual Machine (VM), running on a hardware platform provided with GPU-*based* accelerators, must be able to concurrently and independently access the GPU's, without incurring in security issues [12]. Many techniques to achieve GPU virtualization have been presented. However, all the proposed methods can be divided in two main categories, which are usually referred to as API remoting (also known as split driver model or driver

[37] For more information see: *https://www.influxdata.com/*

paravirtualization[38]) and PCI pass-through (also known as direct device assignment), *respectively*. In the VTU, the pass-through approach has been adopted. A description of this technique can be found in [12].

3.2.2 Dimensioning and Performance

In order to obtain a realistic assessment of the VTU performance, and provide meaningful Key Performance Indexes that can help in dimensioning the associated IT resources, many experiments and analysis have been carried out. Though, such aspects are discussed in the companion SESAME Deliverable 4.4; therefore, in the following we report only one simple experiment, which can give an "idea" of the carried out experiments and on the performance achievable by the VTU, when running on different platforms.

In Figure 13, the performance achieved by the VTU in transcoding a 1080p H.264 at 24 frame per second to the same encoding scheme, with different resolutions and for the three different platforms on which the VTU can run. The performance has been obtained in a single transcoding session (i.e., when only one video file to be transcoded is present in the system).

[38] In computing, paravirtualization is a virtualization technique that presents a software interface to virtual machines that is similar, but not identical to that of the underlying hardware. For further details also see: *https://en.wikipedia.org/wiki/Paravirtualization*

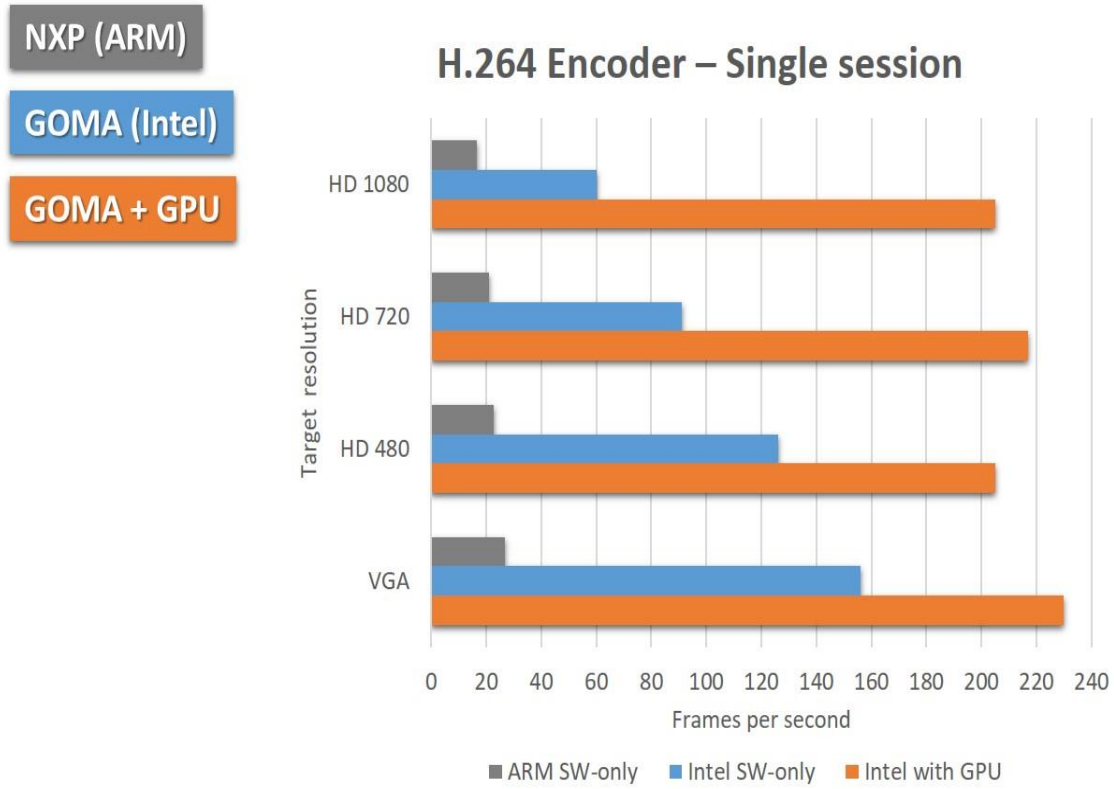

Figure 13: vTU performance for three different platforms.

As one can easily see, the performance obtained with HW acceleration are largely superior to those obtained with the CPU only, be it ARM or Intel. In particular, the ARM platform obtained quite low results compared to the Intel architecture.

3.3 Low Latency Video Analytics VNF

3.3.1 VNF Description

In the context of the evolution of the current communications infrastructure to the 5G systems, two video analytics-*based* VNFs (VA VNFs) have been designed and developed to investigate the two very import KPIs that 5G system aims to address, i.e. low end-to-end (E2E) latency and reduction in the backhaul consumption, which are also the major features that the SESAME platform intends to enable.

As we have begun to see, there are new emerging services and applications that require low end-to-end latencies and are often "linked" to the usage of Internet of Things (IoT) devices, such as tactile internet, augmented reality (AR) and real-time remote control of IoT devices (Smart IoT) as discussed in [13] and [14]. Therefore, 5G systems need to be designed to support the low latency requirements of the most challenging use cases, removing bottlenecks at the air interface, backhaul network and the data processing part. Meanwhile due to the explosive growth in the number of connected IoT devices in the years to come, 5G systems need to deal with the unprecedented amount of data generated by not only human communications but also the huge amount of IoT devices. As the current cellular network's architecture is centralized in that all the user equipment (UE) and IoT data need to be sent to remote data processing servers through operators' backhaul networks to reach operators' core networks (CNs) and the remote data processing servers, this puts "huge" pressure on the network infrastructure, thus resulting in degradation in the quality of service and triggering large investment requirements for carriers struggling to handle the load.

Corresponding to these challenging issues, the SESAME project offers a platform that enables cloud computing capabilities at small cell base stations (BSs) to fully exploit the benefits of Mobile Edge Computing (MEC). In order to evaluate the performance of the SESAME platform in terms of meeting stringent requirements on end-to-end latency and reducing the large volume of traffic's impacts on backhaul networks, real-time VA-*based* VNFs used for AR and Smart IoT services are considered to be appropriate for this purpose; this is due to the fact that streaming video data accounts for a major proportion of the mobile data traffic in this day and age, and services (such as AR and Smart IoT) based on real-time analytics of video data to derive processed video data or analysis results to react to real-time situations generally require tens of milliseconds' end-to-end latencies.

Through these two purposely designed VA VNFs, *on the one hand*, we are able to investigate if the SESAME platform, which features the integrated small cell and edge computing platform, has the capability to support the KPI requirements of such types of VNFs and measure the enabled performance for such types of VNFs; *on the other hand*, the deployment of the VA VNFs on the SESAME platform and the obtained measurement results both serve the purpose of identifying new requirements and ideas to further improve the SESAME platform.

The two real-time VA-*based* VNFs are intended for AR and Smart IoT services, *respectively*. The commonality between the two VA VNFs is that they both take the real-time video data streamed from video cameras, such as fixed CCTV cameras or mobile cameras mounted on smart phones, drones or robots, as the input data, and video analytics is applied to the video inputs in real time to carry out meaningful analyses and output the derived analysis results or processed video data to the relevant receiving devices. However, the difference is in the case of AR service where the

output data is the processed video data with additional information added to the input video data, whereas in the case of Smart IoT service, the output data is the derived video analysis results or instructions for autonomous IoT device control.

In the following figures (Figure 14 and Figure 15), the main functionalities supported by the two VA VNFs are described in more details, and some implementation guidelines associated with each VA VNF are provided.

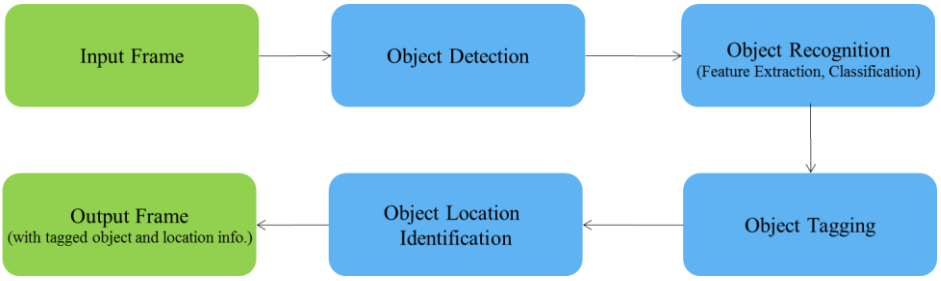

Figure 14: Functional blocks and workflow of the AR VA VNF

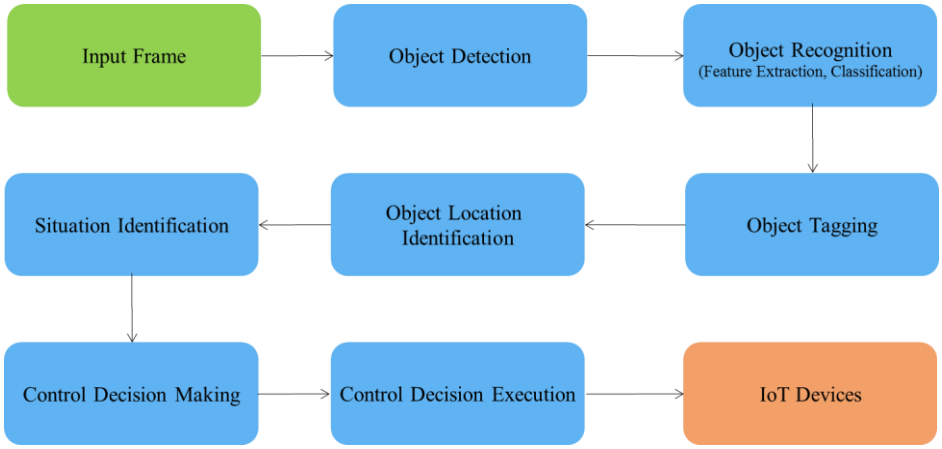

Figure 15: Functional blocks and workflow of the Smart IoT VA VNF

3.3.2 Main Functionalities and Implementation Guidelines

Both VA VNFs need to carry out real-time analysis on the received video streaming data frame by frame, in order to "track" a particular object of interest in the video frames. The object tracking function mainly consists of the following components, that is object detection, object recognition, object tagging and object location identification, which are the primary functionalities supported by the AR VA VNF. The Smart IoT VA VNF, on top of the object tracking function, has further functions such as situation identification, control decision making and control decision execution to realize the situation-aware remote control of IoT devices. All these

function components and functions are implemented in Python[39] using mainly computer vision and machine learning libraries.

The object tracking function commonly used by both AR VA VNF and Smart IoT VA VNF and its implementation guidelines are introduced in Section 3.3.2.1, and the additional functions implemented in the Smart IoT VA VNF are introduced in Section 3.3.2.2, as below.

3.3.2.1 Video Analytics VNF for Augmented Reality

The AR VA VNF is enabled by an object tracking function, which is composed of the following functional components, namely object detection, object recognition, object tagging and object location identification. Their functionalities and implementation guidelines are as follows:

Object Detection

Functionality: This function is used to identify the regions in a video frame that are likely to contain a particular object of interest. For example, if an object of interest is a moving object or a green object, this function is to identify all the image regions in a video frame that contain such kind of objects.

Implementation guidelines: Depending on the features that an object of interest owns, different kinds of object detection techniques can be applied.

If the features of an object are describable by humans such as its shape, colour, being static or moving, then such features can be defined and used to threshold a video image to identify the potential object containing regions. The Open Source Computer Vision (OpenCV) library [15] provides an extensive set of real-time image processing and computer vision libraries to achieve these tasks, for example, the Background Subtraction method[40] can be used to identify all the foreground regions containing moving objects.

However, if the features of an object of interest are not describable by humans or difficult to be described in words, then some more generic object detection techniques can be applied, such as the Canny Edge Detection method[41] for detecting the edge of an object, and some generic feature detection-based methods, such as Harris Corner[42], SIFT[43] and Haar Feature[44]-*based*

[39] For more details see: https://www.python.org/psf/

[40] Background subtraction, also known as foreground detection, is a technique in the fields of image processing and computer vision wherein an image's foreground is extracted for further processing (object recognition etc.). Generally an image's regions of interest are objects (humans, cars, text etc.) in its foreground. After the stage of image preprocessing (which may include image denoising, post processing like morphology etc.) object localization is required which may make use of this technique. Background subtraction is a widely used approach for detecting moving objects in videos from static cameras. The rationale in the approach is that of detecting the moving objects from the difference between the current frame and a reference frame, often called "background image", or "background model". Background subtraction is mostly done if the image in question is a part of a video stream. Background subtraction provides important cues for numerous applications in computer vision, for example surveillance tracking or human poses estimation. For more details also see the information provided in: https://en.wikipedia.org/wiki/Background_subtraction

[41] The Canny edge detector is an edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images. It was developed by John F. Canny in 1986. Canny also produced a computational theory of edge detection explaining why the technique works. More informative data as well as related references can be found, *for example*, at: https://en.wikipedia.org/wiki/Canny_edge_detector

[42] For more informative details also see, *inter-alia*: https://en.wikipedia.org/wiki/Corner_detection

machine learning methods. Basically in the generic object detection case, there is no need for a human to define the features of an object of interest; the features of an object of interest are extracted automatically by the feature detection techniques and used to find the image regions of a video frame that match the identified features of the object. For example, to detect regions containing a book, a vehicle or a human face, all the regions in a video frame that contain a book, a vehicle or a human face are identified based on the generic features of a book, a vehicle or a human face.

Object Recognition

Functionality: This function is in charge of distinguishing if the object included in an object containing region is the particular object of interest. For example, when there are multiple image regions containing moving objects or books, this function is applied to identify which region contains the particular moving object or book that the service wants to track.

Implementation guidelines: This function is achieved by using machine learning techniques, i.e. a classifier for the particular object of interest is trained using a large amount of positive and negative training image data, and then the trained classifier is applied to each object containing region to filter out the regions without the desired object and identify the region that contains the particular object of interest. Many machine learning techniques are supported by OpenCV libraries, such as KNN[45], SVM[46], Harr Cascade Classifiers[47] ([15], [16]). For deeper feature extractions and more accurate object recognition, some deep learning techniques and libraries that are provided by deep learning software frameworks, such as Caffe [17] and TensorFlow can be employed [18].

Object Tagging

Functionality: This function is used to draw a contour of the identified object of interest in a video image and add a label to the contour to denote what the object of interest is.

Implementation guidelines: Depending on the type of contour that needs to be drawn, OpenCV provides a collection of methods [15] to find the bounding box, circle, ellipse etc. of an object region, such as bouningRect(), minEnclosingCircle(), and a selection of methods that can be used to draw the desired contours, such as rectangle(), circle(), ellipse(), drawContours(). In order to

---

[43] For more details and for more related information, see the context presented in: *http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html*

[44] Haar-like features are digital image features used in object recognition. They owe their name to their intuitive similarity with Haar wavelets and were used in the first real-time face detector. For further information see, *among others*: *https://en.wikipedia.org/wiki/Haar-like_features.*
Also see the discussion proposed in: Boggarapu, S., and Sreenivasu, T. (2013, January): "An Optimized Implementation of the Haar like Feature Based Object Detection". *International Journal of Innovative Research & Development 2*(1), pp.37-47.

[45] In pattern recognition, the *k*-nearest neighbors algorithm (*k*-NN or KNN) is a non-parametric method used for classification and regression. For more related information also see, *inter-alia*: *https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm*

[46] In machine learning, support vector machines (SVMs, also support vector networks) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Also see, *inter-alia*: *https://en.wikipedia.org/wiki/Support_vector_machine*

[47] Also see: *http://docs.opencv.org/trunk/d7/d8b/tutorial_py_face_detection.html*

add some object information around the contour area, OpenCV method such as putText() can be employed.

Object Location Identification

Functionality: This function is responsible for estimating the real-world location of the tracked object with respect to the location of the monitoring camera in real time. Assuming the real-world location and orientation of the monitoring camera is known to the VA VNFs, the tracked object's real-world location is estimated by using: the detected dimension and position of the object of interest in each video image; the monitoring camera's real-world location and orientation, and; the monitoring camera's specifications, such as focal length and sensor dimensions. The estimated location data of the tracked object is added onto each video frame that contains the object of interest and is provided to a receiving device of the AR service as additional information regarding the tracked object.

Implementation guidelines: There are two parts to the implementation of the object location identification algorithm.

The first part is to estimate the distance between the tracked object and the monitoring camera, which is achieved by utilizing the mathematical relationship between the real-world dimension of the object of interest and the detected dimension of the object in units of pixels.

The second part is to estimate the horizontal and vertical deviation angles of the real-world position of the tracked object with respect to the boresight of the monitoring camera, which is enabled by measuring by how many pixels the centroid of the tracked object deviates from the centre of a video image, horizontally and vertically. The horizontal and vertical deviations in units of pixels are mapped to the real-world distance deviations with respect to the boresight of the monitoring camera by using the mathematical relationship between the real-world dimension of the object and the detected dimension of the object in a video image, based on which the horizontal and vertical deviation angles can be calculated by using the estimated distance in the first step.

Finally, the real world location of an object of interest is derived by using the distance and the horizontal and vertical deviation angles. The aforementioned mathematical relationships are derived by using detected and actual dimensions of the tracked object, and the specifications of the monitoring camera (in terms of its sensor size, focal length).

3.3.2.2 Video Analytics VNF for Smart IoT

On top of the object tracking function introduced above, the Smart IoT VA VNF contains further functions to achieve smart control of IoT devices based on video analytics, which are termed situation identification, control decision making and control decision execution. There are different kinds of smart IoT controls to serve different purposes. In our case, we implemented a mechanism for smartly and remotely controlling the cameras to achieve the continuous tracking of an object when it moves across multiple cameras views. In the following, the additional functions implemented in the Smart IoT VA VNF are introduced, based on this specific smart IoT example.

Situation Identification

Functionality: This function is in charge of predicting the movement of an object of interest based on its past movement trajectories; based on which it detects the need for video feed switching or relevant camera reconfiguration.

Implementation guidelines: The situation identification function consists of two parts, namely the movement prediction part and the switching or reconfiguration need detection part. The prediction of the movement of an object of interest based on its past trajectories is twofold, that is: movement speed estimation and movement direction estimation. At a time sample t_i, the estimated speed for the next time interval T_i is the mean value of the instantaneous speeds of the object over the last N time samples, and the instantaneous speed at a time sample can be found by using the successive image frames of the video. The polynomial regression method offered by the NumPy library[48] is employed to find the mathematical relationship that can best fit the object's location data of the last N time samples, and the gradient of the line that best fits the data is used as the estimated movement direction of the object in the next time interval T_i .

The detection of the need for video feed switching or camera reconfiguration is based on the estimated time for the tracked object to move out of the boundary of the current monitoring camera's coverage. If the estimated time for the tracked object to leave the current camera's coverage is slightly larger than the required time budget to complete the video feeding switching or camera reconfiguration process, then the situation identification function makes the decision to trigger the control decision making process, which will then start to find the most appropriate camera or camera settings to enable the continuous tracking of the object of interest.

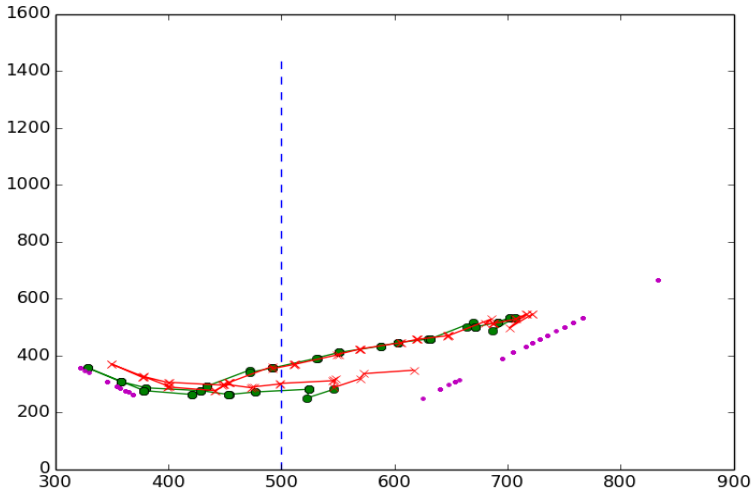

Figure 16: Tracked and predicted trajectories of a moving object

[48] NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. For more details see, *for example*: https://en.wikipedia.org/wiki/NumPy

Control Decision Making

Functionality: This is function is used to analyse the context data such as the available cameras' working status, capabilities and locations on the site, using which it determines the appropriate new video feed and/or new camera configurations to track the object of interest.

Implementation guidelines: Upon the detection of the need for video feed switching or camera reconfiguration, the context data of all the available cameras on the site is utilized to analyse what actions need to be taken in order to achieve the continuous tracking of the object of interest. Depending upon the current location of the object, as well as upon the predicted movement speed and direction of the object in some cases, the required action may be to switch the video feed to a camera that can provide coverage to the tracked object's future locations; in some cases, the required action may be to reconfigure the current monitoring camera's pan and tilt to follow the movement of the tracked object; in some other cases, the required action may be to switch the video feed to a different camera and configure the new camera's pan and tilt in order to capture the object of interest in its view field.

Control Decision Execution

Functionality: Based on the control decisions made, this function is in charge of performing the relevant actions to achieve the continuous tracking of the object of interest when it moves from the view field of one camera to another, such as performing video feed switching, establishing a new connection with the newly selected camera, and resetting the configurations of the relevant camera.

Implementation guidelines: The control instructions are sent to the relevant cameras through REST APIs, which may be a service request to establish a connection, or some pan and tilt parameters for a camera to adjust its configuration accordingly. The Python library such as urllib[49] can be employed to perform such tasks.

3.3.3 Example Use Cases

3.3.3.1 Spectators Entertainment Experience Improvement Use Case

At large sporting or musical event venues, such as stadiums, car racing or horse racing venues, or musical festival venues, live video feeds are received by the AR VA VNF from cameras mounted close to the playing field. The AR VA VNF can be used to offer spectators an object-tracking - *based* AR service. For example, after signing up for this service, spectators are able to track and watch the performance of any designated car, horse or player etc. on their mobile devices, meanwhile additional information about the designated player, car or horse is displayed on their mobile devices while they watch the game or performance in real time. The AR video viewed on the mobile devices are in sync with what the spectators see directly, that is the difference is smaller than what the human eyes can realize (e.g. less than 80 ms).

Such low latency is not only achieved by shortening the data transmission path, but also by optimizing the complete chain from the cameras to the mobile devices.

[49] See, *for example*: https://docs.python.org/3/library/urllib.html

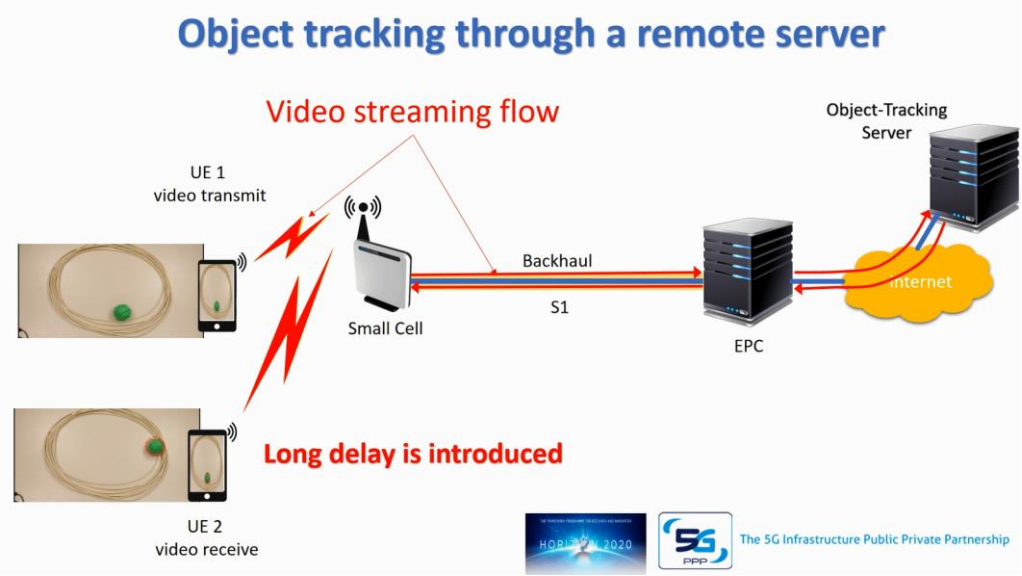

Figure 17: Architecture and performance of the object-tracking - *based* AR service hosted at a remote server

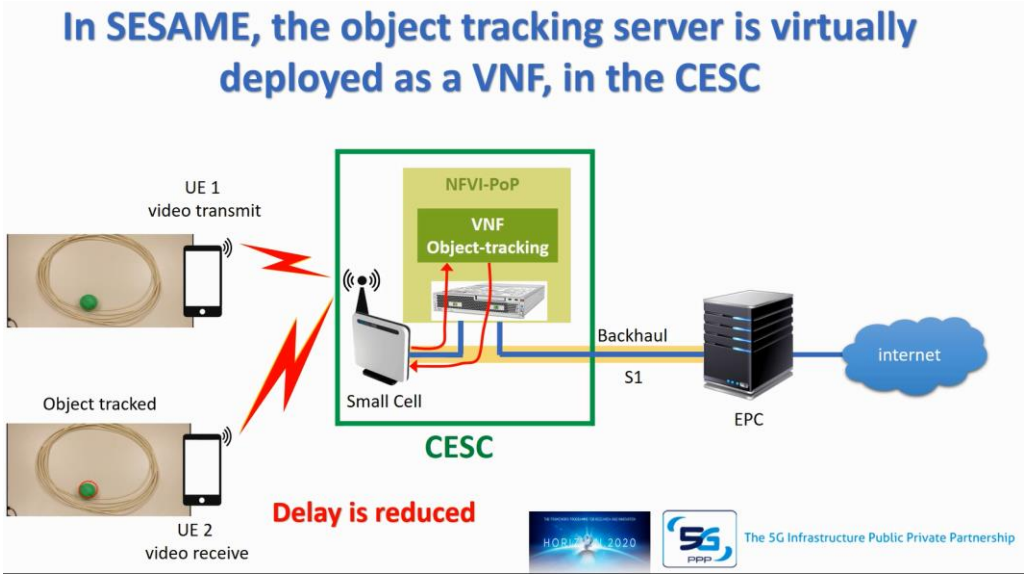

Figure 18: Architecture and performance of the object-tracking - *based* AR service hosted at a mobile network edge server

3.3.3.2 Criminal Case Solving Use Case

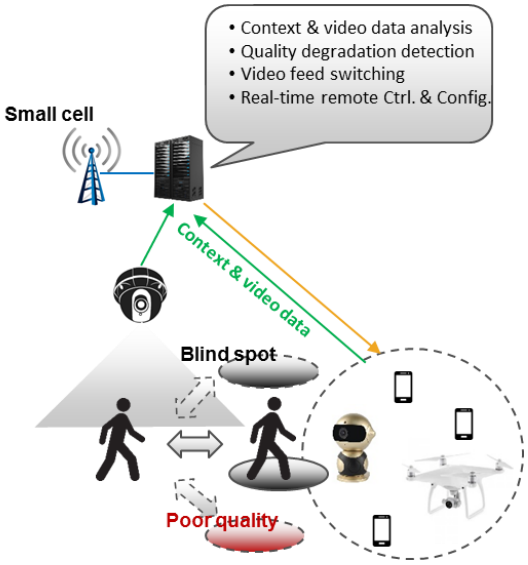

Figure 19: Illustration of the continuous object tracking service based on smart IoT control

Monitoring cameras are set up at the retail shops as well as on the streets to monitor the real-time situations. The live video feeds are sent to the Smart IoT VA VNF for it to carry out real-time video analytics, and the Smart IoT VA VNF can offer the retailers and police officers a criminal and lost item auto-tracking service based on the smart control of IoT devices (i.e. the monitoring cameras). Retailers, on the high streets or in shopping centers, can sign up for this object auto-tracking service, and after submitting the image of the tracked criminal and the lost item, the Smart IoT VA VNF is able to track petty crime criminals and criminal behaviors, e.g. concealment and transferring of stolen items, not only within the retailers' premises, but also outside the premises, enabling strong evidence collection regarding the criminals and the stolen items to win cases in court. Meanwhile, the real-time information regarding the criminals and the stolen items extracted from the tracking video data, such as the up-to-date images of a criminal and the stolen items and their real-world locations, can help police officers apprehend criminals on the run and chase back the stolen items. The general context is as shown in Figure 19.

The reduction in the IoT device response time to the real-time situation (i.e. from a few seconds to tens or hundreds of milliseconds) makes it possible to seamlessly track objects moving at high speeds (i.e. more than 100 km/h).

3.4 vFirewall

Firewalls are the systems which control the incoming and outgoing packets to and from the inner network. They provide security barrier against potential attacks coming from the Internet that can disrupt the services running in the inner network. The common setup for a firewall is depicted in Figure 20. Firewalls are divided into two types: Stateful and stateless firewall. In stateful firewalls, the connection is tracked by the firewall and the packets part of the tracked connection, are allowed to pass by. The stateful Firewall uses attributes in order to track the

traversing packets. These attributes include the source and destination IP addresses, port numbers and sequence numbers which are also known as state of the connection.

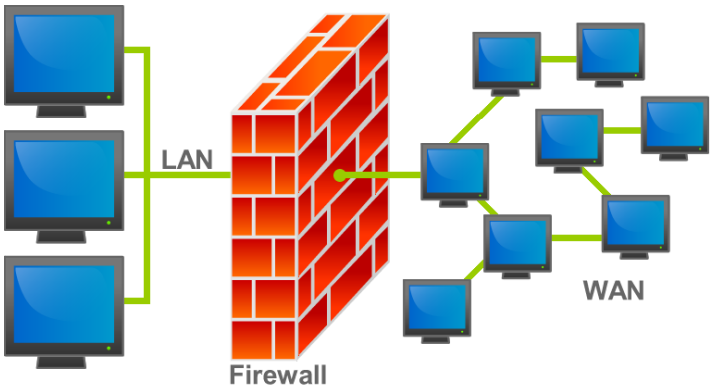

Figure 20: Common firewall setup

3.4.1.1 Open Source Firewalls

PfSense

PfSense (*https://www.pfsense.org/*)is an open source firewall/router computer software distribution based on FreeBSD[50]. It can be installed on both a physical computer or a virtual machine to make a dedicated firewall/router for a network and is noted for its reliability and for offering features often only found in expensive commercial firewalls. It can be configured and upgraded through a web-*based* interface, and requires no knowledge of the underlying FreeBSD system to manage. PfSense is commonly deployed as a perimeter firewall, router, wireless access point, DHCP server, DNS server, and as a VPN endpoint. It supports installation of many third-party packages like IDS and Content Caching applications.

The major features included in Pfsense are presented below:

- **Firewall**: Pfsense firewall offers a large selection of features. Filtering can be implemented by IP protocol, source and destination IP as well as source and destination port for TCP and UDP traffic. Number of simultaneous connections can be limited on a *per-rule* basis. P0f[51] utility is used, allowing passive OS/network fingerprint detection and filtering. Optional traffic logging matching each rule. Highly flexible policy routing possible by selecting gateway on a *per-rule* basis (for load balancing, failover, multiple WAN, etc.). Aliases allow grouping and naming of IPs, networks and ports facilitating management of complex network topologies. Moreover, PfSense performs packet normalization leaving no uncertainties for the packet's destination, dropping TCP packets that have invalid flag combinations and protecting *-at the same time-* the system against specific threats. Furthermore, it includes characteristics such as user

[50] FreeBSD is a free and open-source Unix-like operating system descended from Research Unix via the Berkeley Software Distribution (BSD). For more details also see, *among others*: *https://en.wikipedia.org/wiki/FreeBSD*

[51] See: *https://en.wikipedia.org/wiki/P0f*

defined limitation of simultaneous connections, flexible gateway policy routing and transparent level-2 firewall able to connect different interfaces and filter the traffic between them. Finally, the firewall can be at any time disabled leaving pfsense to operate as a router.

- **State Table**: Pfsense is a stateful firewall, meaning that it performs stateful packet inspection (SPI) keeping track of the state of network connections. This way it can identify the legitimate packets and reject the rest. The state table stores the necessary information regarding the various connections. Handling correctly a state table is not an easy task to accomplish, at least not for most firewalls, but pfsense can provide an easy management with various control features, due to the abilities of the Open BSD Packet Filter. These features include a parameterized table size that can be expanded to contain as many states as necessary, at the cost of memory usage. The user can also decide about the number of simultaneous client connections, states per host, connections per second, new connections per second etc. State handling also offers different options such as keep or modulate state, synproxy state or to not keep state entries. State table can finally be optimized with respect to latency, aggressiveness or to a moderate operation.
- **Network Address Translation (NAT)**: It supports port forwarding with different public IPs and 1-to-1 NAT for individual IPs. Also it offers improved outbound NAT setting, giving the choice to create specific NAT rules. Inbound and Outbound Load Balancing: The first one distributes the load to a number of servers and it is usually used for web servers or mail servers. The restriction with inbound load balancing is that it can only distribute the load equally and cannot verify the validity of the data. The second one is used for WAN connections.
- **VPN**: Pfsense supports the following types of VPN connectivity: IPsec[52], Open VPN[53] and PPTP[54]. IPsec provides connectivity with other devices that support IPsec and can be used for site-to-site connectivity without regardless the type firewall the other site uses (i.e. Pfsense, other open source firewalls or commercial products). Open VPN is a SSL VPN[55] that supports various client operating systems. PPTP is a well liked VPN option because it is supported by almost every operating system. The problem with PPTP server is that in case there is only one IP public available, clients inside the network cannot operate because PPTP are unable to use the same IP address for outbound connections.
- **Reporting and Monitoring RRD Graphs**: Pfsense uses rrd graphs to provide statistical data regarding CPU utilization, throughput, firewall states, throughput for every interface, etc. Additionally real-time data can be obtained using SVG graphs.
- **Dynamic DNS[56]**: It permits to register the public IP address to various dynamic DNS providers such as DynDNS, DHS[57], NO-IP[58] and others.

---

[52] For more details see, *for example*: https://en.wikipedia.org/wiki/IPsec

[53] For more details see, *for example*: https://openvpn.net/

[54] For more details see, *for example*: https://en.wikipedia.org/wiki/Point-to-Point_Tunneling_Protocol

[55] Also consider the context discussed in: http://searchsecurity.techtarget.com/definition/SSL-VPN

[56] Dynamic DNS (DDNS or DynDNS) is a method of automatically updating a name server in the Domain Name System (DNS), often in real time, with the active DDNS configuration of its configured hostnames, addresses or other information. More information can be found, *inter-alia*, at: https://en.wikipedia.org/wiki/Dynamic_DNS

[57] For more details also see: http://www.dhs.org/support/dynamic.shtml

- **Captive Portal**: It forces authentication for network access and it is usually used for hot spot networks or as an extra security measure for wireless networks. As part of pfsense it is responsible for a number of tasks. It controls the number of connections preventing denial of service. It disconnects clients after a specified time frame or if they have been idle for a long period. It offers different authentication options, URL redirection after authentication has been completed and MAC filtering.

iptables

Iptables[59] is a command line utility for configuring Linux kernel firewall implemented within the Netfilter project[60]. The term iptables is also commonly used to refer to this kernel-level firewall. It can be configured directly with iptables, or by using one of the many frontends and GUIs. Iptables is used for IPv4 and ip6tables is used for IPv6.

ebtables

The ebtables program[61] is a filtering tool for a Linux-*based* bridging firewall. It enables transparent filtering of network traffic passing through a Linux bridge. The filtering possibilities are limited to link layer filtering and some basic filtering on higher network layers. Advanced logging, MAC DNAT/SNAT[62] and brouter facilities are also included. The ebtables tool can be combined with the other Linux filtering tools (iptables, ip6tables[63] and arptables[64]) to make a bridging firewall that is also capable of filtering these higher network layers. This is enabled through the bridge-netfilter architecture which is a part of the standard Linux kernel.

IPFire

IPFire[65] is a linux distribution which acts as a router and firewall. It focuses on flexibility, and scales from small- to middle-sized business networks and home networks. Beginning with a small firewall system of a few megabytes, it is possible to run IPFire as a file server or VPN gateway (GW) for staff, branches or customers. Its design enables the user to create a tailor-made system fitting their needs. It ships with an extensive package management utility (Pakfire[66]) which allows the base system to be extended by various add-ons. The graphical web user interface has been designed for beginners and also it offers expert options so that powerful rules can be created.

IPFire employs a Stateful Packet Inspection (SPI) firewall. That means that the firewall internally stores information about every connection and is then able to "associate" every packet that transits the firewall to the connection it belongs to. The GUI can be used to create Network Address Translation rules like port-forwardings (DNAT) and source NAT rules. These two types of address translations, allow to hosting server farms behind the firewall and masquerading any

---

58 See, *for example*: http://www.noip.com/support/knowledgebase/section/dns/
59 For more details see: https://wiki.archlinux.org/index.php/iptables
60 For more details see: http://www.netfilter.org/
61 For more details see: http://ebtables.netfilter.org/
62 For more related information also see: http://www.ques10.com/p/11173/difference-between-snat-and-dnat-1/
63 http://linuxcommand.org/man_pages/ip6tables8.html
64 https://linux.die.net/man/8/arptables
65 More information can be found at: http://www.ipfire.org/
66 See: http://wiki.ipfire.org/en/configuration/ipfire/pakfire/start

private networks or private IP addresses. The firewall can be paired with an Intrusion Detection System (IDS), which will actively scan and block any threats. The IPFire firewall is based on the Linux netfilter Packet Filtering framework which is famous for its command line tool that is called iptables. It is paired with a P2P filter that enriches the feature set by allowing to filter certain P2P protocols.

Untangle

Untangle Unified threat management[67] (UTM) is an OSI layer 7 application layer firewall which filters traffic based on IP-address, port number, protocol, most significantly Active Directory users and groups. UTM is a popular commercial-grade open source complete Linux solution with built-in OpenVPN. It can be upgraded on hardware at a small cost. Provided with graphical interface for configuration and administration, UTM is an open source distribution with less downtime, live backup features along with zero stress of installation and configuration.

UTM supports a wide range of network applications with less resources and improved bandwidth. Apart from this, it provides modules to integrate the following network solutions:

- Firewall: UTM examines the flowing traffic at the transport layer of the OSI model and secure granting access token to the legitimate network only.
- Protocol Control: Untangle has an improved version of protocol control, which extends the feature to control protocol and shuts down the ports if violation occurs.
- Attack, phish, virus, spam and spyware, malware blocker: Based on open source CalmAV[68], Spam Assassinblocks phish, virus and spam, *respectively*. This prevents Distributed denial of service (DDoS) attacks and other attacks by blocking unspecified hosts, and also guards from various threats.
- OpenVPN: Users access the internet from a remote network securely with VPN provided with Untangle. It also administers protected distribution of software and encryption keys.

Open vSwitch as a Firewall

According to ovs.org, Open vSwitch[69] (OVS) is an open source software switch designed to be used as a "virtual switch" in virtualized server environments. The goal of OVS is to implement a switching platform that enables standard, vendor-independent management interfaces and opens the forwarding functions of switches to programmatic extension and control. It supports all versions of OpenFlow protocol.

Open vSwitch is a "software switch", which implements OpenFlow protocol. It manages the Flow Tables for the Datapaths which are used for forwarding the incoming traffic according to matched entries. The ability of either forwarding or blocking packets provides Open vSwitch firewall capabilities.

Below are presented fields of the flow table entries:

[67] For more details also see: *https://en.wikipedia.org/wiki/Unified_threat_management*
[68] For more details see: *https://www.clamav.net/*
[69] For more details see: *http://openvswitch.org/*

- Match Fields: Will be matched against incoming packets. It includes packet header and input port
- Priority: Matching priority for this Flow Table entry.
- Counters: Number of received packets matching this rule.
- Instructions: Used to modify the action to be applied on the packet.
- Timeout: The number of seconds this Flow Table entry lives in the Table.
- Cookie: Opaque value chosen by controller. Not used while processing a packet. It is used by the controller.

3.4.1.2 Firewall VSF Implementation

A virtual firewall is a firewall service running in a virtualized environment and providing the usual packet filtering and monitoring services that a physical firewall would provide. Virtual firewall in bridge-mode acts like its physical-world firewall "analog". Positioned in a strategic point of the virtual network infrastructure, it can intercept virtual traffic destined for other segments. A routing firewall participates in the IP process, whereas a bridging -or transparent- firewall does not. A transparent firewall acts more as a tap on a line, while a routing firewall has to forward traffic onto its next destination.

The advantages of a transparent firewall are that it can be installed in-line between two devices without having to reconfigure the IP subnet used, as the interfaces on the firewall are unnumbered. From a security perspective, a transparent firewall is quieter as it does not participate in IP connections an attacker may not even know it is there, unless something is blocked. An attacker will have difficulty determining the type of firewall being used.

The versatile nature of the transparent firewall greatly matches the VSF[70] needs for on-demand application of a firewall service in line of an existing link without the need IP subnet reconfiguration. The current Firewall VSF is implemented using Open vSwitch software on an Ubuntu 14.04 operating system[71]. The NVF consists of one virtual machine that requires at least two virtual network interfaces.

The Firewall VSF has the ability to be applied in-line on an existing network link deciding which packets will pass through its two network interfaces. It can be instantiated with specific rules that allow only the preferred traffic to be propagated. Additional rules can be applied to the VSF after instantiation via its RESTful web API, providing dynamic security policies to be enforced only by higher level modules of the SESAME project. As illustrated in Figure 21, the Firewall Rule Receiver Web API receives firewall rules in JSON format by an HTTP request and translates it to the appropriate Open vSwitch flow table entry.

[70] For more details about the Varnish Security Firewall (VSF) see: **https://github.com/comotion/VSF**
[71] For more details see: *https://www.ubuntu.com/desktop*

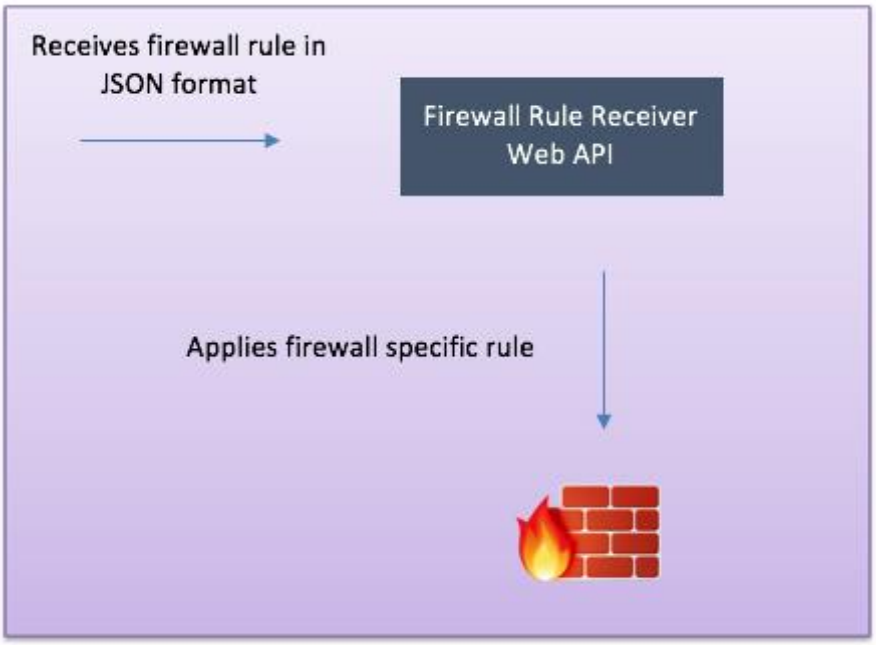

Figure 21: RESTful web API in Firewall VSF for policy enforcement

Furthermore, for purposes of monitoring and security validation, the Firewall VSF provides another service (Figure 22) whose role is to publish information about passing or blocking of packets to external interfaces for further analysis.

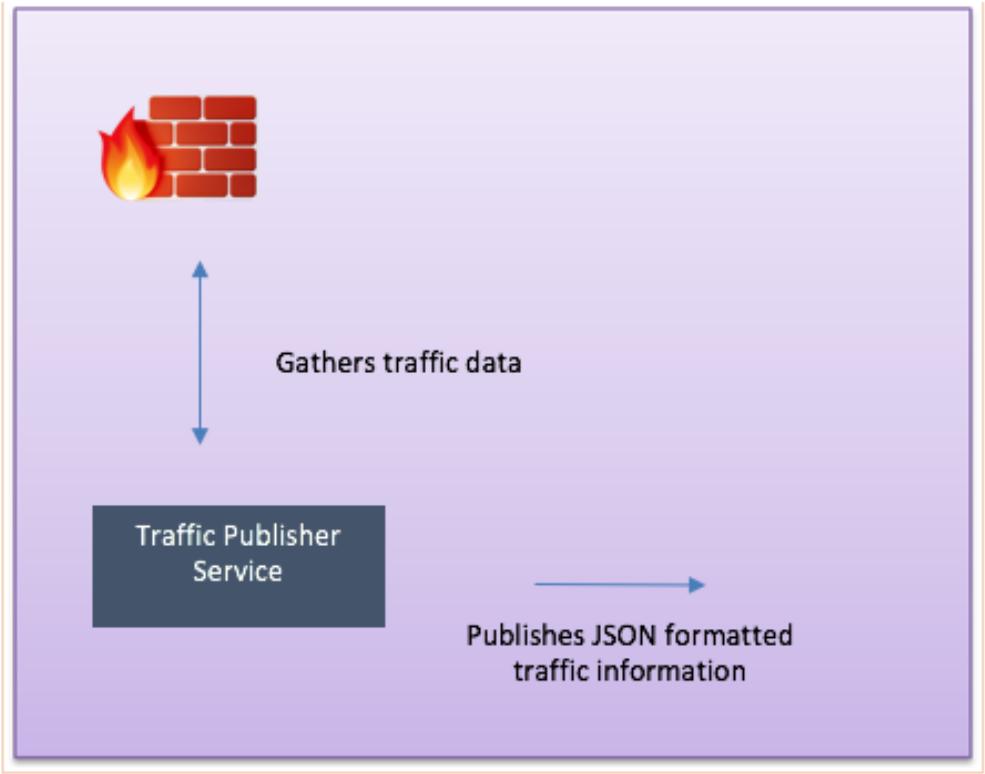

Figure 22: Firewall VSF service publishing traffic rules information

3.5 vIDS

3.5.1 vIDS

3.5.1.1 Intrusion Detection System Definition

Firewalls make filtering decisions only based on network packet header data, while packet content data is not inspected. Analyzing packet payload is often essential for detecting packets with malicious content. This is where intrusion detection systems can be helpful. An IDS monitors and logs the network traffic for signs of malicious activity and generates an alert upon discovery of a suspicious event.

IDS deployment typically consists of one or more sensors placed strategically on the network. Additionally, the solution may contain an optional central console for easier management of all sensor nodes. The sensor placement on the network can of course differ, but in a situation where the objective is to protect internal network from external threats, these would be the optimal choices for the IDS nodes.

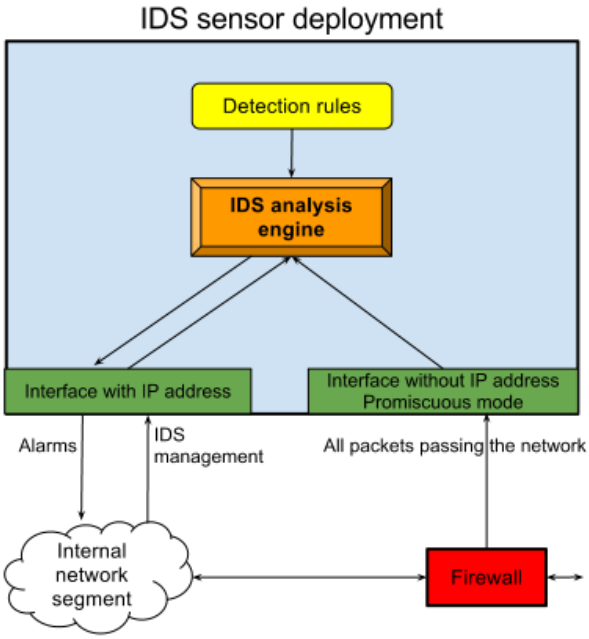

Figure 23: IDS sensor deployment

IDS deployments cannot protect networks on their own. They can only alert the security analyst that a malicious activity took place at a certain time. Therefore, IDS sensors are sometimes augmented with capabilities for firewall interaction (Figure 23). For example, block the source IP address of a DoS attack. However, this is a post-factum measure that cannot stop the malicious packets that triggered the creation of the firewall rule.

3.5.1.2 Overview of open source IDS

Snort

Snort[72] is an open-source intrusion detection system that is developed by Sourcefire. Snort was created in 1998 by Martin Roesch. It is capable of performing real-time traffic analysis and packet logging on IP networks. Snort is compatible with most operating systems (e.g. Linux, Mac OS X[73], FreeBSD, OpenBSD[74], UNIX and Windows).

The Snort detection engine and the Community Snort Rules are GNU[75] GPL v.2 licensed[76]. Sourcefire[77] also offers proprietary Snort Rules which are licensed by Non-Commercial Use License.

The two major components of Snort are the following:

1. Detection engine that utilizes modular plug-in architecture;

[72] For more details also see: *https://www.snort.org/*
[73] See: *https://en.wikipedia.org/wiki/MacOS*
[74] For more details also see: *https://www.openbsd.org/*
[75] For more relevant information see, *for example*: *https://en.wikipedia.org/wiki/GNU_Project*
[76] See: *https://www.gnu.org/licenses/old-licenses/gpl-2.0.html*
[77] For more details also see, *for example*: *https://en.wikipedia.org/wiki/Sourcefire*

2. Flexible rule language to describe traffic to be collected.

Snort structure is illustrated below. The preprocessor, the detection rules, and the alert output components of Snort are all plug-ins, which can be individually configured and turned on or off.

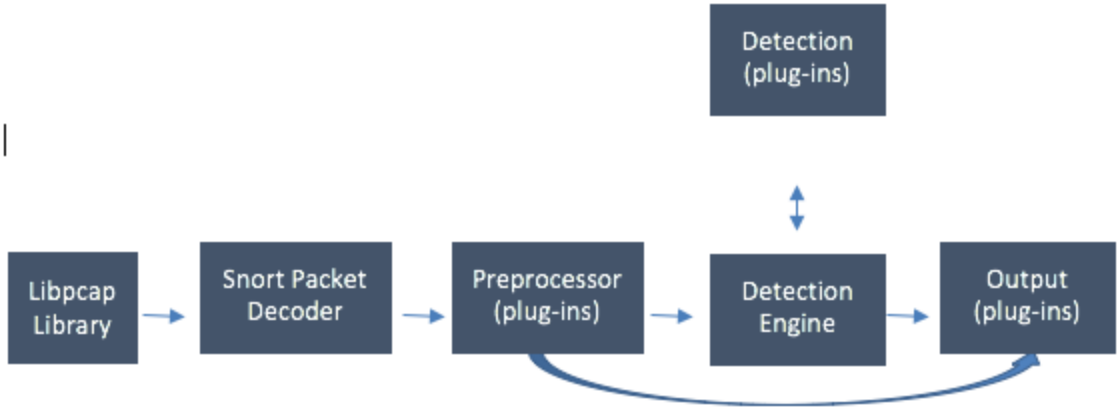

Figure 24: Snort structure and operation

The above figure (Figure 24) also shows how a network packet is handled if it is received by the network interface on which Snort is listening. The handling process is similar for all three assessed IDS solutions, but will be described here using Snort as an example.

1. Packet capture library is a software module that gathers packets from the network adapter. On UNIX and Linux systems, Snort uses libpcap library[78]. On Windows systems, WinPcap[79] is used.
2. Packet decoder receives the OSI layer 2 frame, analyzes packet headers and looks for any anomalies. Packet data is then decoded and prepared for further processing.
3. Preprocessors are plug-ins that operate on the decoded data. Preprocessors can alert on, classify, or drop a packet before sending it to the more CPU-intensive detection engine. By default, Snort comes with a variety of preprocessors, some of which are the following.
    a. Frag3 preprocessor[80] addresses problem of overlapping fragmented IP packets that could be used to avoid IDS/IPS detection.
    b. Stream5 preprocessor[81] makes Snort state and session aware. For instance, it can detect out-of-state packets created by Nmap tool[82].
    c. HttpInspect preprocessor[83] handles HTTP traffic. It extracts compressed data and decodes any hexadecimal or other expressions in the Universal Resource Identifier (URI).

[78] See: http://www.linuxfromscratch.org/blfs/view/svn/basicnet/libpcap.html
[79] More related information can be found at: https://www.winpcap.org/
[80] For more details also see: https://www.snort.org/faq/readme-frag3
[81] For more details also see: https://www.snort.org/faq/readme-stream5
[82] See: https://nmap.org/

4. Detection engine is the most important part of Snort. It operates on the OSI transport and application layers, analyzing packet contents based on the detection rules. The rules contain signatures for attacks.
5. Output plug-ins support a variety of alert and logging methods. When a preprocessor or rule is triggered, an alert is logged in Snort's own text or binary file logging formats, database or syslog.

Snort uses a single-threaded engine, which seems outdated, considering that nowadays multi-CPU and multi-core hardware is commonplace. As a result, by default Snort can only fully utilize one processor core. Snort developers are working on multi-threaded solution, however stable version has not yet been released. To alleviate this problem Snort can be run as multiple processes; each process utilizing a different processor core. This, however, increases the level of complexity, because the default network socket packet capture library needs to be replaced.

Suricata

The Suricata Engine[84] is a fairly new open-source intrusion detection and prevention engine. The initial beta release was made available for download on *January 01, 2010*. It is developed by Open Information Security Foundation (OISF), which is a non-profit foundation supported by the US Department of Homeland Security (DHS) and a number of private companies.

Suricata is compatible with most operating systems (e.g. Linux, Mac, FreeBSD, UNIX and Windows). The Suricata Engine is available to use under the GPL v.2 license. OISF claims that *The Suricata Engine is not intended to just replace or emulate the existing tools in the industry, but will bring new ideas and technologies to the field.* However, the industry considers Suricata a strong competitor to Snort and thus they are often compared with each other. Both systems seem to have their advantages and strong community support.

The operation modes of Suricata are the same as Snort's. It can be used either as an IDS or IPS system. There are no differences when connecting Suricata to the network. Suricata even has basically the same rule syntax as Snort (although not 100%), which means that both systems can use more or less the same rules. The general data flow through Suricata is similar to Snort. Packets are captured, decoded, processed and analyzed. However, when it comes to the internals of the Suricata Engine, differences become apparent. Suricata also features the HTP Library that is a HTTP normalizer and parser written by Ivan Ristic for the OISF. This integrates and provides advanced processing of HTTP streams for Suricata. The HTP library is required by the engine, but can also be used as an independent tool. Suricata uses a multi-threaded approach opposed to the Snort's single threaded engine. Threads use one or more Thread Modules for this. Threads have an input queue handler and an output queue handler. These are used to get packets from other threads, or from the global packet pool.

Bro

Bro intrusion detection system[85] is focusing on network security, but also provides a comprehensive platform for more general network traffic analysis. Bro was created by Vern

---

[83] *http://manual-snort-org.s3-website-us-east-1.amazonaws.com/node17.html#SECTION00327000000000000000*
[84] For more details also see: *https://suricata-ids.org/*
[85] For more details see: *https://www.bro.org/*

Paxson, who is still leading the project jointly with a team of researchers and developers at the International Computer Science Institute (ICSI) in Berkeley and the National Center for Supercomputing Applications in Urbana-Champaign.

Bro and its pre-written policy scripts (rules) come with a BSD license, allowing free use with even less restrictions than the GPL v.2 license of Snort and Suricata. Moreover, it is important to note that Bro policy scripts are written in its own Bro scripting language that does not rely on traditional signature detection. It analyzes network while trying to detect anomalies, e.g. attacker installing hacked SSH daemon. It is said that Bro language takes some time and effort to learn, but once mastered, the Bro user can write or modify Bro policies to detect and alert on virtually any type of network activity. Bro is not a full-blown IPS, but can function as an IDS with active response. Its policy scripts have the functionality to execute programs, which can, *in turn*, perform a variety of tasks (e.g., send e-mail or SMS, insert new rules to the firewall). Furthermore, Bro comes with a useful tool called BroControl[86] which enables the administrator to manage multiple Bro nodes at once. In addition to being able to control the Bro instances, it could even execute shell commands on all nodes. Similar to Snort, Bro is also single-threaded. Although, the developers of Bro have implemented a proof-of-concept (PoC) multi-threaded version of Bro, it is not yet ready for release. Therefore, once the limitations of a single processor core are reached, the only option is to spread the workload across many cores or even many physical nodes. The accompanying BroControl tool provides the means to easily manage many Bro processes. However, similar to Snort, this method significantly increases the level of system complexity.

3.5.1.3 IDS VSF Implementation

The current IDS VSF implementation utilises Snort IDS software on Ubuntu 14.04 operating system. Incoming traffic to the IDS VSF is being analysed in real time and analysis decisions are being communicated to external interfaces as HTTP requests. This VSF consists of one virtual machine which requires to have one virtual network interface where all traffic that need to be monitored must be routed (or mirrored).

In order to provide intrusion detection functionalities that cover external modules needs, this VSF implements a RESTful API (Figure 25) which accepts requests for creating, deleting and modifying rules that can be applied in Snort detection engine. This offers an easy way of external configuration of the VSF without requiring knowledge of its inner workings.

[86] https://www.bro.org/sphinx/components/broctl/README.html

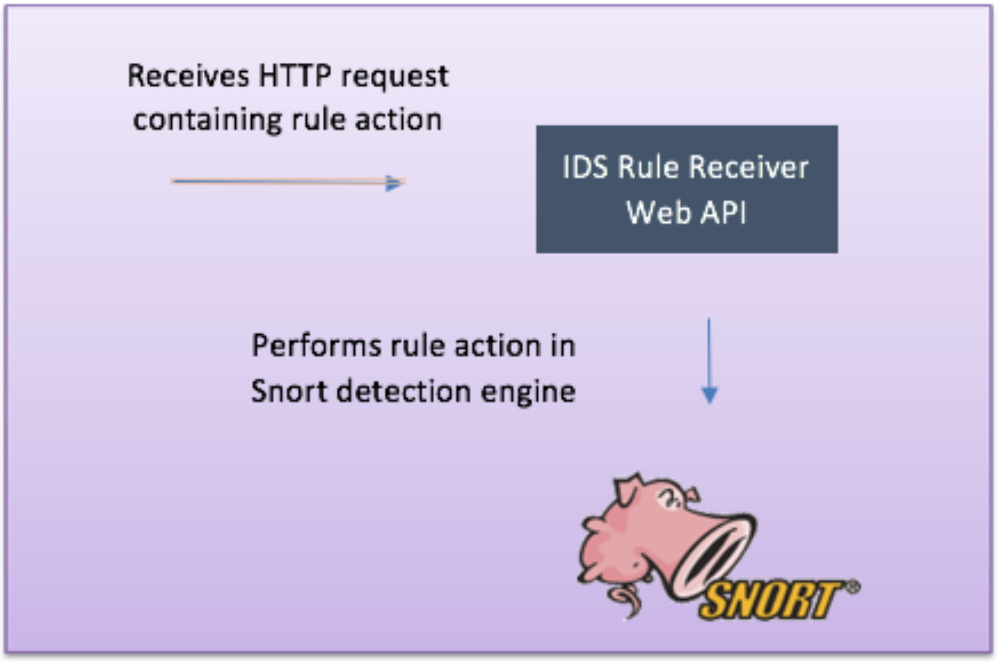

Figure 25: A RESTful API for creating, deleting and modifying rules in Snort IDS

Once traffic enters the IDS VSF, Snort software analyses all packets. Snort detection engine, described above, can contain rules which consist of conditions. When the conditions of a rule are met, the detection engine produces an event and saves it in a log file. Moreover, the IDS VSF provides another functionality, necessary for the utilization of the results produced by Snort packet analysis, the Event Publisher Service (Figure 26). It translates, curates, and publishes events in readable format to external interfaces for further analysis. Snort event logs are saved in Unified2 format[87] so the Event Publisher Service translates them to JSON format, assesses their timestamp to avoid publishing redundant information and publishes the events.

[87] See: *http://manual-snort-org.s3-website-us-east-1.amazonaws.com/node44.html*

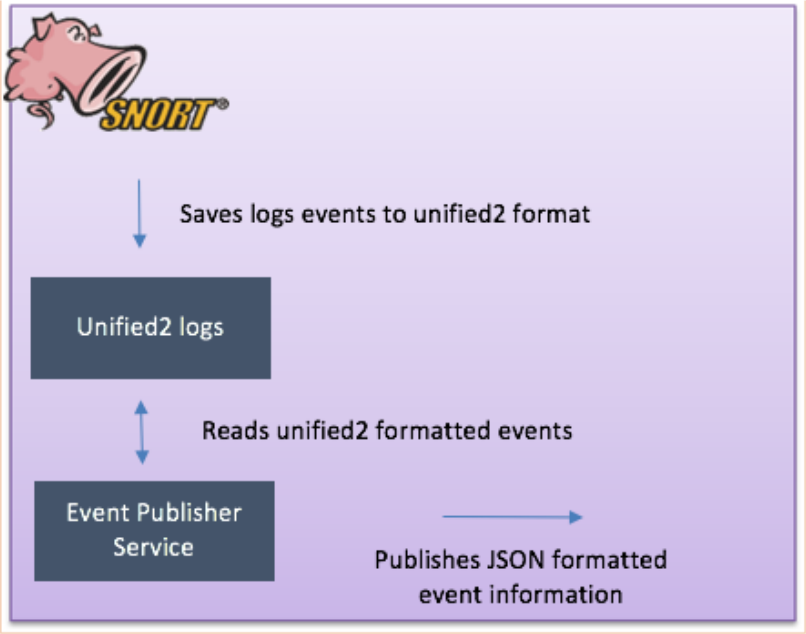

Figure 26: The Event Publisher Service within the IDS VSF

4 VNF performance evaluation and monitoring framework

As stated previously, performance evaluation and monitoring are two important issues on any cloud-*enabled* radio system. In particular, in cases like SESAME where multiple independent instances of mobile edge services operate over the shared radio-cloud environment (each belong to a tenant), performance evaluation becomes more complicated. It demands simultaneous consideration of both, the radio status and the cloud parameters for all the action related to the service lifecycle management. In the most general way, light DC and eNB hardware parameters (e.g. CPU and RAM usage), radio access network performance measurements (e.g. RRC connection related measurements), VNF and NS monitoring metrics (e.g. calls per second) should be extracted and processed. It implies the involvement of the SESAME solution at different levels, from NFVI to CESCM. In this document, we focus only on the service VNFs developed and delivered in the SESAME project. First, based on the ETSI framework we will highlight what parameters can be monitored on VNFs and consequently NSs (a set of connected VNFs together). Next, we will review the selected measurements per SESAME service VNFs.

Bear in mind that, information provided in this document complete the inputs of WP3 (the radio performance metrics), WP5 and WP6 (the management and orchestration solutions defining monitoring measurement extraction mechanism at VIM and CESCM levels as well as decision making processes to react upon failures) in a way that closes a quality of service (QoS) insurance feedback loop in the context of the SESAME project.

Cloud Systems Performance Monitoring

In the cloud environment, monitoring parameters can be divided into three main categories: (i) Network Functions Virtualization Infrastructure (NFVI), micro server's hardware; (ii) Network Functions Virtualisation (NFV) virtual machines preforming specific network functionalities, and; (iii) SDN-*based* monitoring for virtual links. Among these three, number two is the main focus of this document, but we will briefly take a look on the other parts too.

The SESAME NFVI (Light DC) is formed by clustering CESCs. Therefore, it is important to "keep track" of the individual performance per micro server. Key performance metrics can be used for this purpose, i.e.:

(i) *CPU utilization:* as the brain of the micro server, CPU carries out the instructions of VNF, performing the arithmetical, logical, and input/output operations. Thus, it is important to monitor its utilization and possibility keep it as low as possible;

(ii) *RAM utilization:* RAM is used to load information required by VNFs for faster access thereby improving the overall performance. If a micro server runs out of RAM, a portion of the hard drive can be dedicated as the virtual memory. This process is called swapping, which will cause performance degradation since the hard drive is much slower than RAM (e.g. 1000 times slower). It is important to monitor the RAM utilization and possibly add RAM to micro servers in case of need;

(iii) *Hard Disk Drive (HDD) utilization:* the operation system kernel, hypervisor and agents on the micro server needs space on the HDD for normal operating processes including paging files and certain caches. The application running on the server (VNFs) also needs space to write temporary data to cache for efficient operation as well as permanent data that will be accessed by the user. Thus, low free space on a HDD might cause micro severe performance issues, iv- System hardware: micro server might include other devices such as HWA (e.g.: GPU, FPGA, DSP, etc.), CPU fan, power supply, etc., that affect its overall performance. Health performance parameters of hardware as well as metrics such as temperature, air flow and humidity needs to be also monitored. Prosperity or open source solutions, e.g. OpenNMS [19] or ManageEngine [20] can be used for this purpose. The extracted information will be exposed to CESCM (the entity responsible of monitoring inputs process, decision making and failure reactions).

European Telecommunications Standards Institute (ETSI) in [21] divided the monitoring matrices of NFV into two main categories:

(i) VNF monitoring parameters,

(ii) Network Service (NS) – a chain of VNF to provide an added-value service – performance metrics. From a conceptual perspective, a VNF is a virtual machine (VM), a container or a unikernel that runs a specific application inside which permits preforming the same functionality as a network middlebox (e.g. virtual packet gateway (vPGW) in comparison to the actual PGW hardware). Each VNF comes with a descriptor file contains its deployment requirements details, performance monitoring metrics, etc. ETSI in [21] suggested a way to specify different deployment flavors for the VNF in VNF descriptor (VNFD). These parameters can be either VM related information, e.g. CPU utilization, bandwidth consumption, etc., or VNF specific such as, calls per second, number of subscribers, number of rule flows per second, VNF downtime, etc. One or more of these parameters could be influential in triggering a reaction on the QoS loop.

At NS level, monitoring parameters represent metrics that are tracked to check the level of NS compliance with the agreed SLAs (e.g. NS downtime). As a matter of fact, NS monitoring log is an aggregated log of individual VNFs that composed the NS. Selected parameters at NS descriptor (NSD), as suggested by ETSI [21], are used for specifying different deployment flavors of an NS and/or to indicate different levels of NS availability. Examples for these parameters are: Calls per second, number of subscribers, number of rules, flow per second, etc.

Open source solutions like Prometheus[88] (OpenStack plugin) can be used to extract VNF and NS monitoring metrics.

For the third monitoring category, i.e. the SDN-*based*, a deep-dive into the details of the SDN data plane is necessary. SDN by definition provides control of the data plane flows, whereas monitoring itself is specific to the management plane. This raises the question of whether the current SDN controllers should include a dedicated monitoring functionality, or this part has to

[88] For more details see: *https://prometheus.io/*

be outsourced to the management plane components. Anyhow, for example by using the flow counters on the flow tables of open flow protocol [22], it is possible to support a certain level of performance monitoring.

This will be discussed in more details in the future forthcoming WP6 SESAME deliverables.

References

[1] Designing and managing VNFs the right way for network functions virtualization, Nokia Technology White Paper, 2016.

[2] Wind River, "Wind River Content Inspection Engine" on-line: *http://www.windriver.com/products/product-overviews/PO_Wind-River-Content-Inspection-Engine.pdf*

[3] IPOQUE, "Protocol and Application Classification with Metadata Extraction", on-line: *http://www.ipoque.com/en/products/pace*

[4] *http://www.qosmos.com*

[5] L. Deri, M. Martinelli, T. Bujlow, A. Cardigliano (2014): "nDPI: Open-Source High-Speed Deep Packet Inspection". In Proccedings of the International Wireless Communications and Mobile Computing Conference (IWCMC),pp.617-622, Nicosia, Cyprus, August 04-08, 2014.

[6] T. Bujlow, V. Carela-Español, P. Barlet-Ros (2014). *Extended Independent Comparison of Popular Deep Packet Inspection (DPI) Tools for Traffic Classification*. Universitat Politècnica de Catalunya.

[7] 5G-PPP use cases and performance evaluation models, available online at: *https://5g-ppp.eu/white-papers/*

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

[9] 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.

[10] InfluxData InfluxDB project, available online at: *http://docs.influxdata.com/influxdb/v0.9/*

[11] Grafana project, available online at: *http://grafana.org*

[12] Network Functions Implementation and Testing, T-Nova Deliverable 5.31, available online at: *http://www.t-nova.eu/results/*

[13] *https://www.itu.int/dms_pub/itu-t/oth/23/01/T23010000230001PDFE.pdf*

[14] *https://resources.ext.nokia.com/asset/200010*

[15] *http://docs.opencv.org/3.0-beta/modules/refman.html*

[16] *http://docs.opencv.org/3.0-beta/doc/tutorials/tutorials.html*

[17] *http://caffe.berkeleyvision.org/*

[18] *https://www.tensorflow.org/*

[19] *https://www.opennms.org/*

[20] *https://www.manageengine.com/*

[21] 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.

[22] *http://archive.openflow.org/*