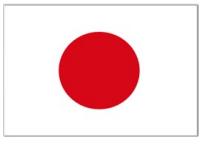


D4.2 – Network Slice orchestration– ver. 1.0.0

Aalto University, Ericsson, Eurecom, Orange, Hitachi,
University of Tokyo, Waseda University

Document number	D4.2
Status	Final
Work Package	WP 4
Deliverable type	Report
Date of delivery	29/June/2018 (M24)
Responsible	ORANGE
Contributors	Aalto University, Ericsson, Eurecom, Orange, Hitachi, University of Tokyo, Waseda University
Dissemination level	PU

This document has been produced by the 5GPagoda project, funded by the Horizon 2020 Programme of the European Community. The content presented in this document represents the views of the authors, and the European Commission has no liability in respect of the content.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the grant agreement No 723172, and from the Swiss State Secretariat for Education, Research and Innovation.

Change history

Version	Date	Status	Author (Company)	Description
0.0.1	24/01/2018	Working	ORANGE	Very first draft with ToC
0.2.0	18/05/2018	Working	AALTO, ERICSSON, EURECOM	Added contributions of three partners
0.3.0	29/05/2018	Working	AALTO, WASEDA	Added contributions of two partners
0.4.0	09/06/2018	Working	ORANGE, ERICSSON, HITACHI, UT, WASEDA	Added contributions, updated contribution of ERICSSON, extensive editing
0.6.5	11/06/2018	Working	ORANGE	Extensive editing, new figures
0.6.7	12/06/2018	Working	ORANGE	Ready for the first review
0.9.5	20/06/2018	Working	FRAUNHOFER, ORANGE, ERICSSON, EURECOM	After the review and validation of comments of the reviewer
0.9.6	26/06/2018	Working	Tarik Taleb (AALTO)	Overall revision of the document
0.9.7	27/06/2018	Working	HITACHI, ORANGE	Modification of Section 4.1.5: figures replaced, changes in text; implementation of changes suggested in the overall revision, renumbering and reformatting of equations
1.0.0	28/06/2018	Final	ORANGE	Version for publication

Authors

Full name	Affiliation
Sławomir Kukliński	Orange
Lechosław Tomaszewski	Orange
Taleb Tarik	Aalto University
Miloud Bagaa	Aalto University
Ibrahim Afolabi	Aalto University
Ilias Benkacem	Aalto University
Nicklas Beijar	Ericsson
Kashif Nizam Khan	Ericsson
Adlen Ksentini	Eurecom Institute
Pantelis Frangoudis	Eurecom Institute
Daisuke Okabe	Hitachi
Hidenori Inouchi	Hitachi
Akihiro Nakao	The University of Tokyo
Shu Yamamoto	The University of Tokyo
Du Ping	The University of Tokyo
Yoshiaki Kiriha	The University of Tokyo
Toshitaka Tsuda	Waseda University
Takuro Sato	Waseda University

Executive summary

The slice orchestration is a complex issue that can be decomposed into intra-slice and inter-slice issues. One of the key aspects of the intra-slice orchestration is the efficient initial placement of virtual network functions and the optimization of the placement during slice runtime. As the VNF placement problem is known to be NP-hard, many heuristic solutions to the problem exist. They may differ in computational complexity, the quality of the decisions and in the processing time. As the problem is an algorithmic one, there is no single best solution. Therefore, in this deliverable, several different approaches to virtual network function placement have been presented. Some of them are based on sophisticated algorithms whereas others look into more pragmatic, simpler but less efficient solutions. In this document, different algorithms related to Network Service or slice orchestration have been described that concern: dynamic service chain management, joint optimization of a number of VNFs and their placement in case of 4G vEPC, an algorithm based on Control Theory allowing to equilibrate the load on the 5G AMF instances in order to maintain an optimal response time within limited computing latency, a pragmatic approach to online optimization of vEPC nodes placement, and a slice-based CDN architecture with real-time resource management based on multi-level service monitoring. The inter-slice orchestration is shown on ICN/CDN example, which concerns dynamic resource assignment for slice stitching and distribution of gateways for such operations, as well as issues related to fast slice provisioning based on the resource pool concept.

Table of contents

List of tables

List of figures

1. Introduction

1.1. Motivation and scope

The slice orchestration issues can be decomposed into intra-slice and inter-slice issues. One of the key aspects of the intra-slice orchestration is the efficient placement of virtual network functions. This topic is the main issue that is addressed in this document. The problem of virtual functions placement can be split into an initial placement that is based on some initial assumptions and adjustment of virtual network functions based on monitoring of the deployed network, i.e. on-line placement of virtual nodes.

The question of placement efficiency has multiple dimensions. Within the scope of this document, the focus will however be on both of them, i.e. optimization of traffic and optimization of resources utilization. The first issue can be measured in terms of delay minimization, shortening of traffic paths, balancing the load of network functions, moving of aggregation points in order to avoid unnecessary traffic overheads, lowering the costs related to traffic routing and processing, etc. In the case of resources utilization, the main criteria should refer to both underlying DC resources (computation, storage) and inter-DC connectivity, which in real life can be additionally complicated by e.g. specific engineering rules established by the operator or redundancy requirements (physical separation of traffic paths or processing nodes).

As the problem is algorithmic one, there is no single best solution. Therefore, in this deliverable, several different approaches to virtual network functions placement have been presented. Some of them are based on sophisticated algorithms whereas others look into more pragmatic, simpler but less efficient solutions. The algorithms deal with the efficient placement of mobile network functions but also with the optimization of some MEC-oriented applications like video streaming. This document also deals with the inter-slice issue, and this topic is addressed in the example of the ICN network.

1.2. Structure of the document

Following this introductory section, the remaining part of the document is structured as follows:

- Chapter 2 provides key terminologies to be used in the document. It includes the descriptions of abbreviations and technical terms.
- Chapter 3 describes different techniques used for intra-slice orchestration. It describes algorithms concerning dynamic service chain management, an algorithm for joint optimization of the number of VNFs and their placement in case of 4G vEPC, an algorithm based on Control Theory allowing to equilibrate the load on the 5G AMF instances in order to maintain an optimal response time with limited computing latency, a pragmatic approach to online optimization of vEPC nodes placement, and a slice-based CDN architecture with real-time resource management based on multi-level service monitoring.

- Chapter 4 includes selected issues in inter-slice orchestration based on the ICN/CDN example. It concerns such aspects as dynamic resource assignment for slice stitching and distribution of getaways for such operations as well as issues related to fast slice provisioning based on the resource pool concept.
- Chapter 5 consists of final remarks.

2. Terminology

2.1. Abbreviations

Throughout this document, the following acronyms, listed in Table 1, are used.

Table 1 – List of acronyms

Abbreviation	Original term
5G PPP	5G Infrastructure Public Private Partnership
AMF	Access and Mobility management Function
API	Application Programming Interface
APPA	Advanced Predictive Placement Algorithm
AUSF	AUthentication Server Function
BBU	BroadBand Unit
BH	Busy Hour
BIP	Binary Integer Programming
BSSO	Business Service Slice Orchestrator
CDC	Central (national) Data Center
CDN	Content Delivery Network
CDNaaS	CDN as a Service
ClN	Cloud Network
CN	Core Network
CP	Control Plane
CPRI	Common Public Radio Interface
CSAR	Cloud Service ARchive
CSMF	Communication Service Management Function
DC	Data Center
DCN	Data Center Network
DP	Data Plane
DSSO	Domain Specific Slice Orchestrator
E2E	End to End
EM	Element Manager
eMBB	enhanced Mobile Broadband
EMS	Element Management System
EPC	Evolved Packet Core
EPCaaS	EPC as a Service
ETL	Extraction Transformation Loading
ETSI	European Telecommunications Standards Institute
EWMA	Exponential Weighting Moving Average
FCAPS	Fault, Configuration, Accounting, Performance and Security
HA	High Availability

Abbreviation	Original term
HO	HandOver
HSS	Home Subscriber Server
ICN	Information Centric Networking
IaaS	Infrastructure as a Service
IMSI	International Mobile Subscriber Identity
IoT	Internet of Things
JSON	JavaScript Object Notation
LDC	Local Data Center
LP	Linear Programming
LQR	Linear Quadratic Regulator
MA	Markov Approximation
MANO	Management And Network Orchestration (ETSI NFV)
MDSO	Multi-Domain Slice Orchestrator
MEC	Mobile Edge Computing
MILP	Mixed-Integer Linear Programming
ML	Middle Layer
MME	Mobility Management Entity
mMTC	massive Machine-Type Communications
MNO	Mobile Network Operator
MOS	Mean Opinion Score
MVNO	Mobile Virtual Network Operator
NF	Network Function
NFV	Network Function Virtualization
NFVI	NFV Infrastructure (ETSI NFV)
NFVO	NFV Orchestrator (ETSI NFV)
NSD	Network Service Descriptor (ETSI NFV)
NSO	Network Service Orchestrator
OAI	Open Air Interface
PNF	Physical Network Function
PoP	Point of Presence
QoE	Quality of Experience
QoS	Quality of Service
PDN	Packet Data Network
PGW	PDN GateWay
POD	Performance Optimized Data-center
PNF	Physical Network Function
PSQA	Pseudo-Subjective Quality Assessment
RAN	Radio Access Network
RDC	Regional Data Center
REST	REpresentational State Transfer
RO	Resource Orchestrator

Abbreviation	Original term
RRH	Remote Radio Head
RU	Radio Unit
SA	Service Area
SDN	Software Defined Network
SGE	Service Graph Embedding
SGW	Serving GateWay
SLA	Service Level Agreement
SMF	Session Management Functions
TA	Tracking Area
TAU	Tracking Area Update
TOSCA	Topology and Orchestration Specification for Cloud Applications
UE	User Equipment
UP	User Plane
uRLLC	Ultra-Reliable and Low Latency Communications
VIM	Virtual Infrastructure Manager (ETSI NFV)
VNF	Virtual Network Function (ETSI NFV)
VNFM	VNF Manager
vXYZ	virtualized XYZ (function)
WAN	Wide Area Network
WIM	WAN Infrastructure Manager

3. Intra-slice orchestration

In general, a network slice is composed of a combination of interconnected VNFs/PNFs, which are optimally positioned on the underpinning network infrastructure in order to satisfy a set of requirements while running under certain network constraints to deliver a particular use case scenario. Every network slice is orchestrated from VNF/PNF ingredients or a combination of both belonging to different segments of the network. From an E2E point of view, a network slice is expected to be orchestrated from network substrates belonging to the Access, Core and Transport part of the network. Running instances of different network slice components are otherwise referred to as sub-slices of that particular network slice. In other words, a full fledged mobile network slice is composed of sub-slices of the access, transport and core network.

Moreover, in the context of this section, the focus would be on the dynamic and optimal replacement of the running VNFs, e.g. a vEPC/5G core that makes up the core network sub-slice of a mobile network slice, after it has already been deployed in order to serve the application users better due to changes in the network dynamics.

3.1. Existing approaches

One of the main goals of the orchestration is to host VNFs in suitable locations of the network to minimize the latency or response time and to maximize the overall resource utilization. The initial VNF placement in network slicing takes care of placing network functions before the deployment of the slices whereas in this section our specific focus is on dynamic management of VNF placement in runtime based on resource utilization and user requirements.

A good number of approaches have been proposed in the literature for optimal and dynamic provisioning/placement of VNFs in cloud environment. We will survey and assess a few interesting observations on the existing approaches in this section.

Most of the existing approaches solve the VNF placement problem with heuristic algorithms which apply the optimal VNF placement in NFV chaining. For example, Xia et al. proposed a near optimal VNF placement using BIP and heuristic algorithms in [1]. This approach particularly focuses on optimal VNF placements which minimize the optical-electrical-optical conversion for NFV chaining in DCN. It places VNFs on pods which are self-contained containers. In this regard, the authors first formulate the VNF placement problem as BIP and then propose a heuristic based approach to solve it. The proposed heuristic algorithm maintains a sorted list of available pods based on their resource availability and another sorted list of VNFs based on their resource demands and then tries to find optimal placements for each VNF in the NFV service chain based on “best-fit” strategy. The algorithm also consolidates pods to reduce the total number of used pods. The results obtained with such placement in comparison to “first fit” (placing VNFs in the first pod encountered with enough resources) are quite promising. This algorithm can be extended to an edge dominated network infrastructure which may include the user demand and edge dominated VNF consolidation by adding these criteria while maintaining the pod lists.

Another interesting way of solving the VNF placement problem is to consider it as SGE problem and solve it using greedy heuristics as proposed in [2]. In this approach, the services are modeled as service graphs, and the NFs are modeled as nodes. This approach also maintains a resource graph of the physical substrate network and the goal of the greedy heuristic approach is to find an overall mapping of the service graph onto the physical resources. The proposed greedy backtracking algorithms map nodes and links of the service graphs based on different orchestration parameters and metrics. The preference parameters are customizable, and hence this approach is suitable and interesting for run-time VNF placement management. The mapping process iterates over the graphs and finds a suitable mapping based on customized criterions, and if the embedding fails it backtracks to the previous step and continues the embedding until it finds a suitable mapping. The proposed algorithm is implemented in an orchestration framework named ESCAPE [2]. The authors compared the performance of the algorithm with a MILP-based solution with promising results.

Some other approaches also tackle the placement problem with different objectives and different strategies. For example, the authors in [3] proposed an MA approach for optimizing the VNF placement with a joint objective of minimizing the operation cost as well as network traffic cost. Whereas, the authors in [4] propose APPA, which places VNFs based on mobile service usage and users' mobility behavior patterns. It places the VNFs to the best predicted location that is less utilized regarding resources and closeness to the users. This approach migrates the VNFs if the predicted location is different from the last allocation over an observation period. The results obtained with APPA compared to other VNF placement strategies are quite promising.

The virtual deployment and placement of core network elements in the form of vEPC/5G core in *ClNs* allows operators as well as network providers to achieve elasticity, flexibility, and a significant reduction in the operational cost of the overall system. Indeed, using NFV and general-purpose hardware in *ClNs* to run network functions helps in dynamically scaling up/down the network according to the demands of users for resources and can largely reduce the cost of both deploying and managing the network. NFV aims at offering diverse network services using network functions implemented in the format of software, deployable in an on-demand and elastic manner on the cloud. In return for its numerous advantages, virtualizing EPC and 5G Core network functions introduce some important challenges, which are mainly related to the placement of the telco-specific VNFs (i.e. MME, PGW, SGW, SMF, and AUSF) over a federated cloud to ensure optimal connectivity for users and simultaneously reduce the deployment cost.

The problem of VNF placement is somewhat similar to the VM placement in the cloud. About the latter, a large library of research work has been conducted for the placement of VMs (not necessarily hosting a VNF) in the same *ClN* or across multiple *ClNs* (i.e. federated cloud). These research works tackle the problem from different angles, considering specific criteria and constraints related to cost, availability, and performance, which pertain to both the network and the cloud infrastructure. For instance, the research work in [5] proposes an approach for VM placement and migration to minimize the time consumed in data transfers. The authors in [6] focus on mapping VMs to physical servers with the aim of improving server resource (e.g. CPU or memory) utilization, overcoming the lack of space in *ClNs* and maximizing the number of mapped VMs in one physical host. In [7], performance isolation (e.g. CPU, memory, storage, and network

bandwidth), resource contention properties (among VMs on the same physical host), and VMs behavioral usage patterns are taken into account in decisions on VM placement, VM migration, and cloud resource allocations. Usually, a *ClN* may start with an initial configuration and then apply adequate solutions to make a series of live migrations to transit the *ClN* from a sub-optimal state to an optimal one, similar in fashion to solving an iterative rearrangement problem. For this purpose, different algorithms can be used, such as N-dimensional set or bin packing [8], the simulated annealing algorithms [9], and ant colony optimization [10]. In other research work, optimal placement of VMs, running specific services, on physical machines, consider electricity-related cost as well as transient cooling effects [11]. Other research work do autonomic placement of VMs as per policies specific by the *ClN* providers and/or users [12]. Other VM placement strategies consider maximizing the profit under a particular SLA and a predetermined power budget [13]. Authors in [14] proposed a solution for solving the problem of cloud federation formation. This solution aims to increase the benefit for different clouds by sharing the resources among them. Similar to [14], authors in [15] proposed a solution for cloud federation formation that uses coalitional game. This solution explores the strength of cloud federation for ensuring the services availability, responding to different users' requests. A solution that creates federations among a set of cloud providers is proposed in [16]. This solution aims to reduce the energy cost when forming a federated cloud. A cooperative game approach is used in that solution allowing different cloud providers to negotiate the resources in a distributed fashion.

More recently, new research work has emerged, proposing algorithms for the placement of VNFs of vEPC/5G Core. In [17], the authors proposed a VNF placement method, particularly for placing mobility-anchor gateways (i.e. SGW) over a federated cloud so that the frequency of SGW relocation occurrences is minimized. This work aimed to conduct efficient planning of SAs retrieving a trade-off between minimizing the UE handoff between SAs, and minimizing the number of created instances of the virtual SGWs. In [18], the focus was on VNF placement and instantiation of another mobile network functionality, namely data anchoring gateway or PGW. The work argued the need for adopting application type and service requirements as metrics for *(i)* creating VNF instances of PGWs and *(ii)* selecting adequate virtual PGWs for UEs receiving specific application types. The placement of PGW VNFs was modelled through a nonlinear optimization problem whose solution is NP-hard. Three heuristics were then proposed to deal with this limitation. In [19], the authors proposed a framework, dubbed softEPC, for flexible and dynamic instantiation of VNFs where most appropriate as per the actual traffic demand. The proposed scheme addresses load-aware dynamic placement of SGW/PGW over the underlying transport network topology according to the traffic demands. The results show that up to 25% of network resources can be saved with same network topology and service points.

As aforementioned, different research work [14], [15] and [16] aimed to share the resources among different cloud networks by forming a federated cloud. These solutions aim at ensuring the stability of the grand coalition, formed by different clouds. The aim of the grand coalition is: *(i)* to share the virtual resources (e.g. RAM, virtual CPU, storage), and *(ii)* to ensure the service availability for responding to different users' requests. In contrast to these research works, the proposed solution, herein, aims to create an EPCaaS slice on top of different *ClNs*. The proposed solution specifies the number and locations of different instances of VNFs of virtual EPC, in each

ClN. Moreover, in contrast to the literature, the proposed solution also specifies the flavor, the number of virtual resources (i.e. the number of Virtual cores – CPU, memory, storage) that should be dedicated for each instance of each VNF *V*, in a fashion that reduces the cost while ensuring the QoS. It is still essential to design algorithms for dynamic placement of VNFs to meet the dynamic demand of different use cases. None of the abovementioned algorithms considered management of the system as well as possible failures, and sub-optimal processing due to fast condition changes.

3.2. Algorithms for dynamic service chain management

As mentioned at the beginning of this chapter, in this deliverable our aim is not only to orchestrate the initial VNF placement, but also to dynamically manage the VNFs and VNF replacement on different edge clouds/cloud sites based on resource utilization, performance metrics (i.e. delay), QoS and user mobility patterns. Flexible allocation of resources is crucial to ensure the QoS as well as to avoid resource contention. User mobility pattern and user concentration points (shopping mall, railway stations) also dictate the placement of VNFs and movement of VNFs to different cloud locations to maintain service quality. Resource demand fluctuates, and so does the user access patterns, which essentially requires flexible management and placement of VNFs and optimization of overall resource consumption while maintaining a constant QoS. To achieve these, we propose the following algorithms which aid the initial placement of service chains/VNFs over physical resources and also optimize the placement whenever the scenario of the network usage/demand changes.

In the following subsections three different algorithms are presented: *Add_chain*, *Delete_chain* and *Optimize_allocation*. As their name indicates, *Add_chain* algorithm adds a service chain as well as places the VNFs in the chain in nodes in the cloud, *Delete_chain* concerns deleting a service chain and deallocation of the VNFs from physical nodes and *Optimize_Allocation* algorithm optimizes the allocation for resource usage efficiency. We first explain the algorithms with an example and then present the algorithms in the following discussion. Fig. 1 depicts an example of cloud hierarchy: for brevity, we do not include the regional data center in this picture. As an example, we consider that one of the local data centers (edge cloud sites) has five nodes: Local Node 1, Local Node 2 … Local Node 5. We denote the local nodes as LNs. The service chains to be placed on these nodes are as follows (for brevity we represent the individual components or VNF as numeric i.e. 1.1, 2.1 etc.):

- Service chain 1: 1.1 → 1.2 (requires 30 units and 35 units of resources respectively);
- Service chain 2: 2.1 → 2.2 (requires 30 units and 50 units of resources respectively);
- Service chain 3: 3.1 → 3.2 → 3.3 → 3.4 (requires 40, 20, 55 and 10 units of resources, respectively);
- Service chain 4: 4.1 → 4.2 → 4.3 (requires 20 units of resources each);
- Service chain 5: 5.1 → 5.2 (requires 30 units and 40 units of resources respectively.

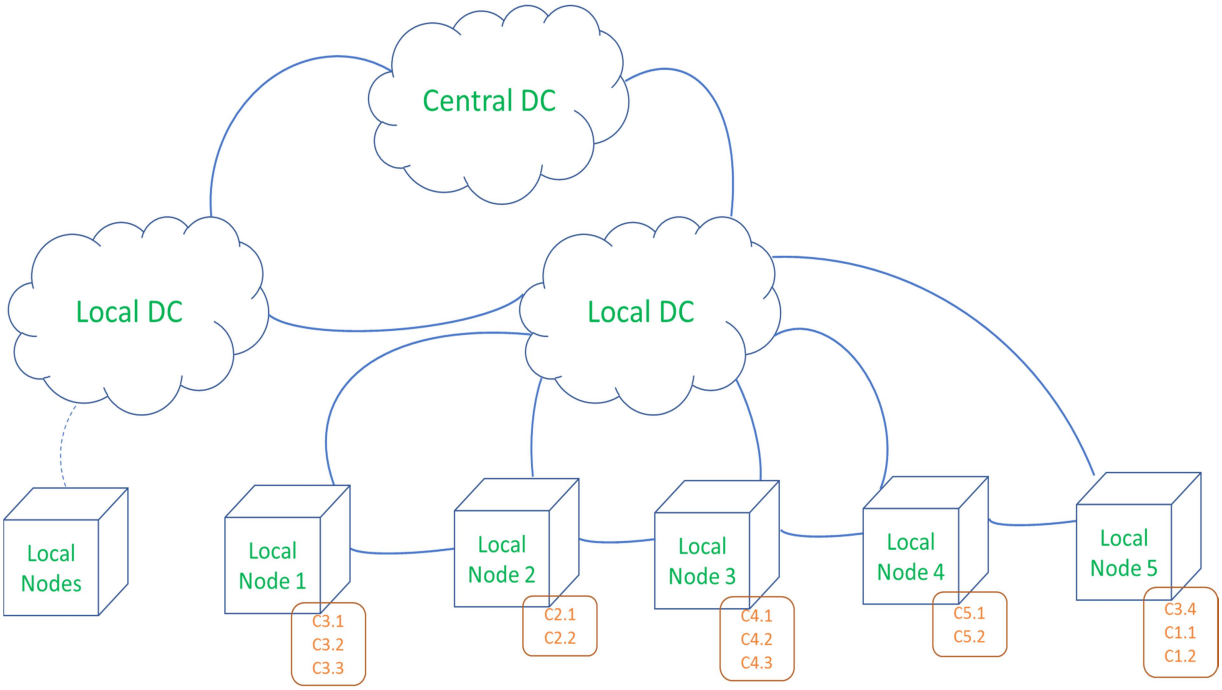


Fig. 1 – Service chain placement on Edge sites using algorithm *Add_chain*

The nodes have the capacity as follows: LN1 has free capacity of 120 units, LN2, LN3 and LN4 all have free capacity of 100 units, LN5 has free capacity of 80 units. The required resources by the VNFs and the capacity of the nodes are presented as numerics for the sake of simplicity. These can be expressed as a list of required/available resources in the actual algorithm. Fig. 1 depicts how the service chains are placed using the *Add_chain* algorithm initially. Note that in cases when the whole chain cannot be placed on an individual node, the chain is split and placed on several nodes. As an example of this case, please note that the placement of service chain 3 in Fig. 1, where the chain is split and placed on LN1 and LN5. After each service chain placement, the algorithm *Optimize_allocation* is called to check if an optimization of the placement is possible based on the criteria of optimization. For example, if we set the criteria of optimization as minimizing the total resource usage and apply the algorithm *Optimize_allocation* after the placement of service chain 5 we get the following scenario as depicted in Fig. 2.

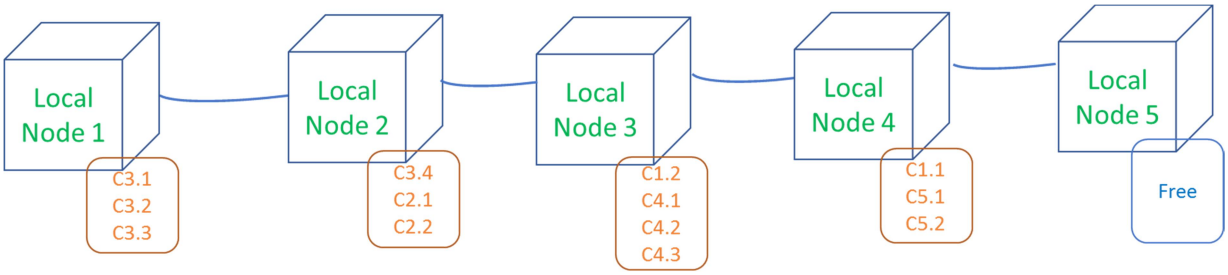


Fig. 2 – Service chain placement optimization using algorithm *Optimize_allocation*

As we see from Fig 2: according to the criteria of optimization, the algorithm *Optimize_allocation* finds new allocation for components 3.3, 1,1 and 1.2 and makes the LN5 free so that it can go to some deep sleep state and the other LNs have increased usage efficiency. Lastly, we show how the algorithm *Delete_chain* deletes the service chain 3 in Fig. 3. This figure shows how the algorithm performs a deletion of service chain for example in scenarios when a service chain is no

longer needed. Also in this case, algorithm *Optimize_allocation* is called after each deletion process to identify potential opportunities of VNF placement optimization.

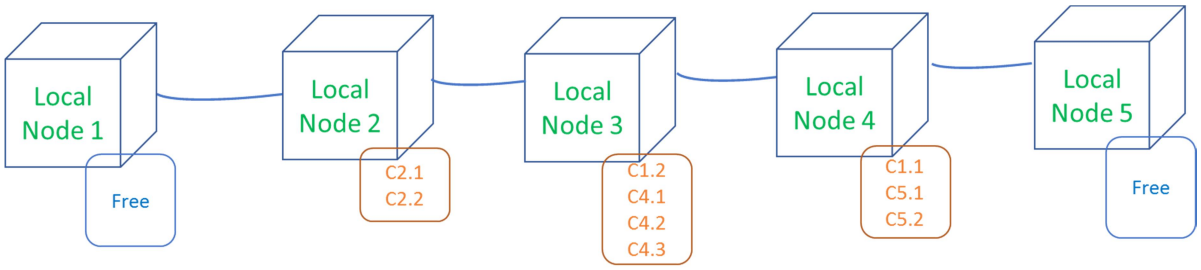


Fig. 3 – Service chain deletion using algorithm *Delete_chain*

In the following discussion, we present the algorithms. The explanation of variables used in all algorithms is provided below.

Table 2 – Explanation of variables used in algorithms

Variable	Explanation
V	List of VNFs
$C = \{c_1, c_2, \ldots, c_n\}$	Set of chains
$c = \{v_1, v_2, \ldots, v_n\}, v \in V$	Chain as ordered list of VNFs $v_1,v_2....v_n$
$N_s = \{n_1, n_2, \ldots, n_n\}$	Set of nodes in site s
$S = \{s_1, s_2, \ldots, s_n\}$	Set of sites where the nodes are situated (edge site, core site)
$R(v)$	Requirements of node v
$F_{site}(s)$	Free capacity of site s
$F_{node}(n)$	Free capacity of node s
$A_{site}(v)$	Site allocation of v
$A_{node}(v)$	Node allocation of v
$R_{capacity}(v)$	Required capacity requirement of v
$R_{site}(v)$	Required site of v
$s_{nearest}$	Nearest site from where the user request is received

3.2.1. Description of the algorithm *Add_chain*

The algorithm *Add_chain* requires a chain c (ordered list of VNFs $v_1>v_2>v_3 \ldots v_N$) to be added and the nearest site information from where the request is coming. This algorithm tries to accommodate the whole chain and the VNFs in one site and more specifically on one physical node. If it cannot place the whole service chain, then it splits the chain and accommodates the VNFs on different nodes. The algorithm first takes a copy of the current allocation and finds a cloud site where it can place the VNF before selecting the exact physical node (Algorithm *Add_chain*: lines 1-2). The nearest site is queried first for the available capacity of the chain. If the nearest site does not have available resources, then the algorithm checks if the VNFs have any requirements regarding physical placement and tries to place the VNFs according to their requirement. If the required cloud site is also overloaded and cannot accommodate the VNF v, the algorithm chooses another cloud site and reserves the resources for the service chain (Algorithm *Add_chain*: lines 3-10). Once the site is reserved, the algorithm then finds the physical node for the VNF placement. The algorithm first selects a node that has the least available resources, but it has at least the capacity to accommodate the service chain (Algorithm *Add_chain*: lines 17-18). If such a node is not available, it means that no individual node has the capacity to accommodate the service chain although the site in which the nodes are has the capacity. This

particularly happens when most of the nodes in a site are loaded 70%-80% or more, and the remaining capacity in a node is inadequate to accommodate all the VNFs in the chain. In that case, the chain needs to be splitted, and the allocation or the process *Add_chain* continues to find VNF placement with fewer/individual VNFs rather than the whole chain (Algorithm *Add_chain*: lines 26-28). On the other hand, if such a node that can accommodate the whole service chain is available, the algorithm *Add_chain* allocates the place and in the last step calls the function *Optimize_Allocation* (cf. Section 3.2.3) to check if the VNF placement can be optimized and resource usage efficiency can be increased using a holistic view of the system and resources (Algorithm *Add_chain*: lines 32-33).

Algorithm $Add\_chain(c \in C, s_{nearest} \in S)$

1: $Site\_allocation := \emptyset$ # An empty list
2: $A'_{site,node} := A_{site,node}, F'_{site,node} := F_{site,node}$ # Copy current allocation
3: **for** $\forall\ v \in c$
4: **if** $(s_{nearest} \neq \emptyset$ and $F'_{site}(s_{nearest}) \geq R_{capacity}(v))$
5: $s_{selected} := s_{nearest}$
6: **else**
7: **if** $((R_{site}(v) \neq \emptyset)$ and $(F'_{site}(R_{site}(v)) \geq R_{capacity}(v)))$
8: $s_{selected} := R_{site}(v)$
9: **else**
10: $s_{selected} := Random\_allocation()$ # Randomly allocate a site
11: **end if**
12: $A'_{site}(v) := s_{selected}$
13: $F'_{site}(s) := F'_{site}(s)$ - $R_{capacity}(v)$
14: $Site\_allocation.append(s_{selected})$
15: **for** $\forall\ s \in site\_allocation$
16: **while** $v \in V$ and $A'_{site}(v) == s$ and $A'_{node}(v) == \emptyset$
17: **for** $\forall\ \{v \mid A_{node}(v) == \emptyset\}$
18: find $n \mid s \in S(n)$ and $F'_{node}(n) \geq R_{capacity}(v)$ with lowest $F'_{node}(n)$
19: **end for**
20: **if** $(n == \emptyset)$
21: stop # nodes do not have adequate capacity
22: **else**
23: $F'_{node} := F'_{node}$ - $R_{capacity}(v)$
24: $A'_{node}(v) := n$
25: **end if**
26: **if** $(v \in V$ and $A_{site}(v) == s$ and $A_{node}(v) == \emptyset)$
27: split c # Split c and continue with individual v allocation
28: **end if**
29: **continue**
30: **end for**
31: **end for**
32: $A_{site,node} := A'_{site,node}, F_{site,node} := F'_{site,node}$ # Finalize current allocation
33: $Optimize\_allocation(N)$

3.2.2. Description of the algorithm *Delete_chain*

The algorithm *Delete_chain* is invoked with a chain *c* to be deleted from the system. This algorithm finds the node and site information in which *c* resides and de-allocates the placement from the site and node (Algorithm *Delete_chain*: lines 2-4). It also keeps track of the affected nodes where the service chains are initially placed (Algorithm *Delete_chain*: line 6). This

information can be used by the function *Optimize_Allocation,* which is referenced from *Delete_chain* to adjust and optimize the allocation again for efficiency since a service chain has been deleted (Algorithm *Delete_chain*: line 13).

Algorithm $Delete\_chain(c \in C)$

1: $N_{affected} := \{\}$
2: **for** $\forall \{v \mid v \in c\}$
3: $s := A_{site}(v)$
4: $n := A_{node}(v)$
5: **if** $(n \notin N_{affected})$
6: $N_{affected} := N_{affected} \cup \{n\}$
7: **end if**
8: $A_{site}(v) := \varnothing$
9: $A_{node}(v) := \varnothing$
10: $F_{site}(s) := F_{site}(s) + R_{capacity}(v)$
11: $F_{node}(s) := F_{node}(s) + R_{capacity}(v)$
12: **end for**
13: $Optimize\_allocation(N_{affected})$

3.2.3. Description of the algorithm *Optimize_allocation*

The algorithm *Optimize_allocation* shows how the total VNF placement and resource usage overview is holistically taken into account (different from single service chain scenario used in the previous two algorithms) to optimize the resource usage and VNF placement and free up lightly loaded nodes if possible. This algorithm gets a copy of the current allocation (Algorithm *Optimize_chain*: lines 2-4), and for each node in a particular side, lists all the VNFs (Algorithm *Optimize_chain*: lines 5-9). Once the list is populated, it tries with different mixes of nodes excluding some particular nodes and including others to check if a better placement is available (Algorithm *Optimize_chain*: lines 12-32). Firstly, the algorithm excludes the current node and all the unallocated nodes and tries to find a placement in the rest of the nodes for the VNFs (Algorithm *Optimize_chain*: lines 13-15). If such a node which can accommodate the VNFs exists, the algorithm compares and replaces the current allocation with the newer one if the new allocation improves the resource usage efficiency. If such a node is not available, the algorithm then goes on to find different mixes of excluded nodes and tries to find a better allocation (Algorithm *Optimize_chain*: lines 16-18). If it fails to find such a node, then it splits the service chain and finds the allocation for the VNFs in different nodes individually as was the case in the algorithm *Add_chain* (Algorithm *Optimize_chain*: lines 19-22). The final step of the algorithm compares the new allocation with the current one and updates the allocation if the new one improves the efficiency regarding resource allocation (Algorithm *Optimize_chain*: lines 33-36). The algorithm *Optimize_allocation* can have two different variants. Since this optimization may be a bit heavy burden on the system when the number of nodes and VNFs in the system are substantial, and it surfs through all the nodes in the system to find alternative placement. The other variant of the algorithm takes only a subset of nodes and tries to find effective alternative placement in that set only. An example of this set is shown in the algorithm *Delete_chain* which keeps track of the affected nodes due to a service chain deletion. The algorithm *Optimize_allocation* can be invoked with the affected nodes only, and the algorithm can then reiterate for a predefined number of iterations on this set of nodes to find alternative placement scenarios.

Algorithm *Optimize_allocation*($N_{affected} \subset N$)

1: $V' := \{\}$
2: $A'_{node} := A_{node}$
3: $F'_{node} := F_{node}$
4: $F'_{site} := F_{site}$
5: **for** $\forall \{n \mid n \in N_{affected}\}$
6: **for** $\forall \{v \mid A_{node}(v) == n\}$
7: $V' := V' \cup \{v\}$
8: $F'_{site} := F'_{site} - R_{capacity}(v)$
9: $F'_{node} := F'_{node} - R_{capacity}(v)$
10: **end for**
11: **end for**
12: **while** $V' \neq \emptyset$
13: $V_{tested} := V'$
14: $N_{excluded} := \{n\} \cup \{\forall n \mid S(n) == s$, unallocated_nodes_from_site(s)$\}$
15: n' := find n where $S(n) == s$, $n \notin N_{excluded}$ for chain c
16: **if** $(n' == \emptyset)$
17: n' := find n where $S(n) == s$, $n \notin N_{excluded}$
18: **end if**
19: **if** $(n' == \emptyset)$
20: split chain c and find allocation for individual VNFs
21: n' := find n where $S(n) == s$, $n \notin N_{excluded}$ for VNF v in chain c
22: **end if**
23: **if** $(n' == \emptyset)$
24: stop # No better allocation can be found
25: **else**
26: **for** $\forall \{v \mid v \in V'\}$
27: $F'_{site} := F'_{site} + R_{capacity}(v')$
28: $F'_{node} := F'_{node} + R_{capacity}(v')$
29: $A'(v') := n'$
30: **end for**
31: **end if**
32: **continue**
33: **if** size(A'_{node}) < size(A_{node})
34: $A_{node} := A'_{node}$
35: $F_{node} := F'_{node}$
36: $F_{site} := F'_{site}$
37: **go to** 1
38: **end if**

3.2.4. Final remarks

These algorithms particularly take into account the initial service chain placement as well as managing and moving the service chains or VNFs in the cloud according to the dynamic behavior of users as well as changing scenarios in terms of system load and resource usage. The system can anytime decide to replace a low priority VNF or service chain with a higher priority alternative on a particular edge cloud site, and such changes can be placed effectively using the aforementioned algorithms. The algorithm *Optimize_allocation* can be improved by including different heuristics in the decision-making process. However, for the sake of simplicity, we keep the alternative placement options to a minimum (as shown by the *excluded_node* list in the algorithm).

3.3. Instantiation and (re)placement of virtualized mobile core network VNFs

3.3.1. Optimal VNF instantiation and (re)placement

The core sub-slice includes the elements that correspond to the 4G/5G core network. These elements naturally support virtualized implementation and instantiation. NFV facilitates tailoring a slice to the needs of its respective service. Allowing the deployment of a virtualized core network over an NFVI comes with the flexibility to customize the compute and other resources allocated to the slice. It also allows the slice to scale on demand for cost and performance optimization and to select the appropriate functional configuration in terms of optimal placement of the core network components (e.g. decide on the number of VNF instances for each core network function to maximize reliability). These tasks are supported by maturing relevant standards such as the ETSI NFV Management and Orchestration (NFV-MANO) [20] framework and available implementations of MANO software stacks. Even though a prior deployment of the core sub-slice elements may be deemed optimal in a *CIN*, momentary conditions of the network due to changes in user requirements and network dynamics may cause the placement of the elements to become sub-optimal.

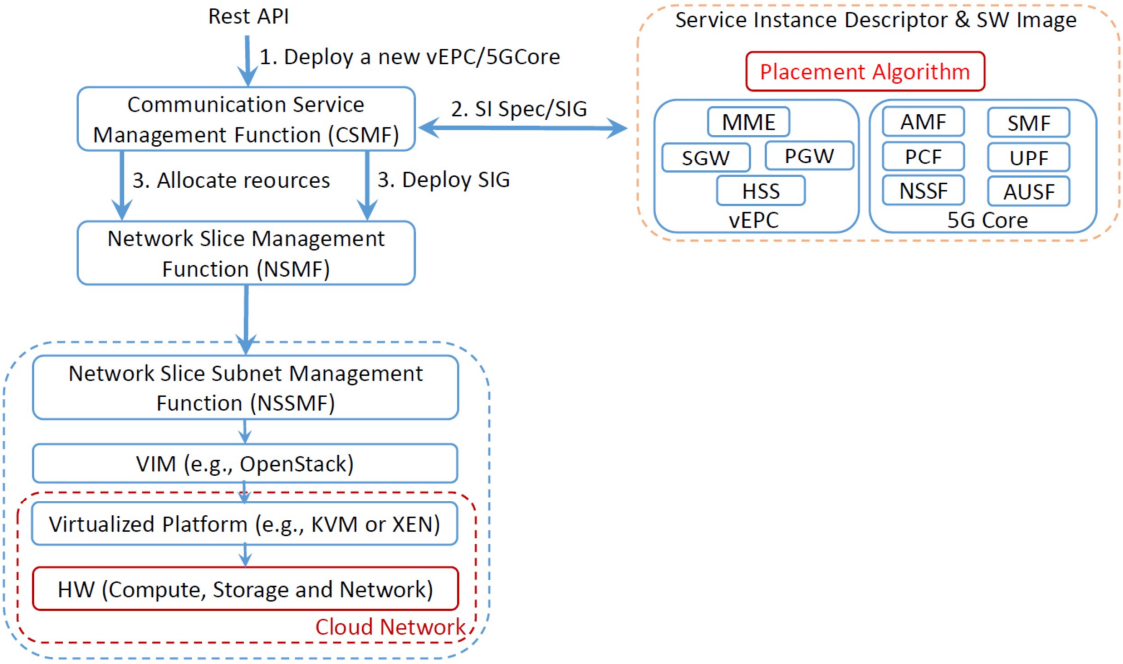


Fig. 4 – NFV-based vEPC/5G Core service instantiation architecture

Therefore, unlike the cited research work [5], [6], [7] and [11]-[19], which tackle either the optimal number of VNFs or the VNF placement, the proposed solution jointly addresses both issues at the same time. Moreover, these research works assume that *CINs* belong to the same cloud operator, which is relaxed in our proposed solution (i.e. *CINs* may belong to different cloud operators). To sum up, the proposed framework aims at finding: *(i)* the optimal number of VNFs to deploy (according to mobile traffic) and *(ii)* the optimal placement of the determined VNFs over the underlying federated cloud. Our proposed framework follows the high-level architecture depicted

by Fig. 4, where an NFV-based vEPC/5G core service instantiation scheme is envisioned following the 3GPP technical report in [21].

In Fig. 4, the NFV-based architecture consists of three main parts: *(i) ClNs,* whereby each *ClN* is managed by a Virtual Infrastructure Manager VIM (e.g. OpenStack), *(ii)* Network Slice Management Function (NSMF) and Network Slice Subnet Management Function (NSSMF) that are responsible for managing, monitoring and orchestrating all VNFs in different *ClNs*, and (iii) a CSMF that is responsible for: (a) translating the communication service related requirements into network slice related requirements, and (b) communication with NSFM.

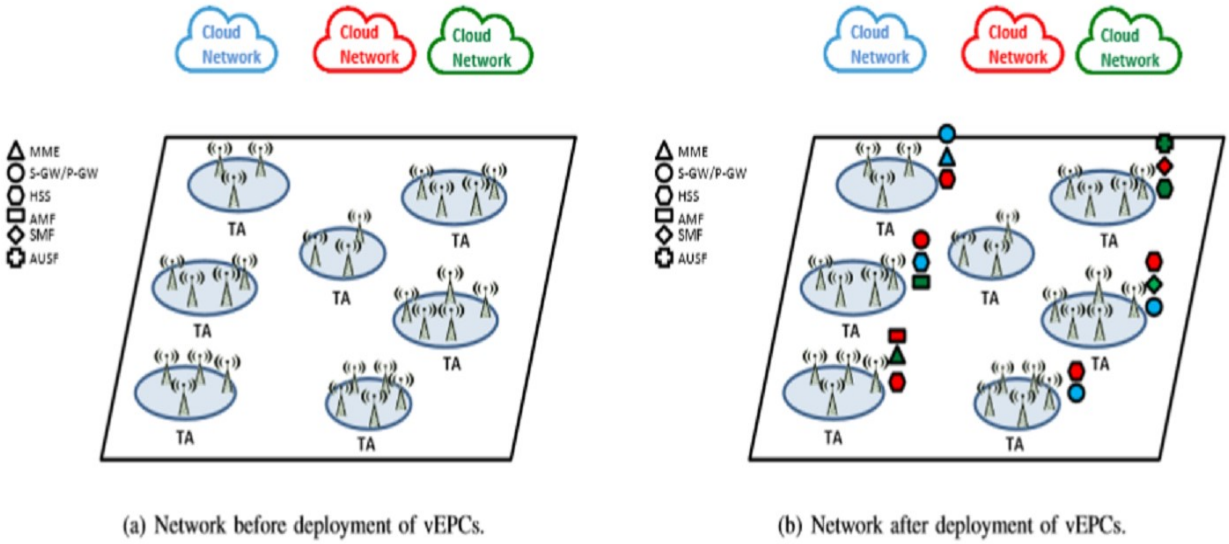


Fig. 5 – NFV-based vEPC/5G Core service instantiation and placement architecture

Following the above service instantiation architecture, a vEPC/5G core deployment and dynamic VNF service placement architecture is developed in order to model the dynamic VNF re/placement of virtual network elements in a *ClN* based on a predetermined set of events that could occur in an LTE network. Fig. 5 illustrates the main overview of the envisioned optimal core network sub-slice's element placement framework. Fig. 5a depicts a network architecture that consists of: *(i)* three *ClN*s managed by three different VIMs (i.e. Rackspace, OpenStack and Amazon AWS) and *(ii)* a set of TAs that form the RAN. As discussed above, using the framework, vEPC (alternatively virtual 5G Core in case of 5G core network functions) will be instantiated to manage and handle the traffic generated by the RANs in these TAs. As depicted in Fig. 5b, the VNF deployment framework will specify the number of instances of each VNF ϑ, their flavors, as well as their locations in different *ClNs*. In this figure, we present only how to instantiate vEPC, however, following the same fashion, the virtual 5G core network slice can be also instantiated. Moreover each TA will be assigned different instances of different VNFs ϑ that are responsible for providing connectivity in that TA. Using the framework, a Service Instance Graph (SIG) will be created for use by the NFVO. The latter will communicate the appropriate information in SIG to different VIMs, managing different *ClNs*, in order to instantiate the specified VNFs. In this figure, the square and triangle shapes represent HSS and MME, respectively, while SGW and PGW are represented with a circle. Different colors of the shapes beside TAs indicate the *ClNs*, where these

VNFs are instantiated. Note that this picture is used to ease the understanding of the idea behind our proposed framework. In real life, one instance of a VNF may handle multiple TAs.

3.3.2. Model for an optimal number of VNF deployment

The set of mobile network events upon which our dynamic VNF (re)placement framework was developed are shown in Table 3 below.

Table 3 – A sample from a list of possible events in a mobile network

Event	Description
Attach	When a UE attaches to the network
Detach	When a UE detaches from the network
PDN connection setup	When a UE connects to a PDN network
PDN disconnection	When a UE disconnects from the network
Handover	When a UE moves between two neighboring eNodeBs
Tracking area update	When a UE enters into a new tracking area. Such TA update can happen with or without SGW relocation
S1-release	Releasing S1 connection when a UE becomes inactive

For every event in the network, such as attach or detach of a UE, a set of procedures should be executed. Each procedure generates tens of messages exchanged among the different network components. We denote by $\lambda_{x,\vartheta}$ the amount of resources in terms of CPU and memory required by an instance of a VNF ϑ to handle an event $x \in \Upsilon$. Formally, $\lambda_{x,\vartheta}$ is defined as the amount of virtual resources needed by ϑ to handle the number of messages generated due to the event x. It is obvious that an event (e.g. S1-release, TAU, and attach) does not necessary have the same impact on a particular ϑ (MME, HSS, SGW or PGW). Moreover, some events $x \in \Upsilon$ do not have any impact on some ϑs. In this case, the value of $\lambda_{x,\vartheta}$ is simply set to zero. For example, the event S1-Release does not have much impact on HSS, hence, $\lambda_{S1-Release,HSS}$=0. Table 3 shows a list of some events (and their descriptions) that can occur in vEPC/5G Core. For a specific event, not necessarily all concerned VNFs would be involved. For example, in case of TAU without SGW relocation, only the MME would communicate with eNodeBs, informing the current SGW of the target eNodeB but not involving a new SGW. Another example is the X2-based handover: when a UE moves from one eNodeB to another one, a set of messages would be forwarded from these eNodeBs to MME. If these eNodeBs are belonging to different SAs, SGW relocation would occur involving a new SGW in the process to start the SA relocation process.

We assume that each *ClN* offers virtual resources with a set of flavors L^{ClN} in terms of capacities. Each flavor would be used as a model class to instantiate the different VNFs. According to the required resources, each *ClN* would use the appropriate flavors to increase QoS; its profits should be increased while QoS and functionality of the system are not impacted negatively. According to the amount of resources dedicated for every flavor, the instance of a VNF that uses it would handle a specific rate of traffic (i.e. the amount of traffic per second). The more resources used by that instance, the higher the traffic rate it can handle. We denote by MAX_l^{ClN} the maximum traffic rate that can be handled by an instance of VNF with a flavor $l \in L^{ClN}$. On the other hand, the cost of an instance of VNF increases proportionally with the resources used in its flavor. The more the

resources required in a flavor are, the higher the cost of the VNF becomes. We assume that each *ClN* has different flavors with different costs. τ_l^{ClN} denotes the cost of a flavor *l* in a *ClN*.

Before the execution of the proposed algorithm, the network is observed for a specific learning period *D*, when information about different events is gathered from the network. We denote by Υ the set of events that occurred in the network during the period *D*. To facilitate the management of our system; we adopted a discrete event dynamic system (DEDS) to observe the events during the period *D*. The time is counted only for an event $e \in \Upsilon$ that has occurred in the network. We denote by T the time in da iscrete format, where each element $t \in T$ represents the time corresponding to one or multiple events. Ω denotes the set of TAs in the network. Each TA $A \in \Omega$ is responsible for a set of events that would be handled by vEPC/5G Core. The number of events of type $x \in \Upsilon$, generated from a TA $A \in \Omega$ until a specific time $t \in T$, is denoted by $\Gamma_A^x[t]$. If a set of TAs is handled by the same instances of a VNF, some events would be omitted. For example, if TA $A \in \Omega$ and $B \in \Omega$ are handled by the same instance of SGW, TA service request event would not be generated, if a UE moves from A *to* B within Ω. $\Phi_{A,B}^x[t]$ denotes the number of events of type $x \in \Upsilon$, which would be omitted, if TAs $A, B \in \Omega$ use the same instance of a VNF.

Each VNF in vEPC/5G Core $\vartheta \in V$ (e.g. SGW or MME) has a specific role to accomplish. This means that two VNFs with two different roles should not be considered as one VNF. Moreover, the instantiation of an instance of a VNF does not have any impact on the number of generated events on another instance of another VNF. We model the instantiation of different instances of each VNF $\vartheta \in V$ as a coalitional game. The system model considers the problem whereby the *ClNs* collaborate to form vEPC/5G Core, in order to increase their profits. The profit of a *ClN* refers to the difference between *P*, i.e. the price that the operator is willing to pay and the cost needed to handle the traffic generated from different TAs associated to the *ClN*, in terms of CPU, storage, and VM management. For the sake of simplicity and without loss of generality, we assume that the price *P* can be split into multiple parts, where each one would be used for deploying a VNF ϑ (i.e. HSS, MME, SGW and PGW). Let P_ϑ denote the price for deploying a VNF ϑ. If the QoS required in vEPC/5G Core is not ensured, no profit is gathered from holding different VNFs. We assume that every *ClN* is selfish and therefore will not host VNFs, if it cannot make some profit.

Algorithm1 illustrates the steps of the proposed framework. The framework starts by constructing T (Algorithm 1: Line 1). Based on the abovementioned remark, the instantiation of VNFs with different roles can be done sequentially. For this reason, we create different VNFs of vEPC/5G Core one by one (Algorithm 1: Line 2). For each VNF $\vartheta \in V$, we create all the required instances in different *ClNs* to handle the traffic generated from different TAs. For every component and at each time $t \in T$, the cumulative number of events generated from different TAs $A \in \Omega$, which allocates the VNF ϑ is computed (Algorithm 1: Lines 3-5). Then, for every pair of TAs $A, B \in \Omega$ and at each time $t \in T$, the cumulative number of events that would be removed, if they are handled by the same VNF is also computed (Algorithm 1: Lines 6-8). Finally, a coalitional game is executed for every VNF ϑ to create instances of ϑ in different *ClNs*, such that the individual profit of each *ClN* in the coalition is maximized.

Algorithm 1 Algorithm of virtual-EPC

Input:

Υ: The set of events that can occur in the network.
Ω: The set of all tracking areas.
$\mathcal{V}$: The set of VNF types that would be deployed in vEPC.
Σ: The set of all data centers.

$\mathcal{T} = computeT();$
for all $\vartheta \in \mathcal{V}$ **do**
  for all $(\mathcal{A}, x) \in (\Omega, \Upsilon)$ **do**
    $\Gamma^{x}_{\mathcal{A}} = computeΓ(\vartheta, \mathcal{T}, \mathcal{A}, x);$
  end for
  for all $((\mathcal{A}, \mathcal{B}), x) \in (\Omega^2, \Upsilon)$ and $\mathcal{A} \neq \mathcal{B}$ **do**
    $\Phi^{x}_{\mathcal{A},\mathcal{B}} = computeΦ(\vartheta, \mathcal{T}, \mathcal{A}, \mathcal{A}, x);$
  end for
  $\Xi = instanceVNF(\vartheta, \Gamma, \Phi);$
  $bestCoalition(\Xi);$
end for

3.3.3. Model for optimal VNF instantiation

Here, we expound on the procedure "*instanceVNF*" presented in Algorithm 1 to instantiate the different instances of one VNF $\vartheta \in V$ (e.g. MME/SGW). We model this problem as a coalitional game, whereby the different *ClN*s are the players desiring to increase their individual profits. We assume that each *ClN* has enough resources to accommodate the different instances needed by vEPC/5G Core. We also assume that each instance has enough resources allocated to it to handle the traffic generated from at least one TA. We are mainly interested in instantiating virtual resources to accommodate the instances of one VNF ϑ. According to the required resources, *ClN* would use the appropriate flavors to increase QoS; its profits should be increased while ensuring that QoS and functionality of the system are not negatively impacted. According to the amount of resources dedicated for every flavor, the instance of a VNF that uses that flavor would handle a specific rate of traffic (i.e. the amount of traffic per second).

For every VNF ϑ (i.e. SGW, PGW or MME), a set of *ClN*s (S) would participate in a coalition to host all the instances of ϑ. Every cloud network $ClN_i \in S$ will handle the traffic caused by a set of TAs $\omega_i \subseteq \Omega$. We define the handling of TAs by a cloud network $CN_i \in S$ as a function $\mathcal{H}: \Sigma \rightarrow \Omega$, where $\mathcal{H}(ClN_i) = \{\omega_i\}$. The different centers agree to form a coalition by ensuring the following two properties:

1) System functionality: Every TA would be associated with at least one instance of the VNF ϑ; i.e. there should be no TA A that has no instance of the VNF ϑ. Formally, $\forall A \in \Omega \Rightarrow \exists ClN \in S \wedge A \in \mathcal{H}(ClN)$. Also, the different *ClN*s in a coalition S should not be overloaded, especially during the peak hours, when the majority of UEs are connected. This allocation of *TA*s to *ClN*s should not degrade the QoS (i.e. reduce the quality);

2) *ClN* profit: $\mathcal{H}$ should be defined in a way that maximizes the profits of the coalition members. In order to achieve this objective, two functions are defined:

   I. $\mathcal{F}(\vartheta, ClN, \omega_i \subseteq \Omega)$: A function defined as $\mathcal{F}: V \times \Sigma \times \Omega \rightarrow \mathrm{IR}^{+}$, which returns the minimum cost to create different instances of the VNF ϑ at ClN, in order to handle the traffic

generated by a set of tracking areas ω. This function ensures the use of optimal flavors L^{ClN} to create different instances with minimum cost, while equally ensuring the system functionality;

II. $\mathcal{G}(\vartheta, S, \Omega)$: A function defined as $\mathcal{G} : V \times \Sigma \times \Omega \rightarrow \mathrm{IR}^{+}$, which returns the minimum possible cost to create different instances of the VNF ϑ in a coalition S to handle the mobile traffic. Besides the optimal allocation of different TAs to different *ClN*s in the coalition, this function guarantees the use of the optimal flavors in each *ClN* to create different instances of ϑ. The set of all TAs ω would be divided among the different coalition members S. In order to increase the profit, each TA should be associated with only one instance of ϑ in a specific *ClN*. Therefore, $\forall ClN_i, ClN_j \in S \wedge ClN_i \neq ClN_j \Rightarrow \mathcal{H}(ClN_i) \cap \mathcal{H}(ClN_j) = \emptyset$.

Based on the solutions of the optimization problems for $\mathcal{F}(\vartheta, ClN, \omega \subseteq \Omega)$ and $\mathcal{G}(\vartheta, S, \Omega)$ which are not presented in this document, Algorithm 2 was developed for optimal orchestration and placement of vEPC/5G core based on coalitional games.

Here, we explain the function $instanceVNF(\vartheta, \Gamma, \phi)$ that uses the coalitional game to deploy the instances of ϑ across different *ClN*s. In this function, we assume that the QoS desired for a *TA A* can be assured by

Algorithm 2 $instanceVNF(\vartheta, \Gamma, \Phi)$

Input:
ϑ: A component of *vEPC*.
Γ: The number of cumulative events.
Φ: The number of cumulative events that would be omitted.

1: $\Xi = \{\{\mathcal{DC}_1\}, \{\mathcal{DC}_2\}, ..., \{\mathcal{DC}_{|\Sigma|}\}\}$;
2: $visited = \emptyset$;
3: **while** $True$ **do**
4: $stop = True$
5: **for all** $\mathcal{S} \in \Xi$ **do**
6: **if** $\mathcal{S} \notin \Psi$ **then**
7: $\Psi[\mathcal{S}] = v(\mathcal{S})$
8: **end if**
9: **end for**
// Merging process
10: **for all** $\mathcal{S}_i, \mathcal{S}_j \in combinations(\Xi, 2) \setminus visited$ **do**
11: $visited = visited \cup \{(\mathcal{S}_i, \mathcal{S}_j)\}$;
12: **if** $\{\mathcal{S}_i \cup \mathcal{S}_j\} \notin \Psi$ **then**
13: $\Psi[\mathcal{S}_i \cup \mathcal{S}_j] = v(\mathcal{S}_i \cup \mathcal{S}_j)$
14: **end if**
// Using Ψ the values of $\Pi_{\mathcal{S}_i}$, $\Pi_{\mathcal{S}_j}$ and $\Pi_{\mathcal{S}_i \cup \mathcal{S}_i}$ are computed
15: **if** $\{\mathcal{S}_i \cup \mathcal{S}_j\} \rhd_m \{\{\mathcal{S}_i\}, \{\mathcal{S}_j\}\}$ **then**
16: $\Xi = \Xi \setminus \{\mathcal{S}_i\}; \Xi = \Xi \setminus \{\mathcal{S}_j\}; \Xi = \Xi \cup \{\mathcal{S}_i \cup \mathcal{S}_j\}$;
17: $stop = False$;
18: $break$;
19: **end if**
20: **end for**
// Split process
21: **for all** $\mathcal{S} \in \Xi \wedge |\mathcal{S}| > 1$ **do**
22: $break = False$;
23: **for all** $\{\mathcal{S}_i, \mathcal{S}_j\} \in \mathcal{S} \wedge \mathcal{S}_i \cup \mathcal{S}_j = \mathcal{S} \wedge \mathcal{S}_i \cap \mathcal{S}_j = \emptyset$ **do**
24: **if** $\mathcal{S}_i \notin \Psi$ **then**
25: $\Psi[\mathcal{S}_i] = v(\mathcal{S}_i)$
26: **end if**
27: **if** $\mathcal{S}_j \notin \Psi$ **then**
28: $\Psi[\mathcal{S}_j] = v(\mathcal{S}_j)$
29: **end if**
// Using Ψ the values of $\Pi_{\mathcal{S}_i}$, $\Pi_{\mathcal{S}_j}$ and $\Pi_{\mathcal{S}}$ are computed
30: **if** $\{\{\mathcal{S}_i\}, \{\mathcal{S}_j\}\} \rhd_s \mathcal{S}$ **then**
31: $\Xi = \Xi \setminus \{\mathcal{S}\}; \Xi = \Xi \cup \mathcal{S}_i; \Xi = \Xi \cup \mathcal{S}_j$;
32: $stop = False$;
33: $break = True$;
34: **break**;
35: **end if**
36: **end for**
37: **if** $break = True$ **then**
38: **break**;
39: **end if**
40: **end for**
41: **if** $stop = True$ **then**
42: **break**;
43: **end if**
44: **end while**
45: **return** Ξ;

every *ClN*. Since both functions $\mathcal{F}$ and $\mathcal{G}$ require an important amount of resources, therefore, we use dynamic programming technique when implementing the function $instanceVNF(\vartheta, \Gamma, \phi)$. The function $\mathcal{F}$ should not be computed twice for the same *ClN* and the same $\omega \in \Omega$. Similarly, the function $\mathcal{G}$ should not be computed twice for the same subset of *ClN*s S. Algorithm 2 is used to explain the general functionality of $instanceVNF(\vartheta, \Gamma, \phi)$. Due to space limitation, the calculation redundancy of function $\mathcal{F}$ is not presented in Algorithm 2.

The function $instanceVNF(\vartheta, \Gamma, \phi)$ first starts by forming a collection Ξ by putting every player *ClN* in a separate coalition (Algorithm 2: Line 1). Then, a variable $visited$ is initialized by $\emptyset$ (Algorithm 2: Line 2). The variable *visited* is used to keep track of every pair of coalitions which was already visited for the merge. Every visited pair of coalitions will be put in the set *visited*. Then, a while loop is executed where the merge and split processes are executed repetitively until achieving the ID_p-stable collection (Algorithm 2: Lines 3-44). $instanceVNF(\vartheta, \Gamma, \phi)$ computes the values of $v(S)$ using $v(S) = \begin{cases} 0 \; if \; |S| = 0 \; or \; QoS \; is \; not \; ensured. \\ \mathrm{P}_\vartheta - \mathcal{G}(\vartheta, S, \Omega) \text{ otherwise} \end{cases}$ and ensures that the function $\mathcal{G}$ is executed only when needed (Algorithm 2: Lines 5-9). To prevent the redundancy in the computation, the vector Ψ is used to store the values of $v(S)$.

In $instanceVNF(\vartheta, \Gamma, \phi)$, the merge and split processes are executed one after the other. In other words, only one merge (Algorithm 2: Lines 10-20) would be executed, and then only one split (Algorithm 2: Lines 21-40) would be executed until achieving the ID_p-stable collection. In the merging process, every pair of coalitions $S_i \; and \; S_j$, which are not yet visited, are tested if they can be merged or not (Algorithm 2: Line 10). These pairs of coalitions are put in the vector *visited* to prevent redundancy (Algorithm 2: Line 11). To prevent the execution of the function $\mathcal{G}$ twice, the value of $v(S_i \cup S_j)$, will be put in the vector Ψ (Algorithm 2: Lines 12-14). If the merging condition $(\{S_i \cup S_j\} \rhd_m \{\{S_i\} \cup \{S_j\}\})$ is verified, we merge these coalitions in the same coalition, and then exit the merging process to execute the split process (Algorithm 2: Lines 15-19). Otherwise, another pair of coalitions which was not visited yet, will be tested. Meanwhile, in the split process, we will consider every coalition S that has more than one player *ClN* (Algorithm 2: Line 21). We try to split every two sub-coalitions $S_i \; and \; S_j$ of S that satisfy the following conditions: *(i)* $S_i \cup S_j = S$; *(ii)* $S_i \cap S_j = \emptyset$ (Algorithm 2: Line 23). The partitioning of the coalition S is done through the partitioning of an integer into two parts [22]. For example, the coalition $\{ClN_1, ClN_2, ClN_3\}$ can be presented with a number 7 (i.e. binary 111), whereas the coalitions $\{ClN_1, ClN_3\}$ and $\{ClN_2\}$ would be presented with the numbers 5 (i.e. binary 101) and 2 (i.e. binary 010), respectively. Enumerating all the possible two sub-coalitions of S that satisfy the condition in Algorithm 2: Line 23 is equivalent to finding all the two numbers whereby the sum of these numbers equals to the number that represents S. Using the same approach, the redundancy in the computation of $\mathcal{G}$ is also prevented in the split process (Algorithm 2: Lines 24-29). Then, the function $instanceVNF(\vartheta, \Gamma, \phi)$ splits S if it is better for the collection Ξ (Algorithm 2: Lines 30-35). If one split succeeds, we exit the split process and re-initiate the merge process. Note that the variable *stop* will be set to *false* if only one merge or one split is carried out, and then the algorithm keeps repeating the loop (Algorithm 2: Lines 3-44) until achieving the ID_p-stable collection. Then, no further merge or split processes will be carried out.

The results of the above-described algorithms are presented and discussed in the next sub-section.

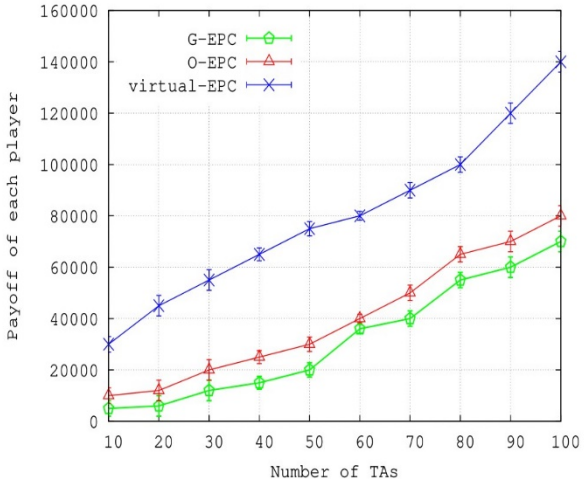
3.3.4. Results and analyses

In this section, we evaluate the performance of the proposed scheme to instantiate vEPC/5G Core instances over a federated cloud of *ClN*s. As comparison terms, we use two trivial solutions. The first one, named *G-EPC*, uses the Grand coalition Σ for the deployment of *vEPC/5G Core*. In this solution, all *ClN*s participate in creating different instances of each VNF ϑ, whereas the second solution uses a random deployment over *ClN*s. In this solution, a greedy algorithm is used, whereby different *ClN*s are randomly selected one by one, as well as the instantiations of VNFs are randomly created one by one, until the traffic generated from Ω is handled. Only then, the greedy algorithm is finished and the random coalition of *ClN*s is selected. The algorithms are evaluated in terms of the following metrics:

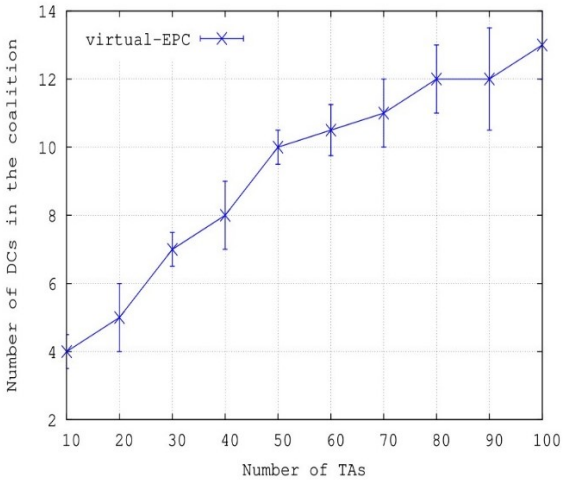
- The payoff of individual *ClN*: is defined as the average value of individual payoffs for each player in the selected coalitions of different instances of each VNF ϑ;
- A number of merge and split: is defined as the average number of merge and split operations needed to deploy each VNF ϑ. This metric shows the complexity of the proposed scheme;
- Number of *ClN*s in the selected coalition: is defined as the average number of players in the selected coalition for each instance of each VNF ϑ.

The algorithms are evaluated using the Python programming language and an extended package for graph theory called networkx [23]. We implement the proposed scheme using IBM ILOG CPLEX version 12.6.1, using the branch-and-bound method to solve the optimization problems. We used historical data from real-life mobile operator network to evaluate the different solutions; i.e. the different events generated in the network, such as attach or detach operation of a UE, the executed procedure and the number of generated messages. In the simulation results, each plotted point represents the average of 10 executions. The plots are presented with 95% confidence interval. The different algorithms are evaluated by varying the number of TAs and the number of *ClN*s. We conducted two sets of experiments. Firstly, we vary the number of TAs and fix the number of *ClN*s to 15. In the second scenario, we vary the number of *ClN*s while fixing the number of *TA*s to 50. The value of P – the price that a vEPC/5G Core operator is willing to pay – is set proportional to the number of *TAs* in the network.

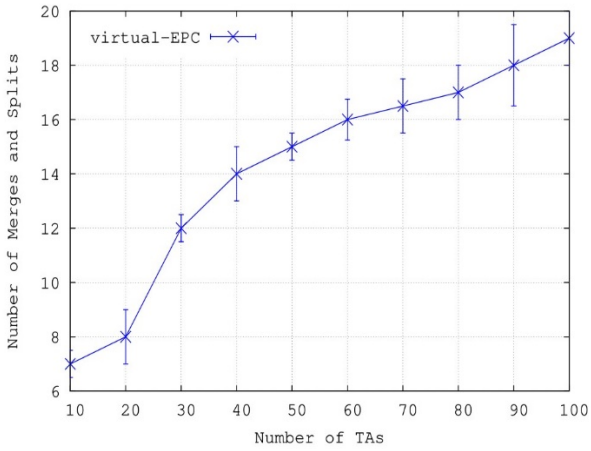
Fig. 6 shows the performance of the three solutions for a varying number of *TAs*. Fig. 6a depicts the impact of the number of TAs on individual payoffs of each *ClN*. We clearly observe that the proposed solution outperforms both base-line approaches. Moreover, we remark that the use of grand coalition or only one coalition does not yield an optimal solution.



(a) Payoff of each *ClN*



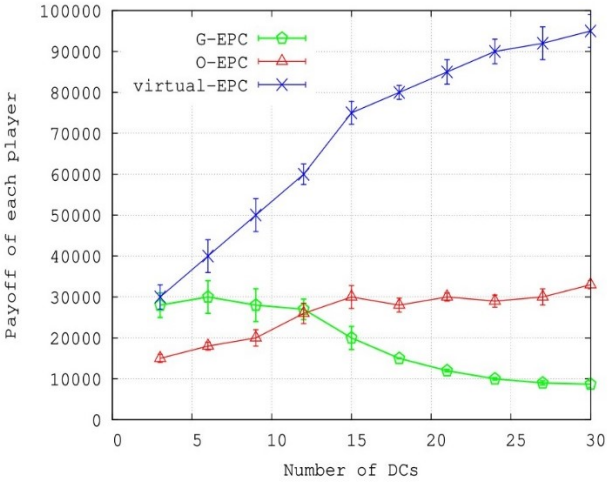
(b) Number of *ClNs* in the selected coalition



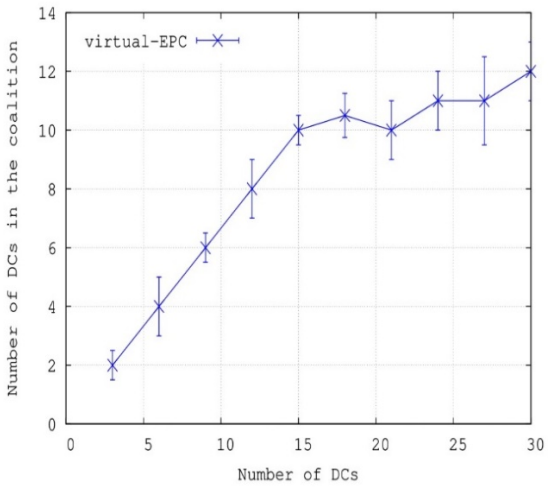
(c) Number of merge and split operations

Fig. 6 – Performance evaluation of the proposed vEPC scheme for a varying number of *TAs*

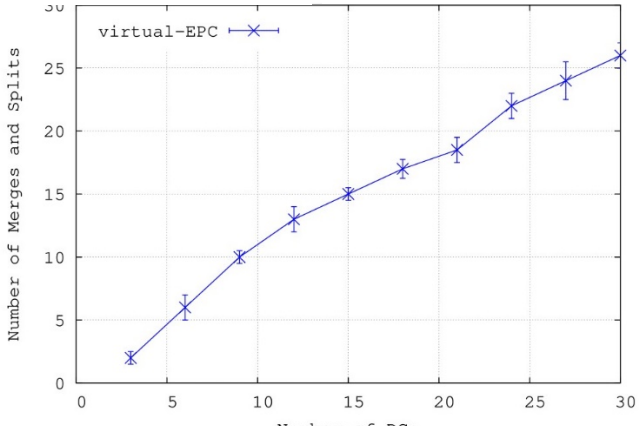
From this figure, an increase in the number of TAs has a positive impact on the individual payoffs in all the solutions. For 100 TAs, the proposed solution achieves an individual payoff of 140000, while the individual payoff of base-line approaches does not exceed 80000, which represents an enhancement of 75%. Indeed, the proposed solution succeeds in forming the optimal coalition for each instance among all the players *ClN*s, which reduces the cost and hence increases the profit of each player in the selected coalition. In Fig. 6b we notice that the number of involved *ClN*s increases proportionally with the number of *TAs* in the network; from which we conclude that the proposed solution uses the average number of *ClN*s to form vEPC/5G Core instances. On the other hand, we observe from Fig. 6c that the number of merge and split operations in the proposed solution does not exceed 20. This means that the proposed solution converges to the optimal solution within a reasonable time. From this figure, we also observe that the number of *TAs* has a negative impact on the number of merge and split operations. The higher the number of *TAs*, the higher the value of $\mathcal{F}(\vartheta, ClN, \omega \in \Omega)$, and consequently the more split and merge operations.



(a) Payoff of each *ClN*



(b) Number of *ClNs* in the selected coalition



(c) Number of merge and split operations

Fig. 7 – Performance evaluation of virtual-EPC for varying numbers of *ClNs*

Fig. 7 depicts the performance of the three solutions for varying numbers of players *ClN*s. In Fig. 7a, we plot the evaluation of individual payoff of each *ClN*. From this figure, we remark that an increase in the number of players has a positive impact on the individual payoff of each player in the selected coalitions formed by the proposed solution. Furthermore, we remark that *G-EPC* outperforms *O-EPC* when the number of players is less than 12 *ClN*s. When the number of *ClN*s exceeds 12, *O-EPC* outperforms *G-EPC*. This can be explained by the fact that when the number of *ClN*s is less than 12 (the ratio of TAs per *ClN* is high), then the players in the selected coalitions equitably participate in handling the different *TAs*. Whereas when the ratio of TAs per *ClN* decreases, many *ClN*s in *G-EPC* end up sharing the profits for handling only a small number of *TAs*. In contrast to these solutions, the proposed solution efficiently selects the coalitions, such that the profit of the players is increased as much as possible. For example, in case the number of *ClN*s is 30, the proposed scheme outperforms the baseline approaches with more than 233%. Fig. 7b shows that the number of *ClN*s in the selected coalitions increases proportionally with the number of *ClN*s in the network. The higher the number of *ClN*s in the network is, the higher the likelihood to select them in the best coalition becomes. Fig. 7c shows that the number of merge and split operations in the proposed solution increases proportionally with the number of players

CINs. An increase in the number of players leads to an increase in the number of possible combinations, which intuitively has a negative impact on the number of merge and split operations. Finally, we observe from this figure that the number of merge and split operations still does not exceed 25; meaning that the proposed solution would converge to the optimal solution within a reasonable time.

3.4. Scalability of the 5G AMF (or 4G MME)

One of the important roles of the AMF is to handle access control and mobility, i.e. attach requests and mobility management procedure of UE. Although each slice may have its dedicated AMF, it is envisioned as a common function among network slices as in [24]. Consequently, the AMF requires a specific focus when dimensioning its resources, as it represents the system bottleneck in 5G. The objective of this section is to provide an algorithm based on Control Theory allowing: *(i)* to equilibrate the load on the AMF instances in order to maintain an optimal response time with limited computing latency; *(ii)* to scale-out and –in the VNF resources running the AMF.

3.4.1. Existing approaches to AMF/MME scalability

In legacy 4G EPC, MNOs develop different strategies to load balance the MMEs in a way to avoid network congestions. Some, for example, deploy multiple MMEs for granted zone, and use the probabilistic distribution of UE arrival at the eNodeB level. Others may just deploy one over-dimensioned MME for a given zone, and thus no load balancing procedure is needed at the eNodeB. However, experiences have shown that even an over-dimensioned MME is subject to persistent overloads [25]. Besides load balancing, dynamic scaling, as offered by NFV, is not really possible in 4G due to the difficulties to instantiate new MMEs on demand.

Nevertheless, in 5G, modern opportunities emerge to tackle this issue in an economical and intelligent manner. Indeed, with SDN and NFV, the network control functions of the MME are virtualized and hosted in the cloud. Thus, the deployment of a new MME will be a matter of software deployment. This modern way of dimensioning the network is cheaper and faster than the classical solution that is based on hardware. In fact, many studies and reflections were conducted on the adaptation of SDN and NFV for 5G networks, i.e. in [26] and [27]. In [28] and [29], authors propose analytical models providing a quick way to help mobile operators to plan and design network optimization strategies without large-scale deployment, thereby saving on cost and time. In [30], authors present an NFV-based traffic offloading framework architecture using vEPC, aiming at enabling on-demand traffic offload when the legacy EPC network capacity is reaching an offload threshold. This can be considered as a transition solution from 4G to 5G.

Some other works have been conducted addressing the MME scaling and load balancing issues within the context of 5G. Authors in [31] propose a new model of state-full MME. They propose to split the MME into three parts: 1) the traffic sorter, 2) the processing functions and 3) the state database. This approach proposes creating groups of IMSIs based on hash value and assigning a processing function instance to a group of UEs. Based on the IMSI group, the traffic sorter will route UE requests to the specific processing function. However, this limits the scalability of the

processing functions, as there should always be at least one worker available to serve each group of UEs. Further, this work proposes only horizontal scaling (processing functions scaling), which is limited by the processing capability of the traffic sorter, as it is the single point of access to the MME node; hence, not addressing MME node scalability.

A distributed MME model is also proposed in [32] and [33]. In contrast with the approach proposed in [31], the MME model is stateless and based on an external users' state storage system. Thus, the migration between MMEs is limited only to UEs in an idle state. However, in the active state, UEs attach to an MME instance. Therefore, if another MME instance receives a network event for that UE, the request has to be forwarded to the correct MME, hence increasing latency of EPC procedures.

Authors in [34] and [35] propose a stateless vMME that is split into three logical components: 1) front-end (FE); 2) MME service logic/Worker (SL); and 3) state database (SDB). The authors assume that MME SL can handle any request from any UE as they are stateless. However, as the users' state database is local and is not shared with the other vMMEs, this limits requests' handling within the same MME. The authors also propose to use one FE element, which may represent a congestion point.

3.4.2. Proposed approach

3.4.2.1 Architecture

We assume the new 5G Core architecture based on the 3GPP work [24]. The overall architecture is depicted in Fig. 8, which illustrates the new core NFs, where some are the result of a split of the current EPC functions, such as the MME. Reference interfaces will hold the communication between those functions, noted as N_x as depicted in Fig. 8. For further information regarding the other functions, the interested reader may refer to [24].

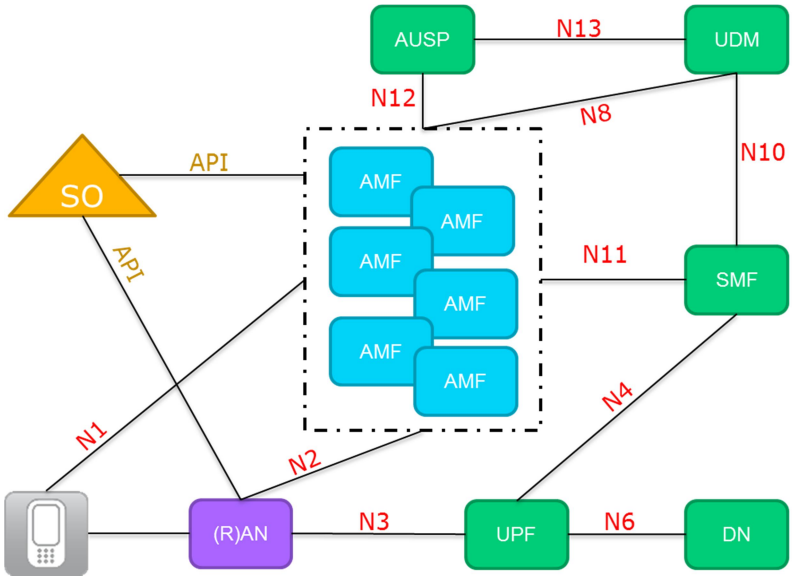


Fig. 8 – Envisioned 5G architecture

In the NGC architecture, the legacy MME component is split into three NF: the AMF, the SMF and the UDM. The AMF handles only access control and mobility requests, the SMF will manage the

UE sessions, and the UDM will handle the UE contexts. Indeed, this approach offers more flexibility to the core network, but still, the AMF may suffer from signaling traffic overload. Therefore, we introduce a mechanism that addresses the AMF scalability by providing a novel scale in/out algorithm depending on the network load.

The scaling in and out process will be run at the NFVO. The latter will be in charge of running the proposed control model, described below, and deploying new AMF instances when needed. The NFVO will rely on a VNFM, which uses management interfaces (or API) to communicate with the EM of the VNF running the AMF. Moreover, the NFVO will use the VNFM to configure the new instances of the AMF and the Load Balancer, if the number of VNF running AMF is more than one.

A. Traffic steering

As mentioned earlier, the traffic load will be distributed over the AMF instances using the proposed control-based algorithm. We assume that once a new instance is created, the VNFM, according to the NFVO, will configure a load balancer to enforce the traffic steering among AMF based on the proposed algorithm.

B. Scale-out

When the control system detects the need of a scale-out for the AMF, the NFVO triggers the instantiation of a new AMF in the architecture. Also, it notifies the VNFM about the creation of the additional AMF. The VNFM will accordingly configure both the new AMF and the load balancer.

C. Scale-in

When the control system detects a possibility for a scale in, the NFVO notifies the VNFM to update the configuration of the load balancer, in order to stop traffic redirection to a specific VNF (running a AMF), which will be later deleted by the VIM.

3.4.2.2 Proposed AMF load balancing and scalability process

A. Model description

In this section, we describe our proposal featuring AMF load balancing and scaling. This model is triggered in order to split the load over the available AMFs in the NGC. When all AMFs are fully used, our proposed algorithm takes control of deploying new one(s) to keep the access to the NGC and balance the load over all the operational AMFs, in order to reduce latency. This model also works in a reverse way. When many AMFs are deployed and the overall system is underutilized, it will trigger a scale in procedure, in order to avoid wasting resources. Fig. 9 depicts the model of our AMF load balancing and scaling proposal. To start, for the sake of simplicity, we assume one (R)AN on the access side and multiple AMFs in the NGC. However, this model can be easily extended to consider a multiple (R)AN case.

The different parameters of the model are described below:

1) λ: The number of arriving UEs request
2) θ: The number of UE request that have not been satisfied and that need to be re-sent by the UEs
3) w: The number of UE request in the buffer waiting to be dispatched over the deployed AMFs

4) q_i: The number of UE request that are sent to AMF$_i$ for processing

5) m_i: The number of UE request being processed by AMF$_i$.

6) μ_i: The number of UE request satisfied by AMF$_i$.

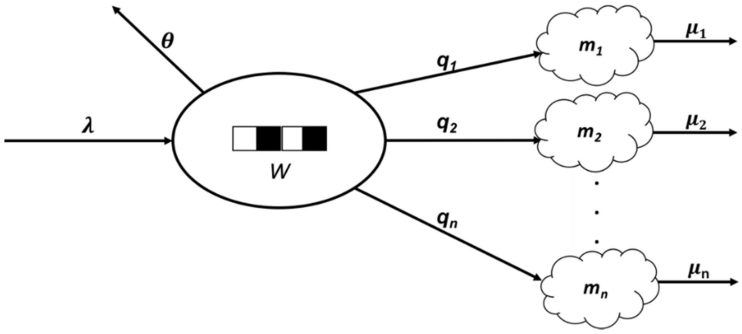


Fig. 9 – System model

Following the generic system parameter description, the state evolution of our model is written below following the discrete-time system of equations:

$$\begin{cases} \mathrm{m_1(k+1) = m_1(k) + q_1(k) - \mu_1(k)} \\ \mathrm{m_2(k+1) = m_2(k) + q_2(k) - \mu_2(k)} \\ \mathrm{..} \\ \mathrm{m_n(k+1) = m_n(k) + q_n(k) - \mu_n(k)} \\ \mathrm{w(k+1) = w(k) + \lambda(k) - \theta(k) - \sum_{i=1}^{n} q_i(k)} \end{cases} \quad \text{(A1)}$$

Let $p_i(k)=qi(k)- \mu_i(k)$ and $\gamma(k)= \lambda(k) - \theta(k) - \sum_{i=1}^{n} \mu_i(k)$. The system represented in (A1) can be reformulated as below:

$$\begin{cases} \mathrm{m_1(k+1) = m_1(k) + p_1(k)} \\ \mathrm{m_2(k+1) = m_2(k) + p_2(k)} \\ \mathrm{..} \\ \mathrm{m_n(k+1) = m_n(k) + p_n(k)} \\ \mathrm{w(k+1) = w(k) + \gamma(k) - \sum_{i=1}^{n} p_i(k)} \end{cases} \quad \text{(A2)}$$

The model (A2) can be written as a discrete-time linear system, in the form:

$$\begin{cases} \mathrm{X(k+1) = AX(k) = BU(k)} \\ \mathrm{Y(k) = CX(k)} \end{cases} \quad \text{(A3)}$$

where, the state vector $X(k) = \left[M_n(k) - M_n^{ref}, w(k) - w^{ref}\right]^T$

and

$M_n(k) = [m_1(k), m_2(k), \dots, m_n(k)]$

The constant vector $M_n(k) = [m_1(k), m_2(k), \dots, m_n(k)]$ represents the targeted load of the AMFs. This will avoid AMF overhead and guarantee that requests are satisfied with a limited latency. w^{ref} is the targeted overload allowing detection if an AMF scale in/out is needed for the system to guarantee an optimal requests' dispatching and processing while minimizing resources' wastage.

The control vector *U(k)* is defined as follows:

$$U(k) = [p_1(k), p_2(k), \dots, p_n(k), \gamma(k)]^T$$

The remaining matrices are, thus, defined as follows:

$$A = C = I_{n+1}$$

$$B_{(n+1)\times(n+1)} = \begin{pmatrix} I_n & 0_{n\times 1} \\ -1_{1\times n} & 1_{1\times 1} \end{pmatrix}$$

The output *Y(k)* of this system represents the load of each AMF and the state of the buffer at time step k.

It can be checked easily that all eigenvalues of Matrix *A* do not satisfy the stability condition[1] [36]. This means that the system described in (A3) is unstable and does not converge to the desired state, if no control action is performed.

The controllability [36] of this model can be analyzed by calculating the controllability matrix, which is defined as follows:

$$C = [B\ AB\ A^2B\ \dots\ A^{n-1}B]$$

To be controllable, the controllability matrix of the system should have a full row rank. Indeed, for our model, the controllability matrix has a full row rank equal to *n+1*. It can, also, be checked that the system is observable [36].

B. Dynamic AMF Load Balancing

Our focus in this part is on the design of the regulator's model to stabilize the whole system by scheduling the UE requests to a given AMF, with the objective to efficiently use the available resources. Also, it will control the decision of AMF scale out in the case where more resources are needed, or a scale in when fewer resources are used to reduce resources' wastage. It is worth recalling that the regulator's model is a function that runs at the SO.

Since the controller, following the requirements listed above, needs to dynamically and in real-time calculate m_i, θ and *w*, it will be based on the LQR [37]. More specifically on infinite-horizon, discrete-time LQR. In LQR model, there are two main characteristics: the performance index *J* and the feedback vector *U(k)*.

The performance index is given as follows:

$$\mathrm{J} = \sum_{\mathrm{k}=0}^{\infty}[\mathrm{X(k)}^{\mathrm{T}}\mathrm{QX(k)} + \mathrm{U(k)}^{\mathrm{T}}\mathrm{RU(k)}] \quad \text{(A4)}$$

where *X(k)* and *U(k)* are the state vector and the feedback (control) vector, respectively.

The LQR aims to minimize the performance index in order to allow the system to converge to the goal with less controller action. Therefore, *Q* and *R* from (A4), as well as the cost matrices, should satisfy:

$$\mathrm{Q} = \mathrm{Q}^{\mathrm{T}} \geq 0, \mathrm{R} = \mathrm{R}^{\mathrm{T}} > 0 \quad \text{(A5)}$$

Finally the feedback vector *U(k)* can be written as the following:

$$\mathrm{U(k)} = -\mathrm{KX(k)} \quad \text{(A6)}$$

[1] All eigenvalues should be strictly smaller than 1

where the matrix $K = [R + B^T S(k+1)B]^{-1} B^T S(k+1)A$ and the matrix S(k) is the solution of the following Riccati difference equation [38]:

$$\mathrm{S(k)} = \mathrm{Q} + \mathrm{A^T S(k+1)A} - \mathrm{A^T S(k+1)B[R + B^T S(k+1)B]^{-1} B^T S(k+1)A} \quad \text{(A7)}$$

In steady state, *S(k)=S(k+1) = S*, thus, Riccati equation expressed in (A7) can be written as:

$$\mathrm{S} = \mathrm{Q} + \mathrm{A^T SA} - \mathrm{A^T SB(R + B^T SB)^{-1} B^T SA} \quad \text{(A8)}$$

The optimal control can thus be described by:

$$\mathrm{U(k)} = -\mathrm{(R + B^T SB)^{-1} B^T SAx(k)} \quad \text{(A9)}$$

C. Dynamic AMF Scaling

Having detailed the AMF load balancing algorithm, we focus now on the AMF scaling process.

Thanks to the control vector *U(k)*, the buffer *w(k)* is calculated and updated in real-time. In addition to those two parameters, new parameters, inspired from [38], will be calculated. Starting by the Average Loss *aLoss* that is determined as follows:

$$\mathrm{aLoss(k+1)} = \mathrm{wf} \times \mathrm{aLoss(k)} + (1 - \mathrm{wf}) \frac{\theta(\mathrm{k})}{\lambda(\mathrm{k})} \quad \text{(A10)}$$

where *wf* is a weight factor. Once we have the *aLoss*, a Congestion probability is deduced as follows:

$$\mathrm{CongP} = \mathrm{aLoss(k)} + \beta \times \sqrt{\mathrm{aLoss(k)} \times \gamma(\mathrm{k})} - \mathrm{w^{ref}} \quad \text{(A11)}$$

where β is a learning factor.

If *CongP* is higher than a fixed probability threshold, then a scale out is needed. Thus, a new AMF is deployed and integrated to the whole system. Otherwise, a scale in possibility is studied. For that, we need to check if the average load of the actual system can be supported if we remove one AMF. If it is the case a scale in the procedure is triggered.

3.4.3. Results

3.4.3.1 Test scenarios

As stated before, the first objective of our model is to scale out or in the AMF dynamically. The second one consists on dispatching the UE requests intelligently in order to have an optimal load on each AMF and, thus decreasing the response latency. To test and validate our model, four scenarios were implemented using Matlab, in addition to the probabilistic EWMA model, which dispatches the UE requests randomly over the active AMFs. The envisioned scenarios will allow us to evaluate the performance of our model, which is based on control theory, against the EWMA model.

For both, the control model and the EWMA model, we fixed the highest number of AMFs (5 AMFs) that can be deployed during the scenarios. Each AMF supports a maximum load up to 30 processes by time step. Above this highest rate, the UE arrivals will be blocked. We also fixed the optimum AMF load rate to 20 requests by time step. For this optimum rate, the AMF can process the requests with no latency. Above this rate, an additional latency, which may take very high values, is added to each request.

The first scenario implements different arrival rates, based on Poisson distribution, over a given period. The arrival rate is then increased over the time until reaching the maximum capability of the system, and then later decreased until reaching a minimal arrival rate. This scenario allows validating the dynamic adaptation of our model and its flexibility by scaling out the AMF NF when the arrival rate increases and scaling in when the arrival rate decreases. This scenario also will prove the stability of our model and will show the ability of our model to keep the AMF load around the optimal processing value.

The three other scenarios represent an underutilized system, a fully-loaded system and an overloaded system. Each scenario is repeated 30 times in order to compute confidence intervals. Those scenarios will allow us to see the behavior of our model following different load patterns and compare it to the behavior of the EWMA model.

3.4.3.2 Results analysis

The results of the first scenario are depicted in Fig. 10. Following the arrival dispatching plots, we notice that the EWMA model schedules the requests randomly over the five AMFs (q_1 to q_5) independently from the arrival load (λ), unlike the control model. From the AMF load level plots, we deduce that when the arrival rate increases, the EWMA model pushes the AMFs loads (X_1 to X_5) to their limits (30 process at a time), hence adding latency to each process. However, the control model dispatches the arrivals in a manner to have an optimal load (20 processes at a time) in each active AMF and thereby processing the requests with no additional latency. From the Number of Active AMF plot, we notice that the EWMA is not stable and falling in what is called the Zeno phenomenon ("bouncing ball"). The control model shows more stability and scales in/out the AMF instances accurately depending on the need.

The results of scenario 2 to 4 are depicted in Fig. 11. Thanks to those scenarios, we are able to compare the behavior of the control model, and the EWMA model in three traffic patterns: underutilized system; fully-loaded system and overloaded system.

In the underutilized system plot, we notice that the EWMA model is using three AMFs as the control model. However, the EWMA load (X) is not balanced between the three AMFs, while the control model has an equivalent load over the three AMFs.

In the fully-loaded scenario, we notice that the EWMA model scheduled the arrivals randomly over the five AMFs with an average of 28 processes at a time. Thus, the EWMA model is loading the AMFs more than the optimal AMF load, and thereby adding latency while satisfying the arrivals. In contrast with the latter scenario, the control model transfers the arrivals to the five AMFs depending on their load in order to maintain the optimum AMF load as much as possible and so satisfying the requests with no additional latency. Despite the following scheduling solution, it is important to notice that the average number of blocked arrivals, in this category of tests, for the control model is lower than the EWMA model. Therefore, the control model is able to distribute load among the AMFs with neither additional latency nor blocked arrivals.

Finally, in case of overloaded conditions, unlike the EWMA model, the control model avoids fully loading the AMF, and tries to keep its load around the optimal AMF load. Thus, some additional arrival will be blocked as shown on the average of the blocked arrival's plot.

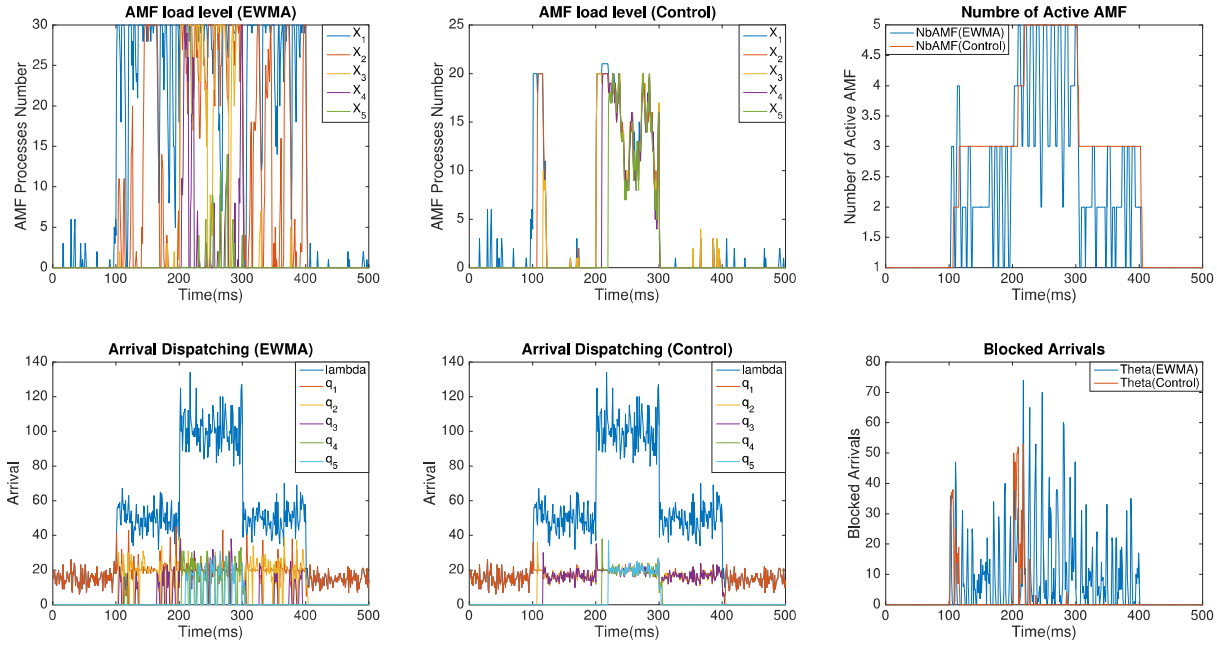


Fig. 10 – Evaluation of the adaptation of the control model

To summarize, the control model can schedule the arrivals depending on the AMFs load in order to maintain an optimal AMF load and to satisfy the request arrivals with no additional latency, especially that decreasing latency is an important requirement for 5G. Furthermore, the control model is designed to avoid resources' wastage by only activating (deploying) the exact number of needed AMF. Indeed, it can scale out the system in order to satisfy an arrival rate increase, if needed, while scale in when arrival rate decreases.

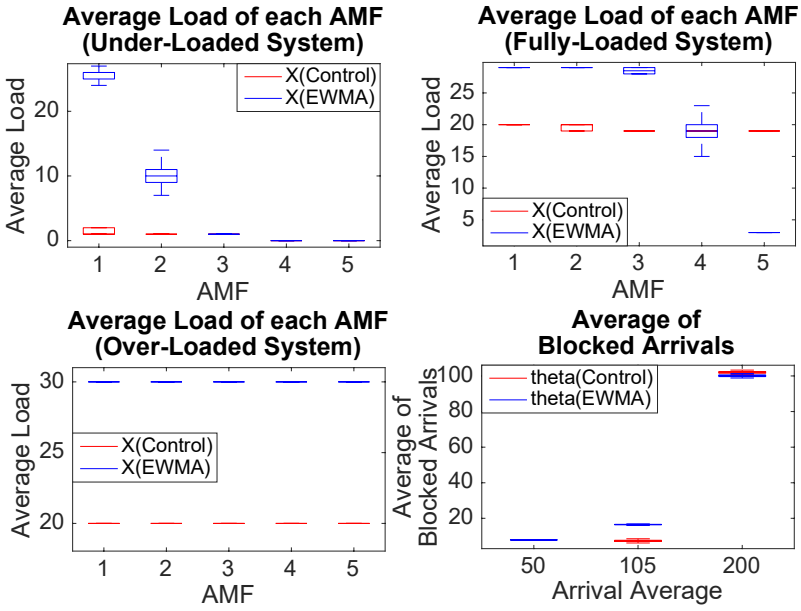


Fig. 11 – Evaluation of control model behavior vs. different arrival load

Unfortunately, reducing latency and scaling dynamically add some penalties on the system. As mentioned above, some sessions will be blocked when the system is overloaded or when scaling up. However, the system can remedy from this penalty by adding a prediction function allowing deployment of a new AMF in advance when arrival rate is predicted to increase.

3.5. A pragmatic approach to online optimization of vEPC nodes placement

5G network (or a network slice) is expected to be a collection of VNFs that will be placed in a distributed, virtualized environment. MANO compliant orchestrator drives the process of initial VNF placement and dynamic allocation of resources to Network Services during their runtime. Due to changing traffic patterns caused by changed users demands or their mobility, the initial VNF placement cannot be far from being suboptimal and new VNF configuration can improve the network performance and reduce the overall traffic. It is therefore essential to monitor system after the initial placement of VNFs based on the traffic patterns, and if needed, make a VNF placement reconfiguration.

The initial placement of VNFs follows the initial network dimensioning based on typical input parameters (cf. the chapter 3.5.2) and uses typically an off-line algorithm that starts from scratch. In general, the VNF placement is a multi-objective optimization problem that is np-complete and many semi-optimal but relatively fast, heuristics based approaches exist. The algorithm analyzes the Network Service properties and available resources of the distributed infrastructure. The placement of VNFs is dependent on the resource layer topology and capabilities, traffic patterns and additional constraints like energy efficient placement, etc. Typically the variable that is changed over time is the traffic pattern. However, if the operator will change some of its policies, it may also require a reconfiguration of the VNF placement. After the placement and putting the network service into operations the resource allocated to VNFs or connections between them can be scaled by MANO according to traffic demands.

The on-line VNF placement optimization lies in making changes in VNFs placement to optimize the network service (or slice) behavior. In opposite to the static VNF placement, the algorithmic delay of VNF cannot be neglected in such case. It has to be noted that during the implementation of topology changes (moving some virtual nodes, adding or removing nodes) the network should work in a non-interrupted way.

In this chapter, a pragmatic approach to the implementation of vEPC of LTE in a distributed cloud environment will be discussed. The analysis is focused on the 4G technology because of the availability of operational traffic information, what is not yet the case of the 5G network. The presented methodology may, however, be applied to 5G networks as well. In opposite to many existing approaches, we do not solve the problem of generic VNF placement, but instead, we focus on specific vEPC and telco infrastructure properties. Due to such approach, the VNF placement problem is much easier to solve. Moreover, the presented approach is based on operators' practice – the solution it provides is "safer" from the operator's point of view.

The approach takes into account the following constraints concerning telco infrastructure and EPC:

- The infrastructure of the operator is composed of multiple data centers, which play different roles. One of them is a central cloud; there are also regional clouds and the local clouds. Such approach in a realistic way reflects the cloud topology of carrier-grade networks, as it has already been mentioned in [39].
- The vEPC is one of many Network Services in the virtualized environment, but at the same time, it is a privileged service, whose proper functioning is under special operator control. Regarding resource allocation, it means that this service has the top priority. Moreover, the lifetime of such service is long (like in the MNO or MVNO case).
- Relocation of existing vEPC nodes between DCs can be perceived as problematic from the service continuity point of view. The EPC network has mechanisms of MME load balancing and re-balancing which should be exploited during vEPC optimization for throttling down the load of the MME instance in initial location and moving the load to the instance in the desired location. The target instance should be put in service before off-loading the UEs. Similarly, in case of vSGW or vPGW instances, the analogous procedure for user plane optimization may exploit the typical procedures used for HO with SGW relocation and indirect tunnelling.
- Adding or deleting a virtualized network node (function) instance is the quite disruptive procedure for network configuration. Dynamic continuous network optimization should rather exploit procedures of traffic management instead of changing the network topology.
- The topology of the EPC network and roles of their nodes impacts to a certain extent their placement, for example, there should be no on-line HSS (re)placement.
- In EPC there is a pretty strict separation between the control plane nodes and the user plane ones. There are different optimization goals of both planes, and they impact the placement of different VNFs. Therefore separate algorithms can be used for control plane and user plane optimization.

Due to the mentioned properties, the VNF problem placement can be solved faster by using relatively simple algorithms in comparison to the generic case. Moreover, the control plane and the user plane optimization can be performed independently. The presented solution is probably far from being optimal, but due to its conservative character, it can be easily adopted by network operators as a low-risk approach.

3.5.1. Telco cloud infrastructure

The software-based mobile networks will be deployed using the telco operator infrastructure. In the softwarized world, such infrastructure will be a set of interconnected DCs of different roles. It can be assumed that the multi-DC architecture will be composed of the following types of entities [39]:

- Central Data Centre (CDC). This DC has significant computing and storage resources and is the most reliable DC of the operator's infrastructure. It may be provisioned in HA 1+1 mode but will usually be seen as a single logical instance of top priority. It may be expected that its location will be chosen at or near to the "network gravity center" in terms of delay.

- Regional Data Centers (RDCs). These DCs are distributed regionally, and their reliability is lower than the reliability of CDC. Moreover, they typically have fewer resources than CDC.
- Local Data Centers (LDCs). These DCs can be numerous and their reliability is much lower than RDCs, and their resources are also limited. They may be located close to customer premises or antennas of the mobile network.

The hierarchical telco cloud infrastructure is presented in Fig. 12. Please note interconnections between DCs of the same level. These connections improve the overall reliability of the system and increase its service-related capability.

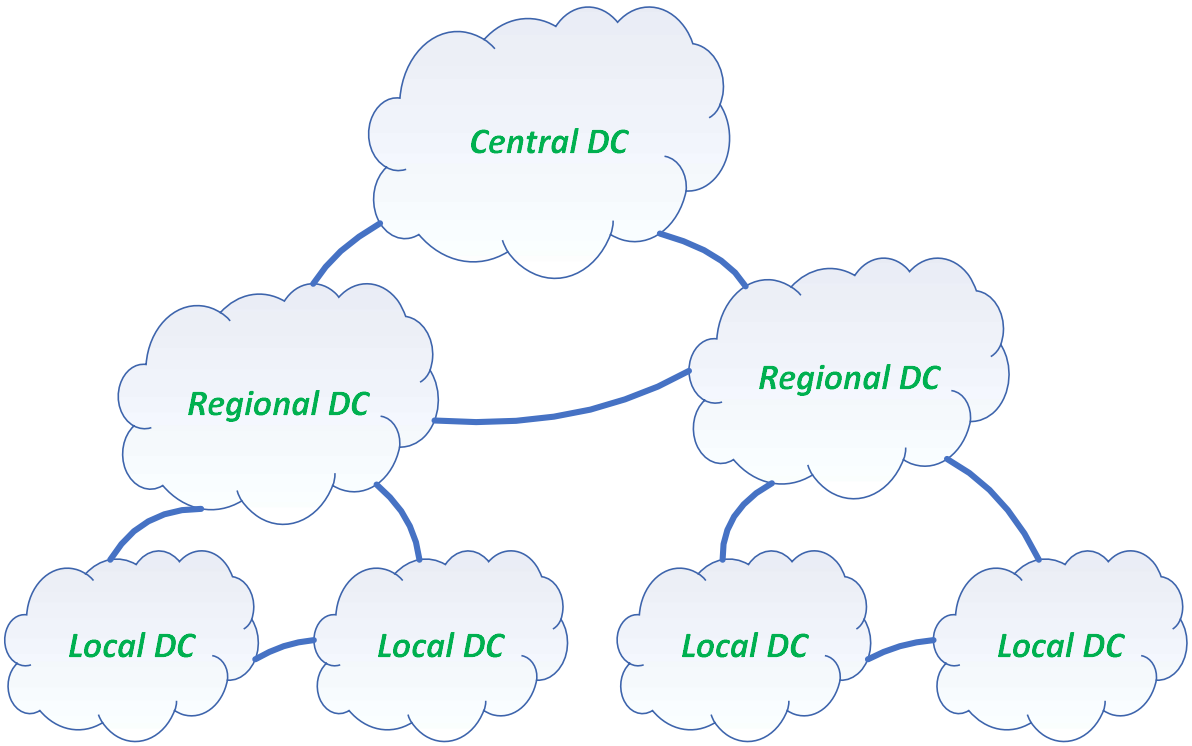


Fig. 12 – A typical structure of telco clouds

The abovementioned DCs are geographically distributed, and their locations typically correspond to the density of population or installed services. There are also other factors such as availability of power, the topology of available physical connectivity infrastructure (optical fibers, copper cables), ownership/leasing of buildings or rooms, capabilities of future extensions of DC rooms, and legal requirements, etc. that can impact the localization of DCs in the operator's network. It is expected that operator's engineering rules set allowed DCs levels for placement of specific network functions (VNFs).

A placement of cooperating VNFs in distant DCs introduces a round-trip-time delay of 12 μs/km. So, the processing of delay sensitive applications should keep their VNFs close to each other. However, the propagation delay between DCs installed in the area of even a large metropolis (radii of several kilometres), should be in the range of 100 μs that is acceptable for most mobile network operations. According to [40], the multi-layer architecture of VNF software (due to VNF functional decomposition into front-end part, stateless functional logic and database part) introduces significant additional delays caused by the REST communication between software layers.

3.5.2. The EPC network dimensioning

The classical approach to EPC dimensioning is based on expected maximum load both in user and control planes. The factors presented in Table 3 are often taken into account for EPC nodes dimensioning:

Table 4 – Criteria of EPC nodes dimensioning

Node	Factors
HSS	number of provisioned subscribers
MME	number of simultaneously attached users intensity of idle/active mobility states transitions signaling transactions intensity that includes the traffic related to following procedures: attach/detach, PDN connection, bearer activation/deactivation, TAU (intra-/inter-MME), X2 and S1 handovers and inter-MME handovers
PCRF	intensity of PCRF transactions
SGW/PGW	peak number of active bearers peak aggregated UEs download (usually dominating) throughput of

As it is evident from the table the most complex traffic concerns MME, an analysis of the MME traffic split can be found in [25]. According to it, the traffic volume split of MME is following:

- session management (61.9%),
- paging (28,7%),
- TAU (4.9%).

In total, these three procedures are responsible for 95.5% of the MME traffic. They should be therefore taken into account in the context of the dynamic placement of MME.

The MME traffic is exchanged with the following nodes/functions:

- UEs/eNodeBs: X2/S1 handovers, TAU, attach/detach, paging, session management,
- SGW: session management,
- HSS: attach/detach (authentication), TAU.

A quick analysis and estimations show that the largest fraction of the MME traffic is exchanged between eNodeBs and twice smaller are the traffic exchanged with SGW. Unfortunately, there are no estimations of the MME-MME traffic. Such traffic is with no doubts dependent on the time of day and mobility of users. In the context of placement, it is desirable to keep MMEs close to eNodeBs and SGWs.

The user plane traffic uses GTP tunnels between SGW and PGW, and the PGW placement is critical in the context of user traffic distribution.

3.5.3. EPC properties in the context of the vEPC implementation

A generic topology of an LTE/LTE-A network with multiple MMEs and PGWs is presented in Fig. 13. The EPC has some specific properties that have to be taken into account for placement of vEPC nodes.

First of all, virtualized RAN nodes are typically decomposed into RRHs that are connected to BBUs via CPRI. Antennas are connected to RRH with short feeders impacting RRH distribution. The connection between the RRH and BBU can be relatively long (within the span of 15 km), but in the existing networks, it is typically kept within the limit of 100-500 m, it generates and receives the most of the network traffic. The RRHs are hardware nodes (PNFs in ETSI terminology), and their location is fixed.

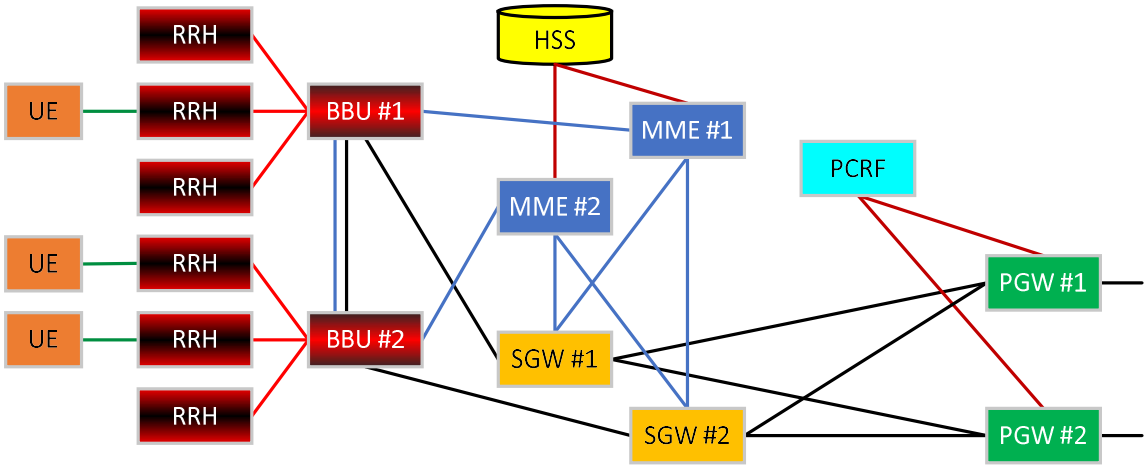


Fig. 13 – Generic topology of complex LTE/LTE-A network

The mentioned features of the mobile network impact the placement of VNFs of RAN and CN. There are, however, more features or constraints of the mobile network that has to be taken into account. For example, in case of LTE/LTE-A:

- **BBU** – The BBU should be placed closely to RRHs it controls, the delay between the units and the amount of the CPRI traffic should be minimized.
- **HSS** – In a system, there is a single logical entity of HSS, and it should be placed in the most reliable DC (i.e. CDC) that is also in the "center of gravity of the network".
- **PGW** – The PGW nodes should be placed close to the interconnection points with other IP networks, most probably in central and regional DCs; there may be, however, an exception to the general rule, which is the case of a local PGW serving a private VPN traffic, and the PGW serves as an SGi gateway for the APN associated with this VPN (e.g. several eNodeBs covering the area of the factory or a power plant, and the PDN is the internal network of these facilities).
- **MME/SGW** – The number of SGWs and MMEs, as well as their placement, can be chosen without specific constraints. However, MME (if there are multiple instances) should be placed closer to the edge if there are many active sessions and TAU procedures from a specific area. Both nodes are candidates for the dynamic placement.

There are some mechanisms of EPC that nicely allow for changing of the control plane or user plane topology, by virtual addition or removal of nodes. The generalized topology of EPC control plane is shown in Fig. 14 In the context of dynamic placement of VNFs it is important to emphasize that:

- The communication between MMEs and HSS is realized by a message bus with internal load balancing mechanisms. The mechanism allows the MME instances of a specific network function (e.g. MME) may be in the active or frozen mode. The frozen instances are pre-

configured and ready to be woken up, if necessary. Global network management mechanisms can provide broadcast information about woken-up instances joining the network or frozen instances quitting the network.

- The MMEs have also load balancing on the terminal side (UE) that allows for attachment of new UEs to less-loaded MMEs.

Both mentioned mechanisms allow for the policy control of each MME to be in a "frozen" or an active state without the need of intensive EPC reconfiguration – the existing mechanisms allow for graceful switching on or off MMEs according to the operator needs.

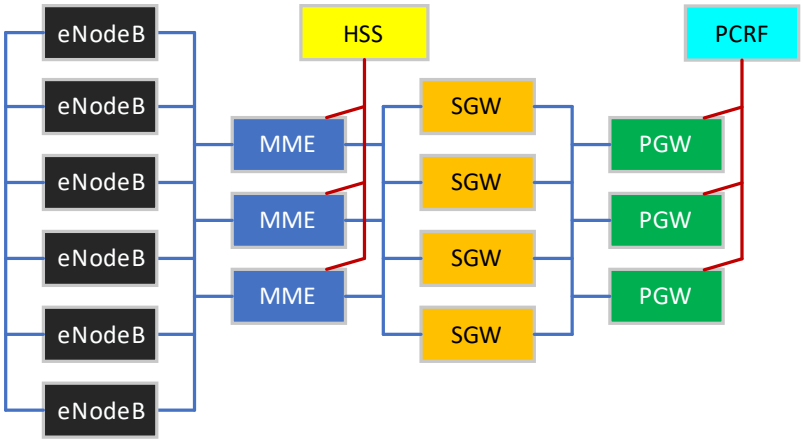


Fig. 14 – Control plane topology of LTE/EPC

The user plane topology is shown in Fig. 15. In this case, as it has been already mentioned, the SGW function is a potential candidate for dynamic placement. It has to be noted that the SGW is not fixedly allocated to eNodeBs or PGWs and therefore such nodes can be easily added (woken up) or removed (put to sleep).

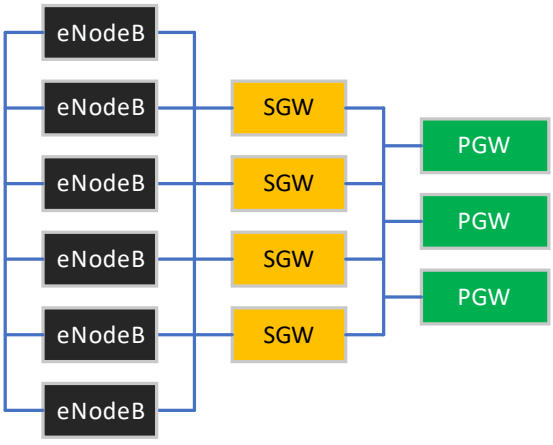


Fig. 15 – User plane topology of LTE/EPC

3.5.4. The proposed approach

In this chapter, we will describe the proposed mechanism for on-line optimization of the placement of vEPC nodes. We will address the data and control plane optimization separately, having in mind such factors as the overall traffic volume and control plane or user plane delays.

The starting point for the optimization is a centralized placement of nodes according to their roles and cloud type. Later on, if it is necessary, new nodes can be added at the appropriate local cloud. We do not plan to remove any of the initially placed nodes or to move them from initial locations to other clouds, but of course, all resources allocated to nodes will be scaled automatically by the orchestrator. We expect that in a real life some nodes will be frozen at the EPC level (i.e. by proper configuration they will reject requests) or by the use of MANO resource pool mechanism. Such approach from the technical point of view simplifies operations and minimizes runtime reconfigurations.

3.5.4.1 Initial placement of vEPC nodes

The proposed concept is focused on 4G/5G-R15 mobile networks. It takes into account the three following factors:

- the specifics of the telco DC infrastructure and topology (hierarchy of DCs).
- the specifics of EPC/LTE networks functions and their dependencies (traffic patterns).
- the fact that EPC data and control planes optimization can be addressed separately.

In the proposed concept, we follow the discussion about constraints of placement of VNFs that compose the LTE network. The initial, time-invariant VNF placement is based on long-term averages. The mechanism of VNF scalability can be nicely used for dynamic allocation of resources to VNFs. As it has been discussed, there are three network nodes that placement can change the performance of the network, namely MME, SGW and PGW. In a virtualized environment such nodes may migrate and new ones can be added to the network to optimize network performance like user plane or control plane delay and the overall traffic paths.

In general, it can be assumed that in regards to the placement of 4G/5G-R15 virtual network functions (VNFs) the assumptions described in Table 5 can be taken into account.

Table 5 – Probability of placement of specific EPC nodes within the hierarchy of DCs

DC type	Probability of placement: High (>0.6)	Probability of placement: Medium (0.2-0.6)	Probability of placement: Low (<0.2)
CDC	PCRF, HSS, MME, PGW	SGW	RRH, BBU
RDC	MME, SGW, PGW	PCRF, HSS	BBU
LDC	RRH, BBU	MME, SGW	PCRF, HSS

As it can be seen in the table above, the placement of HSS, PCRF and RRHs can be taken a priori. A similar situation is with BBUs that use CPRI interface, so they should be placed as close to their RRHs as possible. For the initial placement, it is assumed that in CDC there are placed nodes for HSS, PCRF, PGW and MME and these nodes are excluded from further optimization, their location in CDC is suited for long-term traffic patterns. In LDC there are placed BBUs that have an interface to their RRH, and these nodes' locations are also fixed. Initially, an MME and SGW are deployed in regional clouds. Fig. 16 shows the example initial configuration of a mobile network. The functions in black are fixed (not to be relocated). The blue functions (instances of MME and SGW) located in various DCs can be translocated or created in other locations, i.e. LDCs.

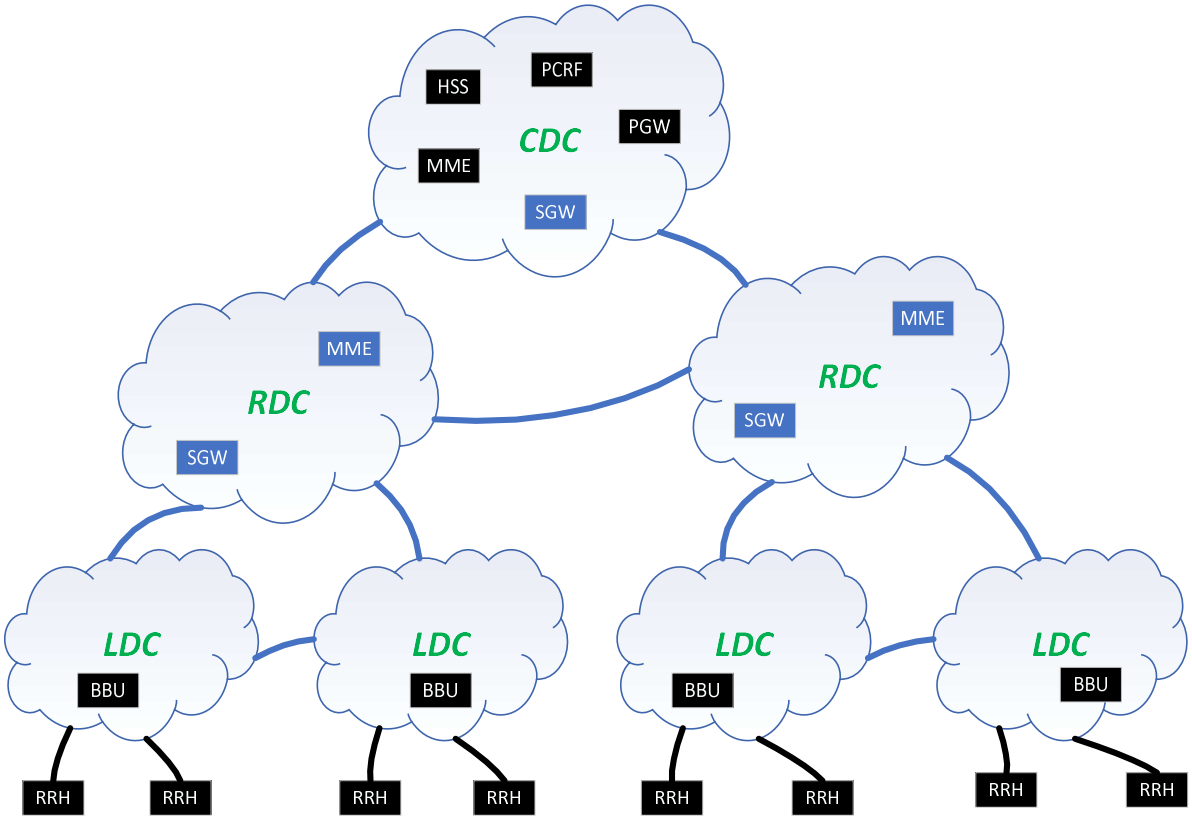


Fig. 16 – Initial placement of vEPC nodes

3.5.4.2 On-line placement optimization of vEPC nodes

Dynamic changes in traffic patterns drive the modification of nodes placement. These changes can be caused by increased number of users, increased mobility of users, changing traffic demands. All the factors may impact the user plane or control plane traffic. As long as these changes are geographically distributed there will be no impact on the placement of vEPC nodes. If the traffic is localized, it may impact the creation of additional nodes that are to be implemented as close as possible to the aggregated traffic source. In most cases, they could be placed in LDC, but in some cases, RDC is also a good candidate for such placement.

Algorithm for online placement of control plane VNFs

In regards to the control plane, the MME is the node whose instantiation in different places is considered. Based on some reports and internal Orange measurements, the model of MME traffic dependencies described in Table 6 () has been assumed.

The control plane traffic related to SGW is the GTP-C traffic and is related to bearers and session management. Therefore, it depends on the dynamicity of sessions, BBU related traffic depends on the number of users and their mobility. The two fractions of the traffic as dominant are taken into account in the context of MME placement. However, the amount of the control traffic towards SGW may trigger deployment of another SGW instance, for example in LDC. The following parameters are taken into account for the deployment or removal of MME in LDC:

- BBUs related traffic volume of an LDC;
- SGW related control traffic volume linked to specific LDC.

Table 6 – Characteristics of CP traffic at interfaces of MME

MME interacting node	Estimated traffic fraction	Purpose	Comments
BBU	35%	network attach/detach, TAU, HO (S1 or X2), MME balancing/re-balancing, paging, mobility states support	S1-MME (S1-AP)
SGW	50%	session creation/termination, bearer control	S11 (GTP-C)
HSS	10%	UE network attach/detach, user profile checking during service request, update UE location	S6a (Diameter)
MME (-x)	5%	inter-MME connectivity for HO and re-balancing support	S10 (GTP-C)

The algorithm for dynamic MME placement is following:

Control_Plane_dynamic_rearrangement

Variable	Explanation
rdc_i	i^{th} RDC
$R = \{rdc_1, rdc_2, \ldots, rdc_n\}$	Set of RDCs
ldc_{ji}	j^{th} LDC subordinated to i^{th} RDC
$L_i = \{ldc_{1i}, ldc_{2i}, \ldots, ldc_{mi}\}$	Set of LDCs subordinated to i^{th} RDC
$\Theta_{S1\text{-}MME}(ldc_{ji})$	Normalized S1-MME traffic originating from ldc_{ji}
$Th_{S1\text{-}MME,i}$	Individual normalized S1-MME traffic difference threshold for i^{th} RDC
$Me(x)$	Median of x
$UE(ldc_{ji})$	set of UEs served by ldc_{ji}

for $\forall\ rdc_i \in R$
  for $\forall\ ldc_{ji} \in L_i$
    if $(\Theta_{S1\text{-}MME}(ldc_{ji}) - Me(\Theta_{S1\text{-}MME}(L_i)) > Th_{S1\text{-}MME,i})$
      Create_MME(ldc_{ji})
      Set_MME_pool(ldc_{ji}, rdc_i)
      Offload$(UE(ldc_{ji}), rdc_i, ldc_{ji})$
    end if
    if $(Me(\Theta_{S1\text{-}MME}(L_i) - \Theta_{S1\text{-}MME}(ldc_{ji})) > Th_{S1\text{-}MME,i})$
      Offload$(UE(ldc_{ji}), ldc_{ji}, rdc_i)$
      Unset_MME_pool(ldc_{ji}, rdc_i)
      Terminate_MME(ldc_{ji})
    end if
  end for
end for
# Hysteresis is $2 \times Th_{S1\text{-}MME,i}$
# Period of repetition to be defined
# Grace period after each transition at ldc_{ji} may be defined

Algorithm for on-line user plane VNFs placement

As it has been already discussed the placement of the BBUs is fixed, and the dynamic placement of SGW and PGW is only possible. The PGW is used as an external gateway to other networks. Therefore a degree of its placement freedom is rather limited. However, due to the use of the GTP protocol, if the traffic of specific LDC or RDC has a significant fraction of the outgoing traffic to BBUs served by the same DC, the addition of SGW/PGW tandem nodes will reduce data

transmission delays and the overall traffic volume in the network. In the concept, the following model of SGW traffic dependencies has been assumed, based on real traffic observations.

Table 7 – Estimation of SGW user plane traffic distribution

SGW interacting nodes	Estimated fraction of the traffic	Purpose	Comments
eNodeB (BBU)	50%-60%	Handling UE data traffic	S1-U (GTP-U)
PGW	40%-50%	BBU partly ends in SGW, but the rest should go to connected with SGW, multiple PGWs	S5/S8 (GTP-U)

As it can be seen in the table above, the traffic of between SGW and BBU, and between SGW and BBU is almost the same. The algorithm for dynamic SGW/PGW placement is following:

User_Plane_dynamic_rearrangement

Variable	Explanation
rdc_i	i^{th} RDC
$R = \{rdc_1, rdc_2, \ldots, rdc_n\}$	Set of RDCs
ldc_{ji}	j^{th} LDC subordinated to i^{th} RDC
$L_i = \{ldc_{1i}, ldc_{2i}, \ldots, ldc_{mi}\}$	Set of LDCs subordinated to i^{th} RDC
$\Theta_{S1\text{-}U}(ldc_{ji})$	Normalized S1-U traffic originating from ldc_{ji}
$Th_{S1\text{-}U,i}$	Individual normalized S1-U traffic difference threshold for i^{th} RDC
$Me(x)$	Median of x
$UE(ldc_{ji})$	set of UEs served by ldc_{ji}

for $\forall\ rdc_i \in R$
  for $\forall\ ldc_{ji} \in L_i$
    if $(\Theta_{S1\text{-}U}(ldc_{ji}) - Me(\Theta_{S1\text{-}U}(L_i)) > Th_{S1\text{-}U,i})$
      $Create\_SGW(ldc_{ji})$
      $HO\_SGW(UE(ldc_{ji}), rdc_i, ldc_{ji})$
    end if
    if $(Me(\Theta_{S1\text{-}U}(L_i) - \Theta_{S1\text{-}U}(ldc_{ji})) > Th_{S1\text{-}U,i})$
      $HO\_SGW(UE(ldc_{ji}), ldc_{ji}, rdc_i)$
      $Terminate\_SGW(ldc_{ji})$
    end if
  end for
end for
# Hysteresis is $2\times Th_{S1\text{-}U,i}$
# Period of repetition to be defined
# Grace period after each transition at ldc_{ji} may be defined

Fig. 17 shows an example final configuration of a mobile network after a dynamic reconfiguration. The functions in red are replicated (scaled) MME or SGW functions which were added (woken-up) in LDCs due to UE mobility trends and significant growth of UE traffic density in specific areas. Additionally, the new PGW was temporarily added to an LDC node due to specific circumstances (LDC serving an area covering a stadium, where the sports event will take place, and a special locally terminated PDN for supporting a TV transmission is provisioned). Scenario A shows the case for replication of SGW within a specific LDC, scenario B shows the case where replication of MME is followed by replication of SGW, and scenario C is the case where the local EPC is fully

provisioned within a specific LDC, e.g. for mobile VPN within a power plant/factory or a local traffic offloading for an occasional event.

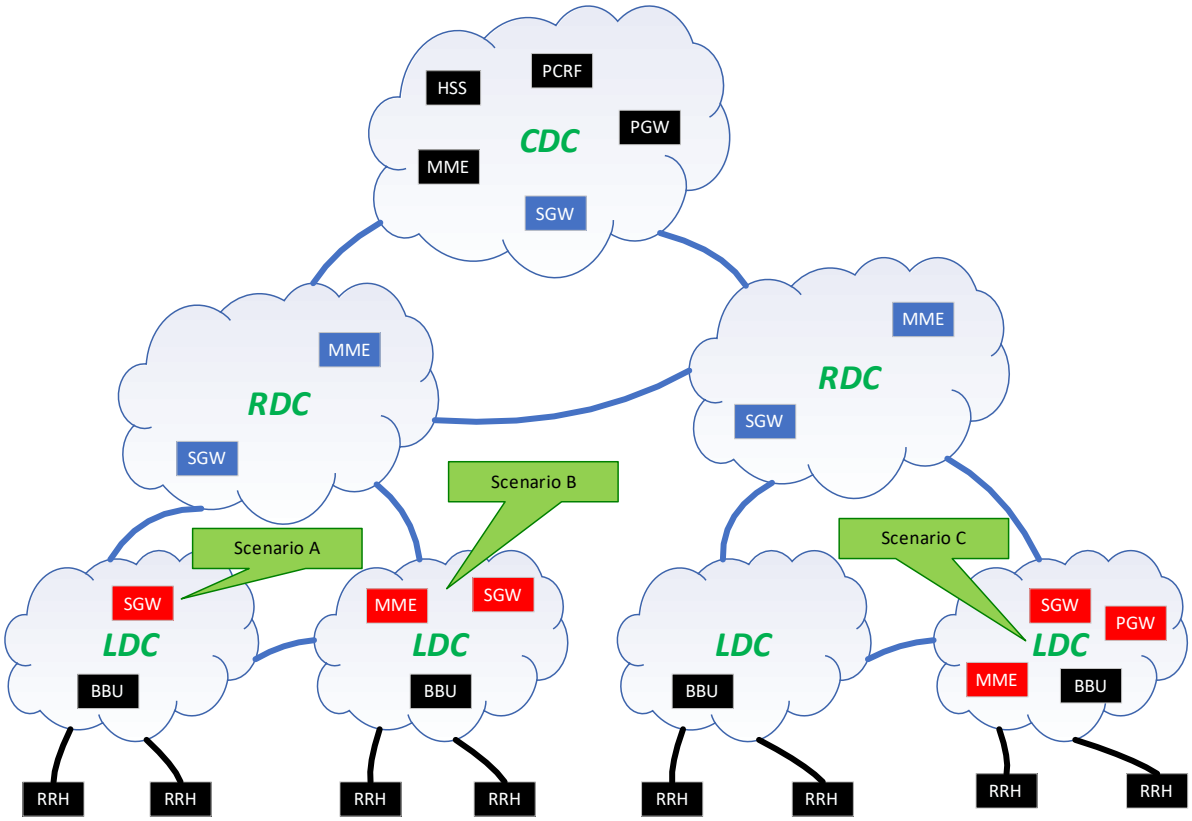


Fig. 17 – Final network configuration

As it has been mentioned, the presented approach is simple and fast but takes some assumptions which make it a sub-optimal solution.

3.6. Slice performance monitoring and elastic resource management

In this section, we address the challenges of mobile video content delivery in a 5G network context. We present an architecture that allows content providers to deploy virtual CDN instances dynamically over a federated multi-domain edge cloud. Our system exposes a northbound API over which customers can request the creation of CDN slices for a specific duration. These CDN slices consist of VNFs deployed over the multi-domain cloud, such as virtual transcoders, virtual streaming servers, and virtual caches, appropriately chained together and configured with optimally-assigned resources for specific service-level performance targets. Cloud resource allocation decisions and elastic resource management, so that a virtual CDN instance is scaled up/down following end-user demand, are some of the main issues our design tackles. Resource allocation decisions are measurement-driven: with respect to compute resources, we use empirical models of video QoE as a function of service workload, while service demand information (e.g. maximum number of simultaneous video streams in an area covered by a

specific macro-cell) submitted by the customer at service instantiation time can be used to appropriately provision the CDN slice with radio access and core network resources.

Multi-level monitoring information, both at the cloud infrastructure and the CDN slice service/application level, is necessary for elastic slice resource management. Our design provides the support and the appropriate building blocks to collect and utilize this information, and it does so in a standards-based way: our architecture complies with recently proposed standards for Network Functions Virtualization, and the ETSI NFV Management and Orchestration (NFV-MANO) framework in particular.

3.6.1. Existing approaches

MEC and NFV are complementary technologies, sharing common concepts such as the existence of a virtualization infrastructure where applications can be launched and which is managed by a Virtualized Infrastructure Manager component. As such, the MEC and NFV platforms could be running independently or could share some reusable components (e.g. parts of the virtualization infrastructure and its management utilities).

Focusing on the specific video streaming service, Fajardo et al. [41] introduce a network-assisted adaptive streaming application to enhance QoE of the delivered multimedia content. Architecture with distributed parallel edges to increase QoE for content delivery has been proposed by Zhu et al. [42]. Chang et al. [43] deploy independent small-scale data-centers at the network edges, which are capable of performing video caching and streaming on their own. Jararweh et al. [44] integrate caching with proxy functionality at the edge to store media content. They also enforce computation offloading to increase the lifetime of mobile devices. Video transcoding in the cloud has recently received significant research attention. Utilizing virtual instances of the cloud to perform video transcoding upon request has been proposed in [45] as the simplest and straightforward use case. As an efficient way of video transcoding in the cloud, an approach to reduce the bit-rate of the transcoded video by using a higher quantization parameter without reducing the frame size or the frame rate has been proposed in [46]. Distributed video transcoding in the cloud to enhance efficiency have also been studied in [47].

In all the above research works, MEC is considered as a promising solution for handling video services, although mainly focusing on streaming, caching and compression techniques. The computationally-intensive transcoding functionality has been generally proposed to be treated on traditional clouds.

5G will offer the opportunity to dynamically build and manage end-to-end network slices [48]. A slice can be considered as an autonomous network service instance, encompassing all service-specific functionality as well as the respective resource management. Slices coexist over the same physical infrastructure by means of Cloud, NFV and SDN technologies. Resource management per slice, but also per participating network entity is a critical aspect. Li et al. [49] therefore differentiate between vertical and horizontal slices and the resource management aspects therein. The vertical case refers to resource sharing among different end-to-end services and applications, while the horizontal case focuses on resource sharing among network nodes and devices typically at the edge. Various types of value added services can be delivered over network slices. We focus

on content delivery and, to the best of our knowledge, this work is the first to deal in depth with this application case in this 5G slicing context.

As part of our prior research work ([50], [51] and [52]), we presented an architecture for the provision of video CDN functionality as a service over a multi-domain cloud. We introduced the concept of a CDN slice, that is, a CDN service instance which is created upon a content provider's request, autonomously managed, and spans multiple potentially heterogeneous edge cloud infrastructures. The content provider is able to manage a number of CDN slices running across multiple cloud domains. Basically, a virtual CDN slice consists of four main VNFs, namely virtual transcoders, virtual streamers, virtual caches, and a CDN-slice-specific Coordinator for the management of the slice resources and video contents across different private and public IaaS providers. A Multi-Domain Orchestration that manages the life-cycle of VNFs and scaling out and in the CDN slice in terms of virtual resources. Our design is tailored to a 5G mobile network context, building on its inherent programmability, management flexibility, and the availability of cloud resources at the mobile edge level, thus close to end users.

3.6.2. Proposed approach

In the context of CDNaaS, real-time slice resource management decisions require multi-level service monitoring. In this section, we detail the monitoring-related components of our architecture, and how they can be used to dynamically and elastically adjust the allocated resources to meet the desired performance criteria. We identify two major types of resources that affect user experience in a video content delivery setting: cloud and network-related ones. In the first case, we focus on compute ones, while in the second case, we are mostly interested in the conditions in the mobile edge and attempt to adjust the delivered video bitrate on a per-user basis to address network capacity and link quality changes.

Our design in Fig. 18 and Fig. 19 is ETSI NFV-MANO- and MEC-compliant. The components of our architecture map to some building blocks defined in the standard specification including the following proposed components at the CDNaaS Slice Orchestrator level: Resource Usage Monitor, Quality Assessor and Elasticity Decision Maker.

A. Resource Usage Monitor (RUM)

This component is responsible for monitoring resource usage for all VNF instances of a slice. Such information can be provided by the VIM. For example, if OpenStack is used as the VIM platform, as in our implementation, resource usage information per VM can be provided by OpenStack's Ceilometer module. RUM periodically monitors every instance's CPU and RAM utilization (i.e. other parameters are monitored too, but are not considered in the methods we are describing in this work) and communicates these information to the Elasticity Decision Maker (EDM) component.

It should be noted that scaling decisions should take into account capacity constraints: The allocated resources cannot exceed the capacity of the respective physical infrastructure. At a low level, such capacity information is made available by the VIMs and is passed on to the EDM by the RUM. The interactions between the RUM, QA, EDM, and the rest of our CDNaaS architecture are shown in Fig. 18. Fig. 19, on the other hand, focuses on the individual user monitoring

functionality of the Quality Assessor, and how the latter interacts with a MEO to initiate the transcoding service.

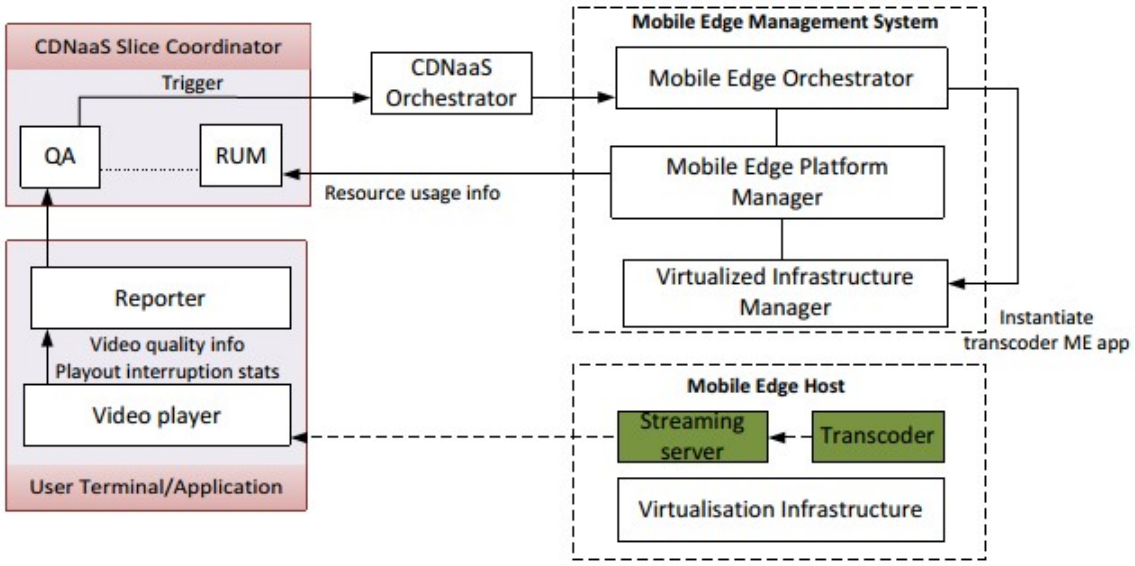


Fig. 18 – Cloud resource and QoE monitoring components at the CDNaaS Slice Coordinator level and their interactions

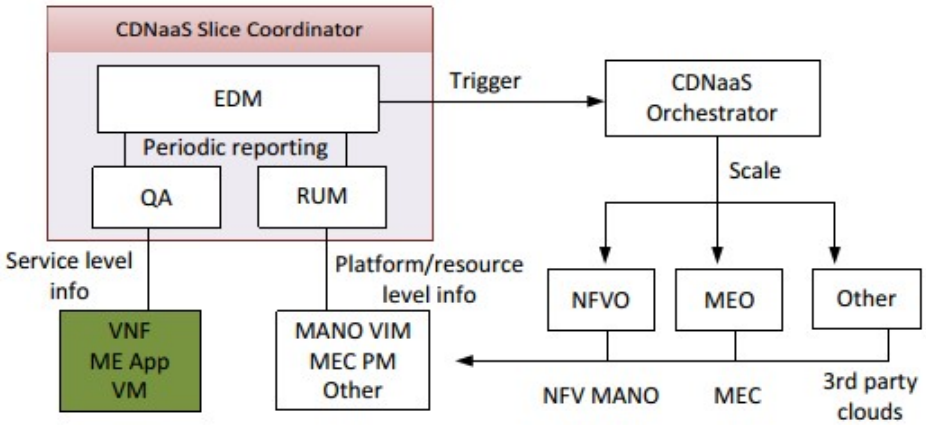


Fig. 19 – Individual user QoE monitoring functionality of the QA and transcoding instantiation on top of the edge cloud

B. Quality Assessor (QA)

System level information is important for resource management decision making, but do not offer a feel about how users perceive the CDN service. We are interested in having a user-centric view about the experience of the end-users of the CDN slice and, in turn, perform QoE-aware management decisions. To this end, we have introduced the QA component, which can estimate the CDN slice performance in QoE terms. Our proposed scheme thus decides on whether to scale resources or not also based on the QoE perceived by users, as estimated and reported by the QA.

The QA has two main operation modes that function in parallel. First, it operates as a QoE probe with the aim of getting an overall view of the QoE that can be delivered by a VNF instance from a compute perspective. Namely, the probe's purpose is to detect any degradation related with heavy workloads and not individual user characteristics (e.g. link quality and display size) or

network capacity issues. As such, it can be launched in the same host or data center as the monitored VNF instance(s) (to minimize network related effects) and emulate a user who receives a video stream from the specific monitored VNF instance. The probe records QoE-relevant parameters and then uses the PSQA [53] model to calculate a QoE estimate evaluated as MOS and reports it to the EDM. A simpler implementation option for the probe is to monitor the number of simultaneous video streams served by the VNF instance and use the empirical model of QoE vs. workload derived in [54] to estimate a MOS value. Our QoE metric is the MOS, i.e. the expected rating that a panel of users would give to the quality of the transmitted video in the 1-5 (poor-excellent) scale [53].

The second operation mode has to do with individual end-user QoE monitoring. It works under the assumption that at the user end, there is the appropriate functionality in place to record the relevant input parameters for QoE calculations and report them to the QA. The QA receives real-time information of the service status from the client. The status includes the ID of the last downloaded segment and of the playing segment, the playback interruption count and duration, and the QP value of the video playing at the end-user device. Considering these parameters, the PSQA model is used again to generate a MOS estimate in real-time in an automatic manner for a specific end-user. If the MOS value is below the target threshold set by the customer, the QA needs to verify that this is an individual end-user issue or that this is due to heavy load. This information is available by the RUM. In case resource utilization is within the acceptable thresholds, the QA triggers the CDNaaS Orchestrator to initiate the video transcoding service as illustrated in Fig. 19. The latter, in turn, communicates with the MEO of the edge network serving the specific user to instantiate the appropriate transcoder VNF on the edge cloud. In a different case, this is a problem that affects all users associated with a VNF instance and will be tackled by the Elasticity Decision Maker by scaling up the CDN slice.

C. Elasticity Decision Maker (EDM)

The main EDM functionality consists in deciding when to trigger the CDNaaS Service Orchestrator to enforce the elasticity operation of an instance, indicating what resources should be allocated to the instance while preventing service interruption. The EDM periodically receives monitoring information from both the RUM and QA. Based on that, it identifies the appropriate amount of resources to (re)allocate to an instance to meet the required service level objectives in terms of QoE, also avoiding cloud resource underutilization. The EDM employs a Multi Attribute Decision Making (MADM) algorithm using multiple inputs, such as the resources allotted to an active instance, resources being effectively in use, maximum resources that can be allocated for the service, instance flavor size, the available flavor types/sizes, and the MOS of the running VNF instance as reported by the QA. The thresholds of each of these inputs are preset for variable flavors and are tuned offline ensuring optimal operating conditions.

When EDM takes a resource re-allocation decision, it triggers up-/down-scaling of a running instance specifying its new size/flavor. The term scaling refers to increasing or decreasing the resources in terms of CPU, RAM, and storage. Scaling can be either vertical or horizontal. Through vertical scaling, the CDNaaS Orchestrator increases the computing power on a running instance, while horizontal scaling adds computing power in the form of adding another instance in parallel to the one in question. While horizontal scaling is relatively straightforward; it is costly as it

involves more VMs and is also more complicated from a management perspective. Furthermore, horizontal scaling requires load balancers to be more effective. In contrast, vertical scaling exhibits less operation cost but is more complex from an implementation point of view. In this work, we report on results obtained in the case of vertical scaling, as its horizontal counterpart is relatively easier to implement.

Note: We are using the term flavor, introduced in OpenStack, to represent a block of fixed resources in terms of CPU cores, RAM, storage, ephemeral disk and swap memory. When a flavor is allocated to a VNF instance, a certain amount of resources that remain fixed during the operation of that VNF instance are assigned to it.

The EDM compares the CPU usage and RAM usage (RUM output), as well as the MOS estimate (QA output) against their respective threshold values, and decides on how to adjust the allocated resources elastically as follows:

- In case the resource usage exceeds the maximum defined threshold and the MOS score is below the optimal value, the EDM initiates a vertical scale-up.
- In case resource usage is low and below a minimum threshold while MOS is acceptable, the EDM initiates a vertical scale-down.
- In case the resource usage is within the threshold range but the MOS value is below the optimal value, the EDM initiates a vertical scale-down.
- In case the resource usage is within the threshold range but the MOS value exceeds the optimal value, EDM takes no action in terms of elasticity, deeming the system being healthy.

3.6.3. Problem (re)definition

In a CDN slice running over multi-edge domains, the EDM component monitors the virtual resources of all VNFs composing the CDN slice. As explained earlier, in case of under-utilization or over-utilization, EDM triggers the CDNaaS Service Orchestrator to enforce the elasticity operation of an instance, specifying the new resources allocated to the instance (i.e. vertical scaling) or adding new VNF instances (i.e. horizontal scaling) to ensure the desired QoE. In this section, we introduce the underlying EDM algorithm responsible for the optimal dynamic resource allocation over a distributed heterogeneous VNF instances in a CDN slice. Our proposed algorithm is aimed at ensuring a balanced sessions' load over different instances and to guarantee the highest QoE required by the slice end-users while minimizing as much as possible the cost in terms of pricing (e.g. reducing the number of created VNF instances). According to our proposed architecture, EDM periodically receives monitoring information from both *(i)* the RUM component that monitors every instance's CPU and RAM utilization requested to the VIM(s) and communicates this information to the EDM; *(ii)* the QA component that estimates the CDN slice performance in terms of QoE based on PSQA ([53], [55] and [56]) and the number of streaming interruptions. Hence, the EDM algorithm can perform elasticity management decisions based on load and QoE awareness.

A. Mathematical modeling of the EDM algorithm

The proposed EDM algorithm is fed with the aforementioned information in order to identify the appropriate amount of resources to (re)allocate to an instance to meet the required service level objectives in terms of QoE and cost pricing, and to also avoid cloud resource underutilization/ overutilization. In this section, we model the physical network, representing the cloud infrastructure of a CDN slice. In this regard, we define $\vartheta = \{v_1, v_2, .. v_n\}$ as a set of running VNF instances hosted on different public and private IaaS. We define $\xi = \{\zeta_1, \zeta_2, .. \zeta_K\}$ as a set of all available cloud domains. We denote by $F = \{f_1, f_2, .. f_n\}$ the set of available flavors across all administrative cloud domains, whereby $f_{i,cpu}$, $f_{i,ram}$ and $f_{i,cost}$ denote, respectively, the number of CPU cores, memory capacity and the cost ($/hour) of flavor f_i. In other words, each cloud domain $\zeta_i \in \xi$ has a set of flavors has a set of flavors $\{f_{\zeta i,1}, f_{\zeta i,2}, .. f_{\zeta i,ki}\}$ where $\cup_{\zeta i \epsilon \xi}\{k \epsilon [1, k_i], f_{\zeta i,k}\} = F$. Since the QA and RUM components update the EDM component on the basis of a time period *T*, each VNF instance $v_i \epsilon\, \vartheta$ has a set of initial properties. In the beginning of every period, v_i is hosted on a cloud domain ξ with an initial flavor f_i^*. RUM also receives the average number of parallel streaming sessions ρ_i and the initial load l_i^* representing the average resource usage in a VNF instance v_i during the time period *T*. QA provides the estimated QoE Q_i^* from the perspective of the end-users of the CDN slice. For an optimal flavor assignment over VNF instances, we solve the problem using a branch and bound method. Each potential assignment of a flavor f_j to a VNF instance v_i we define a set of values in terms of cost, load and QoE. An assignment *(i,j)* represents three pieces of information when allocating flavor f_j to VNF instance v_i:

- The pricing cost c_{ij} in dollar per hour of the allocated flavor.
- The estimated CPU load l_{ij} based on the number of active parallel streaming sessions. The load can be considered optimal when $l_{ij} \epsilon [l_{min}, l_{max}]$. In our model, we estimate l_{ij} as $\left(l_{ij} = f_{j,cpu} \times \frac{l_{ij*}}{f*_{j,cpu}}\right)$.
- The estimated QoE can be provided if we allocate the virtual resources f_j to v_i.

In our prior work [54], our experiments revealed the linear performance scalability as the number of vCPUs allocated to a streamer instance grows: by adding one more vCPU to the instance, it could handle double the number of parallel video streams with no interruptions, keeping the same QoE level. Accordingly, we apply the equation introduced by Yala et al. [54] for every possible assignment of $f_j \in F$ to $v_i \in \vartheta$ as follows.

$$Q_{ij}(\rho) = 5 - \frac{1.046}{f_{j,cpu}} \times 10^{-8}\rho^2 + 5\sigma \times \eta(f_{j,cost}) \quad \text{(B1)}$$

In addition, we assume that the cost of a flavor has an impact σ on QoE (e.g. in case of two flavors with the same virtual resources, the cheaper one will provide a lower QoE). For this reason, we normalize the cost of flavor function using the normalization function η. In our simulations, we ensure that the number of parallel video sessions does not exceed ρ_{max} otherwise QoE takes negative values.

$$\rho \leq \rho_{max} = \sqrt{\frac{5 \times f_{j,cpu}}{1.048 \times 10^{-8}}} \quad \text{(B2)}$$

The aggregate utility minimization LP problem is shown as follows.

$$
\begin{cases}
\min z = \sum_{i\in\vartheta}\sum_{j\in F} c_{ij}.x_{ij} \\
\text{s. t.} \\
\quad \forall i \in \vartheta, \quad \sum_{j\in F} x_{ij} = 1 & (C1) \\
\quad \sum_{i\in\vartheta}\sum_{j\in F} x_{ij} = n & (C2) \\
\quad \forall i \in \vartheta, \quad \forall j \in F, \quad l_{ij}.x_{ij} \le l^{max} & (C3) \\
\quad \forall i \in \vartheta, \quad \forall j \in F, \quad l_{ij}.x_{ij} \ge l^{min} & (C4) \\
\quad \frac{1}{n}.\sum_{i\in\vartheta}\sum_{j\in F} Q_{ij}.x_{ij} \ge Q^{min} & (C5) \\
\quad \forall k \in \xi, \quad \sum_{j=k}^{k_{i+1}} f_{j,cpu}.\sum_{i\in\vartheta} x_{ij} \le C_{\zeta_k}^{max} & (C6) \\
\quad \forall i \in \vartheta, \quad \forall j \in F, \quad c_{ij}, l_{ij}, Q_{ij} > 0 & (C7) \\
\quad \forall i \in \vartheta, \quad \forall j \in F, \quad x_{ij} \in \{0,1\}, & (C8)
\end{cases}
$$

(B3)

The objective aims at reducing the total incurred cost while ensuring an optimal load and QoE required by end users receiving video content from all VNF instances. The constraints in our LP model are used to ensure the following conditions:

- (C1) & (C2): ensure that each instance will be assigned one and only one flavor.
- (C3) & (C4): ensure that all selected flavors can handle a load between l_{min} and l_{max}, so as to avoid under/overutilization of cloud resources.
- (C5): ensures that the average QoE in the new resource allocation satisfies the users' requirement. The average QoE of a CDN slice is defined as the mean QoE experienced at the total VNF instances composing the CDN slice.
- (C6): A cloud domain $\zeta_k \in \xi$ has a maximal capacity of resources $C_{\zeta_k}^{max}$. It represents the total number of CPU cores available at an IaaS. The constraint guarantees that the maximum capacity is not exceeded when allocating the flavors.
- (C7): ensures that the incurred cost, QoE and load parameters are valid values.
- (C8): shows binary variables whereby: $x_{ij} = \begin{cases} 1, & if\ flavor\ f_j\ is\ assigned\ to\ VNF\ v_i \\ 0, & otherwise \end{cases}$

3.6.4. Used algorithm: *EDM*

In the following chapter, we describe the operations of the EDM algorithm. The algorithm is run periodically after receiving updates from QA and RUM regarding the resource usages and QoE. It then finds an optimal assignment of flavors to active VNF instances using the linear programming (B3). In case an optimal solution is found, EDM will simply make scaling up/down decisions by increasing or reducing the virtual resources in the running VNF instances. However, in case the LP model is infeasible, we aim to figure out which boundaries make the solution not possible. We

assume that QoE is required for the satisfaction of end-users. So we first relax the model only from the load perspectives and set the maximal load l^{max} to 100 in constraint C3. If an optimal resource allocation is found, this means the slice will be over-utilizing its resources in order to ensure the minimum QoE required. Thus, EDM decides to add a new VNF instance to the CDN slice and runs again the algorithm with the *(n+1)* VNF instances. Then, we also relax the model from the other side and set the minimal load l^{min} to 0 in constraint C4. If a solution is found, this means that the slice will be under-utilizing its resources. Thus, EDM decides to remove some existing instances ensuring the minimal QoE required and load range of $[l_{min}, l_{max}]$. *Algorithm 1* describes more precisely the EDM elasticity for a dynamic allocation and re-allocation of virtual resources for a running CDN slice consisting of multiple VNFs. To summarize the main features of the algorithm in two main decisions, *(i)* EDM scales up and down the instances in terms of resources when the optimal solution is found; otherwise, *(ii)* it scales in and out the slice in case the LP model is infeasible.

Algorithm Elasticity Decision Maker Algorithm

Require: ϑ a set of running instances. ξ a set of available cloud domains. F a set of flavors. In a time period T.
while Time period T **do**
  Receive QoE information from QA and monitoring and resource usage from RUM.
  Run linear programming model (3): find an optimal allocation of flavors to running VNF instances.
  while Model is infeasible **do**
    Remove constraint (C3).
    Check if an optimum is found with a $load \geq l^{max}$.
    if Yes: possible optimization with over-utilization resources **then**
      Scaling out actions: add new VNF instance.
      Repeat: considering the newly added VNF instance.
    end if
    Restore constraint (C3) and remove constraint (C4).
    Check if an optimum is found with a $load \leq l^{min}$.
    if Yes: possible optimization with under-utilization resources **then**
      Scaling in actions: remove an existing VNF instance.
      Repeat: considering the newly removed instance.
    end if
  end while
  Scaling up/down actions: Contact VIM(s) for new resource re-allocation to existing instances.
end while

3.6.5. Results: efficiency and scalability of the EDM algorithm

A. Experiment setup

To simulate our proposed solution, we developed the EDM algorithm using Python programming language. The linear programming model (B3) is implemented using the Gurobi Optimization tool and is evaluated using the following metrics: *(i)* Mean Cost per unit representing the average pricing cost paid in dollar per instance for its virtual resource allocation; *(ii)* QoE-aware resource allocation representing the performance when users require a minimal QoE to be ensured; and

(iii) Execution time representing the time required for the EDM algorithm to take an optimal decision. The optimization problem is solved by varying the number of VNF instances in a CDN slice and other parameters related to slice owner or user requirements including the open sessions load and minimal QoE required. In our simulations, we consider 45 cloud domains and 1,417 flavors in total with their respective pricing cost in dollar per hour. This information is collected from real IaaS platforms such as Amazon AWS service, Microsoft Azure and Rackspace. Hence, using real data reflects accurate and meaningful findings in our final simulation results. For the sake of simplicity, the total number available CPU cores C^{max} in all cloud domains is set to 20, and the impact of the flavor cost on QoE of a single instance σ is set to 0.1. Finally, we run the simulations varying the number of VNF instances from 10 to 300 with a step of 10. Every instance is defined by an initial flavor chosen randomly from the set of 1.417 flavors, a number of active parallel streaming sessions respecting (B2). Then, an estimated QoE is deduced in (B1) based on results in [54]. In our simulation results (i.e. Fig. 20 and Fig. 21), each plotted point represents an average of 35 times of executions. The plots are presented with 95% confidence interval.

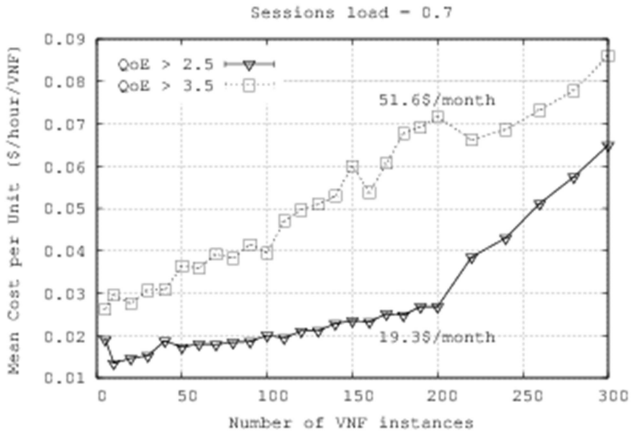
B. Performance with QoE-awareness & Load-awareness

Fig. 20a shows the performance of our EDM algorithm in terms of mean cost of flavor per a deployed VNF instance. In this experiment, we set the average session load in a CDN slice to 0.7, and run the EDM algorithm to find the optimal assignment of flavors to VNF instances with the objective of minimizing the total deployment cost. Fig. 20a shows that the mean cost per unit is much reduced when a lower QoE is required. For a QoE exceeding 3.5, the deployment of 200 VNF instances will cost 51.6$ per month for each instance. However, it will cost only 19.3$ per month in case the minimal QoE required is 2.5. Additionally, Fig. 20b shows an intuitive result that the mean cost per unit is much higher in case of an over-loaded system. For a system load greater than 50%, the deployment of 200 VNF instances will cost 40.7$ per month for each instance. However, it will cost nearly 212$ per month in case the minimal load required is 80%. Figs.3.a and 3.b also show an increasing mean cost per unit when increasing the size of the CDN slice. This is due to the limitations at the hosting cloud domain in terms of the amount of available resources and the maximum session load an instance can handle.

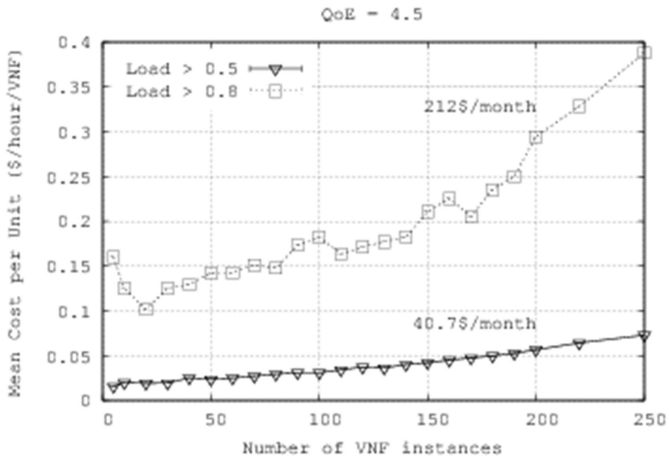
Based on the obtained results, the EDM algorithm is able to take reverse decisions. When the end-users require a low QoE and the slice owner has a budget limitation, the EDM algorithm can reversely propose the maximal number of VNF instances that can be deployed in a CDN slice respecting the owner's specified budget.

C. Scalability of the EDM algorithm

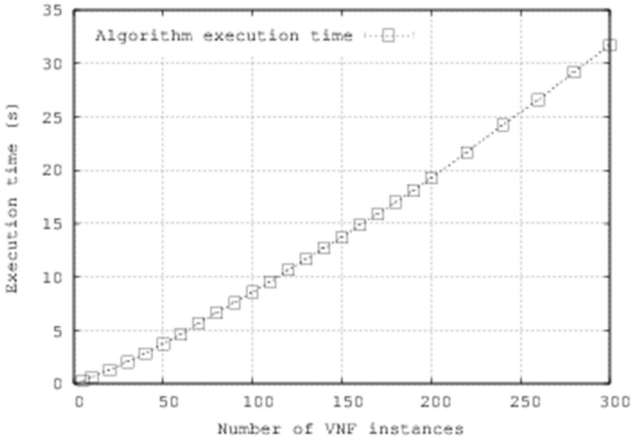
Fig. 20c presents the execution times of the proposed EDM algorithm for different sizes of the target CDN slice. The figure shows the execution time as a linear function of the number of VNF instances. For 10 instances, the algorithm spends 10 seconds to find the optimal resource allocation out of the available 1,417 flavors. The execution time becomes around 30 seconds in case of a CDN slice of a size of 300 VNF instances. Also, we got approximate results when changing other parameters including $Q^{min}, l^{min}, l^{max}, C^{max}$ and σ Thus, we conclude that the execution time depends only on the number of VNF instances and can be estimated as follows: $\Delta t = 10 \times instances$.



(a) – Mean cost per instance in case of different QoE requirements and for varying numbers of VNF instances. The minimum session load is set to 0.7.



(b) – Mean cost per instance in case of different session loads and for varying numbers of VNF instances. The minimum QoE is set to 4.75.

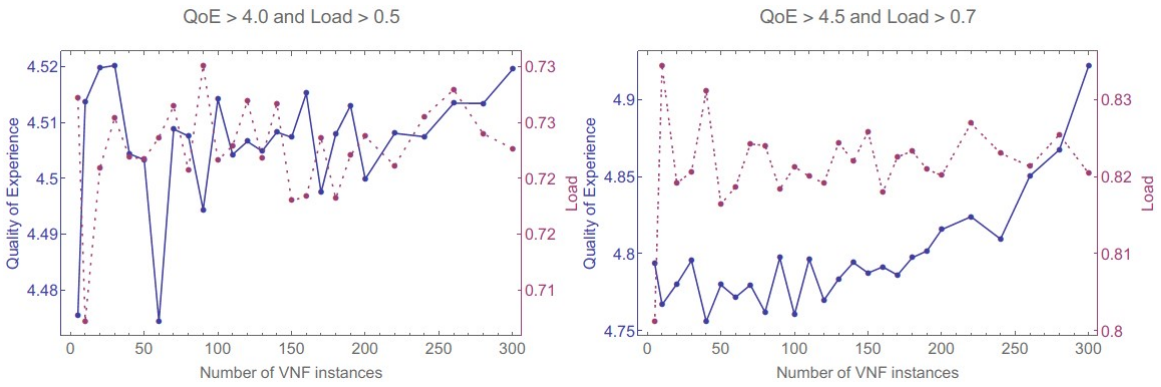


(c) – EDM algorithm's execution time as a function of the CDN slice size.

Fig. 20 – Mean cost per unit and the EDM algorithm's execution time as a function of the CDN slice size (KVM VM with 16 CPU cores).

D. QoE behavior against system load

Fig. 21 presents the average QoE experienced by end-users of a CDN slice of different sizes and that is under different loads. Fig. 21a shows the correlation between load and QoE when the minimum system load and the minimum desired QoE are set to 50% and 4, respectively. Fig. 21b shows the same in case of a higher system load (greater than 70%) and a higher QoE (greater than 4.5). Both Fig. 21a and Fig. 21b show that with less system load, the CDN slice performs better and exhibits better QoE values, exceeding always the minimum QoE value.



(a) – Average QoE behavior vs. average load variation, when minimizing the deployment pricing cost with minimum requirements: minimal QoE = 4 and minimal load = 0.5.

(b) – Average QoE behavior vs. average load variation, when minimizing the deployment pricing cost with minimum requirements: minimal QoE = 4.5 and minimal load = 0.7.

Fig. 21 – Average QoE behavior vs. average load variation in a CDN slice when increasing the number of VNF instances

3.6.6. Final remarks

We identified opportunities for cooperation between content providers and network operators for QoE-optimized over-the-top mobile video delivery. In particular, leveraging MEC and NFV technologies, we expanded on the architectural and technical aspects of the dynamic instantiation and elastic management of CDN slices over shared, multi-domain, edge (and other) cloud infrastructures. The design and mechanisms we proposed come with improved QoE awareness. This is valuable both at service instantiation time, when our empirical models facilitate informed CDN slice resource dimensioning decisions, but also at run time, when multi-level monitoring information are applied to resource scaling and video-service-level adaptations, in order to optimize service quality and resource utilization.

4. Inter-slice orchestration

For the inter-slice orchestration, a new set of mechanisms will be devised for slice sharing between different applications such as IoT services, enabling the services to connect to the same devices using a common connectivity slice. For this purpose, resource allocation mechanisms for the different slices will be extended to support such slice correlations.

4.1. Dynamic resource provisioning for slice stitching

Dynamic resource provisioning is also requested for slice stitching. Especially, the stitching happens while one or more slices to be stitched are in operation. Depending on the nature of involved slices, different conditions have to be taken into account in designing the provisioning algorithm, but this section describes the basic problems inherent to the case of ICN and CDN slices stitching for an efficient content delivery service.

4.1.1. Introduction

The use case "ICN/CDN combined contents delivery" is aimed at providing an efficient content delivery service by combining the Dynamic CDN slice (CDNaaS) of Aalto University and the ICN/NDN slice that Japanese partners (Waseda University together with University of Tokyo and Hitachi) are trying to realize, based on 5G!Pagoda architecture. Contents delivery, especially video contents, is already a major traffic occupying around 70% in mobile communication, and still increasing. This will cause a risk of the core network congestion when 5G mobile system starts its operation. 5G!Pagoda approach to this congestion problem of "ICN/CDN combined contents delivery" is to combine two types of slices dynamically. One of them is a dynamic CDN slice that allocates the contents cache servers in the optimal locations depending on the nature of the offered contents and the possible user's distribution. Cache servers will be placed in different domains even across the national boundaries. Another slice is an ICN slice that is used in the regional distribution part of the network. ICN slice serves efficient contents delivery due to the robustness of name-based access and the autonomic contents caching by the network nodes. These two factors enable the contents delivery from the nearby network node, thus reducing the duplicated contents transfer on the transmission link and providing shorter response time. This use case also intends to demonstrate some of the innovative features of 5G!Pagoda. Those are the deep data plane programmability, the slice stitching (slice chaining) capability, Multi-domain orchestration, optimum VNF allocation and slice scalability. Two slices have different life-cycles. ICN slice has a longer life-cycle, connecting varying contents sources. On the contrary, dynamic CDN slice is dedicated to special events, hence, has a shorter life-cycle. Possible example is the coming case of the Olympic Games in Tokyo. The dynamic CDN slice for Olympic Games will be created covering many domains such as Japan, Finland, Germany and France, and cache servers are placed at locations based on the provisioning algorithm. Then these cache servers are stitched to the existing ICN slices which cover each domain. For the ICN slice, created CDN cache servers are treated as new contents sources. Since the protocols and the data formats of each slice are

different, a gateway function is requested to stitch these slices. The gateway function is created in the ICN slice based on the provisioning algorithm.

4.1.2. The basic operations of ICN/CDN slice stitching

Fig. 22 shows the overview of ICN/CDN combined contents delivery service. This approach aims at reducing the traffic of core network with the use of dynamic CDN slice creation using the optimum location assignment of content servers, and the use of ICN slice in regional distribution part of the network, and by combining these two types of slices dynamically. In this use case scenario, as it was mentioned in the previous section, the ICN slice has longer lifetime supporting a variety of contents provided by a variety of contents creators, and the CDN slice will be dynamically created for special events, therefore it has a shorter lifetime. The slice stitching happens, when the new CDN slice is created and connected to the existing ICN slice. There is a difference of communication protocols used in both slices. And also the packet format of content delivery is different. Therefore, a gateway function is requested to combine these two slices. The gateway function is created in ICN slice by assigning one of the ICN nodes to perform the gateway functionality. Together with the creation of the gateway function, the communication links establishment between the ICN gateway and CDN content caches is necessary for content retrieval, as well as the CDN organizer for management of information communication. This process requires dynamic resource assignment to ICN slice. The detailed sequence is described in Section 4.1.5.

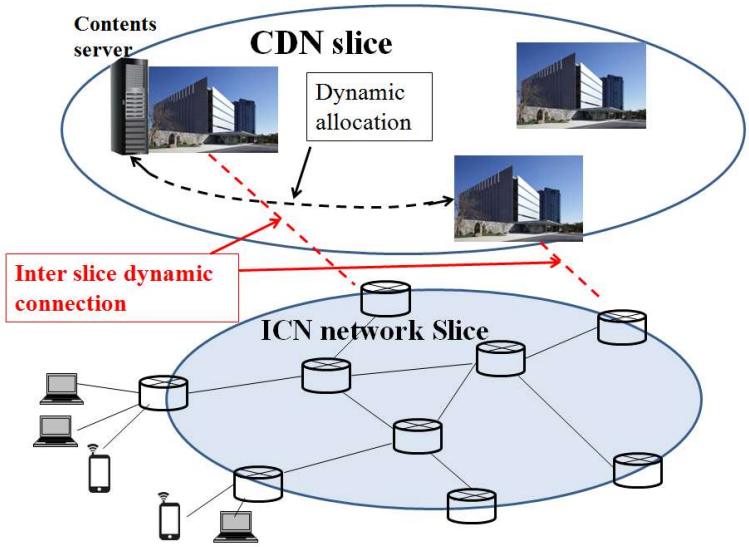


Fig. 22 – Slice stitching of CDN slice and ICN slice for an efficient content delivery service

4.1.3. Dynamic resource assignment problems inherent to slice stitching

This section points out the factors to be included in the dynamic resource allocation algorithm related to slice stitching. The key issue here is gateways placement, for which the following factors have to be considered:

- Link cost: It is better to place the gateway function at the ICN node closest to the CDN cache server of the dynamic CDN slice. In this scenario, however, the contents go through the link between CDN cache server and the ICN gateway basically only once, the weight of this requirement is smaller than other slice stitching scenario.
- Available resource: When the ICN node is assigned to function as a gateway, it is preferred to enlarge the cache capacity of that node to improve the cache hit ratio and avoid the duplicated interaction with the dynamic CDN slice.
- Flow centrality: It is preferred that the gateway is placed at the node having higher flow centrality. When the total sum of the links from the node to all the other nodes is smaller, the node is considered as having higher flow centrality. In this use case scenario, the link counting should be modified as total sum minus duplicated counts of the link. For example, in case of tree-type topology, the upper level node has higher flow centrality.
- Gateway distribution: It is possible that ICN slice is connected to several CDN slices at the same time. Then it may be better to allocate the gateway to each CDN in a distributed manner rather than having a centralized gateway. This needs further study.
- User distribution: It is preferred that the gateway is placed in highly populated areas. In ICN, this weight is not high because the contents are autonomously cached at the place where the request is frequent.

4.1.4. Functions and procedure for slice-stitching based on "Resource Pool"

As mentioned in the previous section, in order to provide and develop a new and flexible service such as "ICN/CDN combined contents delivery", it is necessary to analyze the life-cycle of service and network resources and to provide a mechanism for dynamically allocating available service and network resources to the ICN/CDN service provider. Recently, management of network slices life-cycle has caught much attention as well as construction of network slices. Section 5 of D4.1 [57] introduces "Resource Pool" concept to realize an agile and flexible service on top of network slices. Fig. 23 describes the proposed "Resource Pool" architecture. The network infrastructure and the service layer are separated and/or isolated by the newly introduced ML ("Resource Pooling") which makes use of "Resource Pool" to implement life-cycle management of the service and network layer resources, especially to accelerate resource allocation, deallocation and reallocation.

There exist both requirement and architectural gaps between the top, and bottom layer, i.e. the top layer is a service layer provided by ICN service provider, and the bottom layer is a virtualized infrastructure layer provided by infrastructure providers.

In order to compensate these gaps, a middle layer (ML) is constructed. The important function of the ML is "*response time acceleration*", which complements the temporal gap between the top and the bottom layers. Specifically, a long life-cycle time of conventional infrastructure facilities cannot meet a short life-cycle time required for the service layer. Therefore, the ML provides it by analyzing the upper layer requests, caching necessary resources, collecting individual settings as

templates, and providing them as the pooling resources for the service layer as immediately available resources.

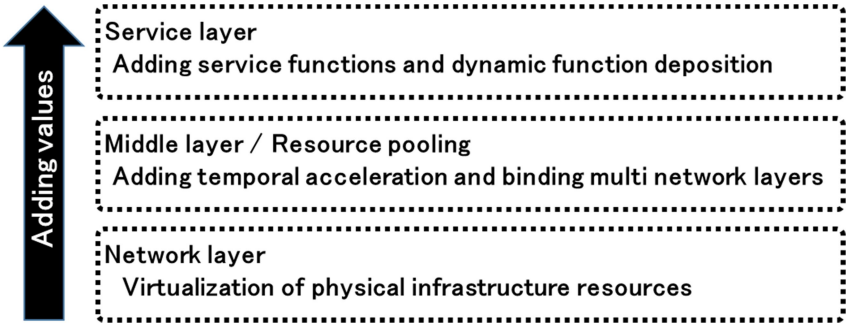


Fig. 23 – "Resource Pool" architecture

The effect of the acceleration function is divided into two parts. One is a provisioning function referred to as "*resource pooling*", which caches resources and shortens the response preparing time for the resource as a state that can be used immediately. The other is a profiling function referred to as "*smart provisioning*", which evaluates the state of resources to be pooled, adjusts the composition of resources to be prepared according to the analysis of service demands, and it improves the turnover rate of pooling resources. Fig. 24 describes the "Resource Pool" of Slice Orchestrator.

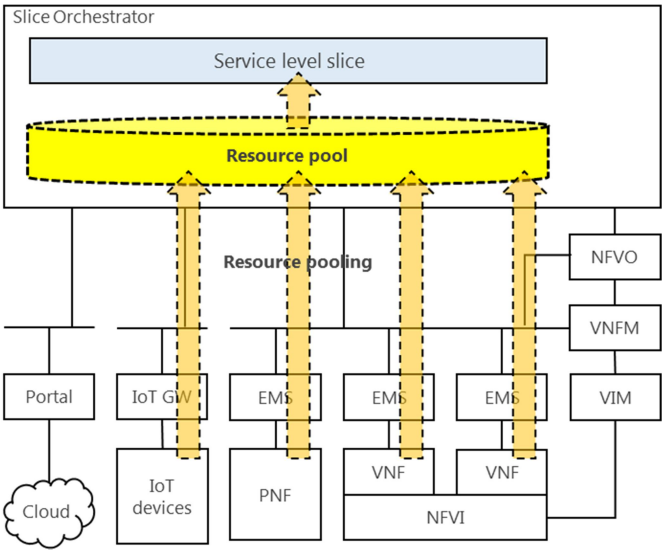


Fig. 24 – "Resource Pool" of Slice Orchestrator

In order to make resource pooling effective, inventory management of pooling resources is important, and the following consideration is necessary:

- Quantitative balance adjustment of pooling resources;
- Matching allocated resources according to request;
- Adjustment according to service/resource situation.

Smart provisioning realizes automatic management, by monitoring service status and resource usage status, creating KPIs and manifests from the resource production plan. The resource production plan is built based on the turnover rate of pooling resources, provisioning time,

resource usage status, which can be obtained from the EMS of the network layer, and the service status, which is acquired in cooperation with OSS/BSS. Various analytical methods can be used by organizing the monitoring items.

Section 5 of D4.1 describes the basic architecture for scalable orchestration of slices configured with multiple technology domains. The section also extends the architecture to achieve the federated operation of slices from multiple maintenance/operational domains. "Resource Pooling" is considered to be the core of the architecture, which can be generalized as the ML capabilities between the network infrastructure and the service layer.

The ML should have various capabilities; orchestration of functional components from multiple domains, mediation of virtual resources to adopt service developer requirements in some cases, event handling/filtering between the upper and lower layers, maintaining the system-wide integrity of networking attributes like policy. The ML also plays a significant role in case of slice stitching.

Two aspects have to be considered for such arrangements, vertical and horizontal segmentation or integration of bounded system contexts.

Vertical segmentation can be defined as a layered operation, which includes slice federation and service interaction between the stitched slices. The slice federation performs the interconnection of domain-specific slices, each of them operated within a dedicated maintenance domain. The service interaction procedures handle the service functions coordination between the dedicated service slices, simultaneously and independently.

In order to achieve the stitching of the vertically separated functions, a gateway capability should be configured between the stitched slices, which is considered as the horizontal integration of the bounded contexts. Networking related coordination like addressing scheme traversal and policy integrity maintenance is the gateway capabilities for the ML. The ML management function that is almost similar to the so-called orchestrator in a broader sense can exchange the information related to such network-related coordination via the gateway.

On the contrary, the service interaction functions on each of the service dedicated slices exchange the information for service layer federation independent from the ML operation. The ML provides a communication channel for this information exchange for the service level coordination, by interconnecting the ML capabilities on each side through the gateway. The resource pooling on each domain-specific slice can handle the resources in another side of stitching as those of its own; even the resource should be handled through the gateway.

4.1.5. Service sequence for "ICN/CDN combined contents delivery"

The following sequence charts illustrate the operational sequences with demo system-dependent details:

- FLARE resource registration (Fig. 25);
- ICN slice creation ~ activation (Fig. 26);
- ICN/CDN slices stitching (Fig. 27).

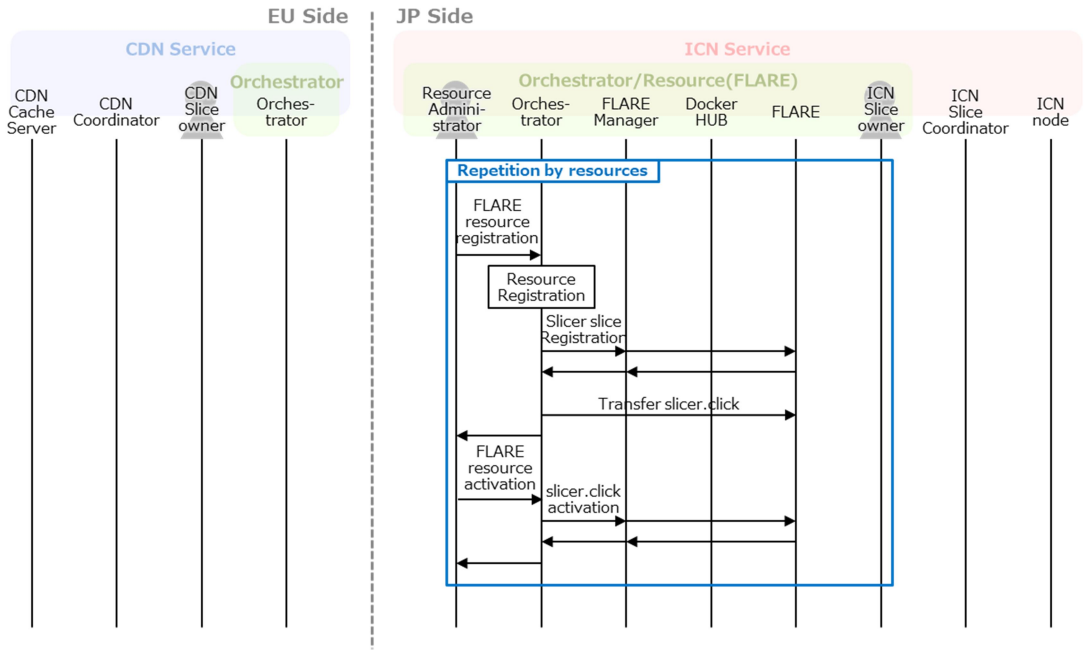


Fig. 25 – FLARE resource registration

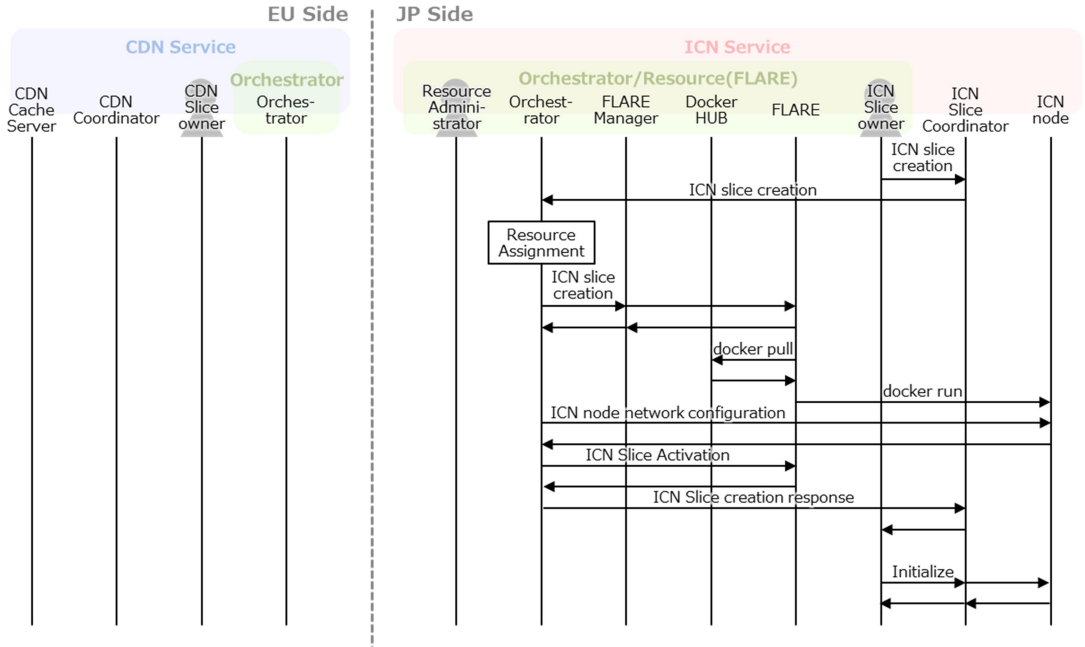


Fig. 26 – ICN slice creation ~ activation

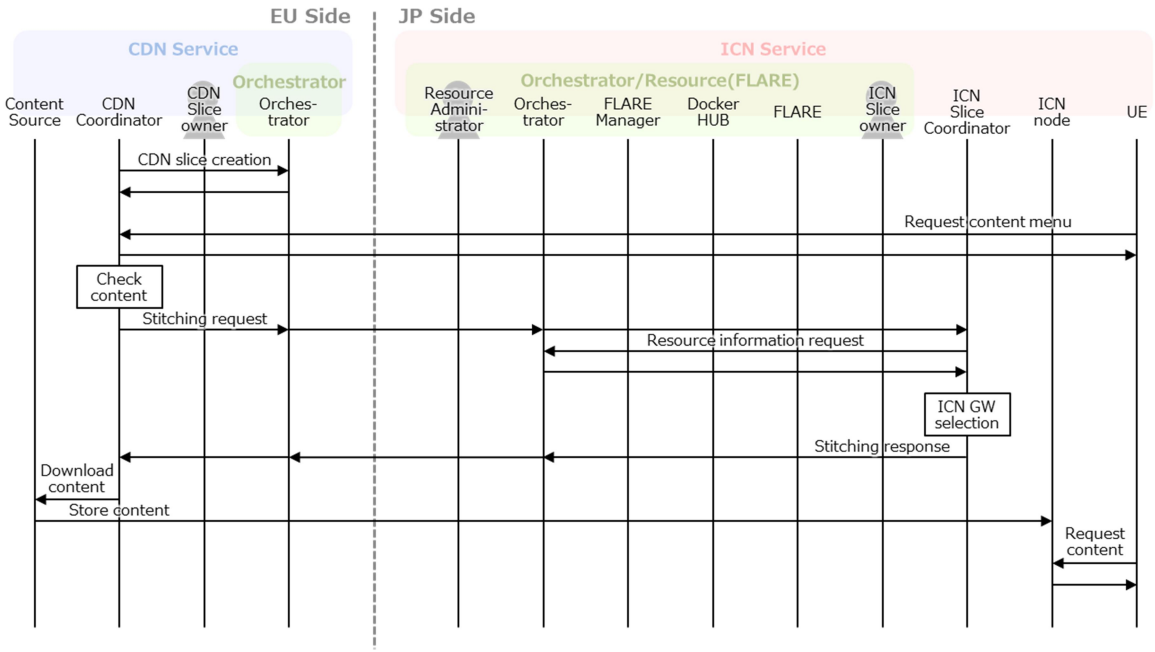


Fig. 27 – ICN/CDN slice stitching

5. Concluding remarks

The slice orchestration is a complex issue that can be decomposed into intra-slice and inter-slice issues. One of the key aspects of the intra-slice orchestration is the efficient initial placement of virtual network functions and optimization of the placement during slice runtime. For this operation the NFVO of the ETSI MANO is responsible. Despite some ETSI standardization efforts the implementation of the NFVO algorithm that is responsible for the VNF placement is (fortunately) still left to the implementers. Such approach gives the possibility of selecting the appropriate algorithm according to the needs. As the VNF placement problem is known to be NP-hard, many heuristic solutions to the problem exist. They may differ in computational complexity, the quality of the decisions and in the processing time. The problem of virtual functions placement is split in this document into an initial placement that is based on some assumptions and the on-line placement optimization approach that is based on the monitoring of the deployed network. As the problem is algorithmic one, there is no single best solution, therefore in this deliverable, several different approaches to virtual network functions placement have been presented. Some of them are based on sophisticated algorithms whereas others look into more pragmatic, simpler but less efficient solutions. This document introduced different algorithms related to Network Service or slice orchestration that concern: dynamic service chain management, joint optimization of number of VNFs and their placement in case of 4G vEPC, an algorithm based on Control Theory allowing to balance the load on the 5G AMF instances in order to maintain an optimal response time with limited computing latency, a pragmatic approach to online optimization of vEPC nodes placement, and a slice-based CDN architecture with real-time resource management based on multi-level service monitoring. Some of them are based on sophisticated analytic tools whereas others (i.e. the pragmatic approach to vEPC) are based on simple heuristics that exploit the properties of EPC and the telco cloud infrastructure.

In this deliverable, issues relevant to inter-slice orchestration were discussed based on an integrated ICN/CDN slice example. Different aspects were discussed and that includes dynamic resource assignment for slice stitching, distribution of gateways, and issues related to fast slice provisioning based on the resource pool concept.

References

[1] M. Xia, M. Shirazipour, Y. Zhang, H. Green and A. Takacs, "Network Function Placement for NFV Chaining in Packet/Optical Datacenters", in Journal of Lightwave Technology, vol. 33, no. 8, Apr. 2015, pp. 1565-1570.

[2] B. Németh, B. Sonkoly, M. Rost and S. Schmid, "Efficient service graph embedding: A practical approach", in Proceedings of 2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), Nov. 2016, pp. 19-25.

[3] C. Pham, N. H. Tran, S. Ren, W. Saad and C. S. Hong, "Traffic-aware and Energy-efficient vNF Placement for Service Chaining: Joint Sampling and Matching Approach", in IEEE Transactions on Services Computing, Feb. 2017.

[4] A. Laghrissi, T. Taleb, M. Bagaa and H. Flinck, "Towards Edge Slicing: VNF Placement Algorithms for a Dynamic & Realistic Edge Cloud Environment", GLOBECOM 2017 - 2017 IEEE Global Communications Conference, Dec. 2017, pp. 1-6.

[5] J. T. Piao and J. Yan, "A network-aware virtual machine placement and migration approach in cloud computing", in Proceedings of 2010 9th International Conference on Grid and Cloud Computing, Nov. 2010, pp. 87-92.

[6] M. G. Rabbani, R. Pereira Esteves, M. Podlesny, G. Simon, L. Zambenedetti Granville and R. Boutaba, "On tackling virtual data center embedding problem", in 2013 IFIP/IEEE International Symposium on Proceedings of Integrated Network Management (IM 2013), May 2013, pp. 177-184.

[7] G. Somani, P. Khandelwal and K. Phatnani, "VUPIC Virtual Machine Usage Based Placement in IaaS Cloud", CoRR abs/1212.0085 (2012).

[8] S. Skiena, "The Algorithm Design Manual", ISBN 0-387-94860-0.

[9] S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi, "Optimization by Simulated Annealing Science", 220 (1983) 671-680.

[10] M. Dorigo and T. Sttzle, "Ant Colony Optimization", ISBN 978-0-262-04219-2.

[11] K. Le, J. Zhang, J. Meng, R. Bianchini, Y. Jaluria and T. D. Nguyen, "Reducing Electricity Cost Through Virtual Machine Placement in High Performance Computing Clouds", in Proceedings of Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, Nov. 2011.

[12] C. Hyser, B. McKee, R. Gardner and B. J. Watson, "Autonomic Virtual Machine Placement in the Data Center", in Proceedings of HP Labs, HPL-2007-189, Feb. 2008.

[13] W. Shi and B. Hong, "Towards Profitable Virtual Machine Placement in the Data Center", in Proceedings of 2011 IEEE 4th International Conference on Utility and Cloud Computing, Dec. 2011.

[14] L. Mashayekhy and D. Grosu, "A Coalitional Game-Based Mechanism for Forming Cloud Federations", in Proceedings of 2012 IEEE 5th International Conference on Utility and Cloud Computing, Nov. 2012, pp. 223-227.

[15] B. K. Ray, S. Khatua and S. Roy, "Cloud Federation Formation Using Coalitional Game Theory", in Proceedings of ICDCIT 2015 11th International Conference on Distributed Computing and Internet Technology, Feb. 2015.

[16] M. Guazzone, C. Anglano and M. Sereno, "A Game-Theoretic Approach to Coalition Formation in Green Cloud Federations", in Proceedings of 2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), May 2014, pp. 618-625.

[17] T. Taleb and A. Ksentini, "Gateway Relocation Avoidance-Aware Network Function Placement in Carrier Cloud", in Proceedings of ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems, Nov. 2013.

[18] M. Bagaa, T. Taleb and A. Ksentini, "Service-Aware Network Function Placement for Efficient Traffic Handling in Carrier Cloud", in Proceedings of 2014 IEEE Wireless Communications and Networking Conference (WCNC), Apr. 2014.

[19] F. Z. Yousef, J. Lessmann, P. Loureiro and S. Schmid, "SoftEPC Dynamic Instantiation of Mobile Core Network Entities for Efficient Resource Utilization", in Proceedings of 2013 IEEE International Conference on Communications (ICC), Jun. 2013.

[20] ETSI, "Network Function Virtualisation (NFV); Management and Orchestration", ETSI Group Specification NFV-MAN 001, Dec. 2014.

[21] 3GPP, "Study on management and orchestration of network slicing for next generation network", 3GPP Technical Report TR 28.801 version 15.1.0, Jan. 2017.

[22] D. Knuth, "The Art of Computer Programming", Vol. 4, Fascicle 3: "Generating All Combinations and Partitions", Addison-Wesley Professional, 2005.

[23] Networkx, http://networkx.lanl.gov.

[24] 3GPP, "Study on Architecture for next Generation Systems", 3GPP Technical Report TR 23.799, version 14.0.0, Dec. 2016.

[25] D. Nowoswiat, "Managing LTE Core Network Signaling Traffic", https://insight.nokia.com/managing-lte-core-network-signaling-traffic.

[26] H. Hawilo, A. Shami, M. Mirahmadi and R. Asal, "NFV: state of the art, challenges, and implementation in next generation mobile networks (vEPC)", IEEE Network, vol. 28, no. 6, 2014, pp. 18-26.

[27] T. Phung-Duc, Y. Ren, J.-C. Chen and Z.-W. Yu, "Design and Analysis of Deadline and Budget Constrained Autoscaling (DBCA) Algorithm for 5G Mobile Networks", arXiv preprint arXiv:1609.09368, 2016.

[28] Y. Ren, T. Phung-Duc, J.-C. Chen and Z.-W. Yu, "Dynamic Auto Scaling Algorithm (DASA) for 5G mobile networks", in 2016 IEEE Global Communications Conference (GLOBECOM), Dec. 2016.

[29] S. Jeon, D. Corujo and R. L. Aguiar, "Virtualised EPC for on-demand mobile traffic offloading in 5G environments", in 2015 IEEE Conference on Standards for Communications and Networking (CSCN), Oct. 2015.

[30] Y. Takano, A. Khan, M. Tamura, S. Iwashina and T. Shimizu, "Virtualization-based scaling methods for stateful cellular network nodes using elastic core architecture", in 2014 IEEE 6th International Conference on Cloud Computing Technology and Science, Dec. 2014.

[31] X. An, F. Fabio, I. Widjaja and U. G. Acer, "DMME: A distributed LTE mobility management entity", Bell Labs Technical Journal, vol. 17, no. 2, 2012, pp. 97-120.

[32] A. Banerjee, R. Mahindra, K. Sundaresan, S. Kasera, K. van der Merwe and S. Rangarajan, "Scaling the LTE control-plane for future mobile access", Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies, CoNEXT, Dec. 2015,

[33] J. Prados-Garzon, J. J. Ramos-Munoz, P. Ameigeiras, P. Andres-Maldonado and J. M. Lopez-Soler, "Modeling and Dimensioning of a Virtualized MME for 5G Mobile Networks",arXiv preprint arXiv:1703.04445, 2017.

[34] G. Premsankar, K. Ahokas and S. Luukkainen, "Design and Implementation of a Distributed Mobility Management Entity on OpenStack", in IEEE CloudCom Nov.-Dec. 2015.

[35] M. C. De Oliveira, J. Bernussou, Jacques and J. C. Geromel, "A new discrete-time robust stability condition", Systems & Control letters, vol. 37, no. 4, 1999, pp. 261-265.

[36] R. E. Kalman, "Contributions to the theory of optimal control", 1960.

[37] K. Zhou, Kemin, J. C. Doyle and K. Glover, "Robust and optimal control", Prentice Hall New Jersey, vol. 40, 1996.

[38] C. Wang, B. Li, Y. T. Hou, K. Sohraby and Y. Lin, "LRED: a robust active queue management scheme based on packet loss ratio", INFOCOM 2004, 23rd Annual Joint Conference of the IEEE Computer and Communications Societies, Nov. 2004.

[39] F. Slim, F. Guillemin, A. Gravey, Y. Hadjadj-Aoul, "Towards a Dynamic Adaptive Placement of Virtual Network Functions under ONAP", 3rd International NFV-SDN'17-O4SDI - Workshop on Orchestration for Software-Defined Infrastructures, Nov. 2017.

[40] H. Alipour, F. Belqasmi, M. Abu-Lebdeh, R. Glitho, "Towards HSS as a virtualized service for 5G networks", 2015 11th International Conference on Network and Service Management (CNSM), Nov. 2015.

[41] J. Fajardo, I. Taboada and F. Liberal, "Improving content delivery efficiency through multi-layer mobile edge adaptation", IEEE Network, vol. 29, no. 6, 2015, pp. 40-46.

[42] W. Zhu, C. Luo, J. Wang and S. Li, "Multimedia Cloud Computing", IEEE Signal Processing Magazine, vol. 28, no. 3, May 2011, pp. 59-69.

[43] H. Chang, A. Hari, S. Mukherjee and T. Lakshman, "Bringing the cloud to the edge", in Proceedings of Workshops of IEEE INFOCOM 2014 - IEEE Conference on Computer Communications, Apr.-May 2014.

[44] Y. Jararweh, L. Tawalbeh, F. Ababneh and F.Dosari, "Resource Efficient Mobile Computing Using Cloudlet Infrastructure", in Proceedings of IEEE 9th International Conference on Mobile Ad-hoc and Sensor Networks (MSN), Dec. 2013.

[45] Y. Zha, H. Jiang, K. Zhou, Z. Huang and P. Huang, "Meeting Service Level Agreement Cost-Effectively for Video-on-Demand Applications in the Cloud", in Proceedings of IEEE INFOCOM 2014 - IEEE Conference on Computer Communications, Apr.-May 2014.

[46] F. Jokhio, T. Deneke, S. Lafond and J. Lilius, "Bit rate reduction video transcoding with distributed computing", in Proceedings of 20th International Conference on Parallel, Distributed and Network-Based Processing, Feb. 2012.

[47] M. Kim, S. Han, Y. Cui, H. Lee, H. Cho and S. Hwang, "CloudDMSS: Robust Hadoop-Based multimedia streaming service architecture for a cloud computing environment", Cluster Computing, vol. 17, no. 3, Sep. 2014, pp. 605-628.

[48] I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini and H. Flinck, "Network Slicing & Softwarization: A Survey on Principles, Enabling Technologies & Solutions", IEEE Communications Surveys Tutorials, vol. PP, no. 99, Mar. 2018, pp. 1-1.

[49] Q. Li, G. Wu, A. Papathanassiou and U. Mukherjee, "An end-to-end network slicing framework for 5G wireless communication systems", CoRR, vol. abs/1608.00572, 2016.

[50] S. Retal, M. Bagaa, T. Taleb and H. Flinck, "Content Delivery Network slicing: QoE and cost awareness", in Proceedings of 2017 IEEE International Conference on Communications (ICC), May 2017, pp. 1-6.

[51] I. Benkacem, T. Taleb, M. Bagaa and H. Flinck, "Optimal VNFs placement in CDN Slicing over Multi-Cloud Environment", IEEE Journal on Selected Areas in Communications (JSAC), vol. PP, no. 99, March 2018.

[52] I. Benkacem, T. Taleb, M. Bagaa and H. Flinck, "Performance Benchmark of Transcoding as a Virtual Network Function in CDN as a Service Slicing" in Proceedings of 2018 IEEE Wireless Communications and Networking Conference (WCNC), Apr. 2018.

[53] G. Rubino, "Quantifying the Quality of Audio and Video Transmissions over the Internet: The PSQA Approach", in Communication Networks & Computer Systems, J. A. Barria, Ed. Imperial College Press, 2005.

[54] L. Yala, P. A. Frangoudis and A. Ksentini, "QoE-Aware Computing Resource Allocation for CDN-as-a-Service Provision", in 2016 IEEE Global Communications Conference (GLOBECOM), Dec. 2016.

[55] W. Chérif, A. Ksentini, D. Négru and M. Sidibé, "A PSQA: PESQ like non-intrusive tool for QoE prediction in VoIP services", in Proceedings of IEEE International Conference on Communications (ICC), Jun. 2012, pp. 2124-2128.

[56] W. Chérif, A. Ksentini, D. Négru and M. Sidibé, "A PSQA: Efficient real-time video streaming QoE tool in a future media internet context" in Proceedings of the 2011 IEEE International Conference on Multimedia and Expo (ICME 2011), Jul. 2011, pp. 1-6.

[57] 5G!Pagoda, "D4.1: Scalability-driven management system", https://5g-pagoda.aalto.fi/assets/demo/attachement/delivrables/D4.1_Scalability-driven_management_system_v1.0.pdf, Feb. 2018.