DEDICAT 6G: Dynamic coverage Extension and Distributed Intelligence for human Centric Applications with assured security, privacy and Trust: from 5G to 6G

Deliverable D3.2

Second release of mechanisms for dynamic distribution of intelligence

Project Details

Call	H2020-ICT-52-2020
Type of Action	RIA
Project start date	01/01/2021
Duration	36 months
GA No	101016499

Deliverable Details

Deliverable WP:	WP3 (Mechanisms for supporting dynamic distribution of intelligence)
Deliverable Task:	Task T3.1 (Architectural techniques for supporting dynamic, optimal placement of intelligence) and T3.2 (Intelligence placement optimization)
Deliverable Identifier:	DEDICAT6G_D3.2
Deliverable Title:	Second release of mechanisms for dynamic distribution of intelligence
Editor(s):	Y. Carlinet (Orange)
Author(s):	A. Anttonen (VTT), Y. Carlinet (Orange), P. Demestichas (WINGS), R. Eyckermen (IMEC), G. Iecker Ricardo (Orange), M. Forsell (VTT), V. Lamprousi (WINGS), J. Moreno (ATOS), K. Mößner (TUC), S. Penjivrag (VLF), N. Perrot (Orange), P. Reiter (IMEC), H. Resende (IMEC), D. Ribar (Airbus), C. Silva (IMEC), J. Renart (ATOS), V. Stavroulaki (WINGS)
Reviewer(s):	A. Anttonen (VTT), J. Moreno (ATOS)
Contractual Date of Delivery:	30/09/2022
Submission Date:	30/09/2022
Dissemination Level:	PU
Status:	Final
Version:	v1.0
File Name:	DEDICAT6G_D3.2 Mechanisms for dynamic distribution of Intelligence_v1.0.doc

Deliverable History

Version	**Date**	**Modification**
V1.0	*30/9/2022*	*Final version, submitted to EC through SyGMa*

Table of Contents

List of Acronyms and Abbreviations

Acronym/Abbreviation	Definition
AAA	Authentication Authorization Accounting
ACL	Access Control List
AGV	Automated Guided Vehicle
AI	Artificial Intelligence
AMF	Access Mobility Function
AP	Access Point
API	Application Programming Interface
AR	Augmented Reality
B5G	Beyond 5G
BLE	Bluetooth Low Energy
BLEMAT	Bluetooth Low Energy Micro-location Asset Tracking
BS	Base Station
CFS	Customer Facing Service
CISC	Complex Instruction Set Computer
CLI	Command-Line Interface
CPU	Central Processing Unit
C-RAN	Cloud Radio Access Network
CU	Control Unit
D2D	Device-to-Device
DA	Distributed Agents
DDoS	Distributed Denial of Service
DNS	Domain Name Service
DoI	Distribution of Intelligence
DoS	Denial of Service
DU	Distributed Unit
DVFS	Dynamic Voltage and Frequency Scaling
E2E	End-to-End
EC	Edge Computing
EMS	(Network) Element Manager System
EN	Edge Node
eNB	e(volved) NodeB (a.k.a. E-UTRAN NodeB)
ESM	Emulated Shared Memory
ETSI	European Telecommunications Standards Institute
FC	Functional Component
FCAPS	Fault/Configuration/Audit/Performance/Security
FE	Functional Entity

FG	Functional Group
FL	Federated Learning
FLOPS	FLoating OPeration per Second
GDPR	General Data Protection Regulation
GKE	Google Kubernetes Engine
gNB	(next)g(eneration)NodeB *(replaces 4G eNB)*
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
GPU	Graphical Processing Unit
GUI	Graphical User Interface
HE	Hosting Entity
HMI	Human Machine Interface
IDaaS	Intelligence Distribution as a Service
IDDM	Intelligence Distribution Decision Making
IEEE	Institute of Electrical and Electronics Engineers
IMS	IP Multimedia Sub-system
IoT	Internet of Things
IoV	Internet of Vehicle
ISG	Industry Specification Group
JSON	Java-Script Object Notation
K3S	Lightweight Kubernetes
K8S	Kubernetes
KPI	Key Performance Indicator
LAN	Local Area Network
LCM	Life Cycle Management
LDM	Local Dynamic Map
LOS	Line of Sight
LTE	Long-Term-Evolution
MAC	Medium Access Control
MANO	Management Network Orchestration
MAP	Mobile Access Point
MC-PTT	Mission Critical Push-To-Talk
MCS	Mission Critical Service
MCV	Manned Connected Car
MCX	Mission Critical {PTT, Video, Data Services}
MEC	Mobile Edge Computing
MEP	Multi-Access Edge Computing Platform
MEPM	Multi-Access Edge Computing Platform Manager
MIMD	Multiple Instruction Multiple Data

ML	Machine Learning
MME	Mobility Management Entity
MOO	Multi-Objective Optimization
MORL	Multi-Objective Reinforcement Learning
NFV	Network Virtualization Function
NFV-I	NFV Infrastructure
NFV-O	NFV Orchestrator
NG-RAN	Next Generation RAN
NLOS	Non-Line of Sight
NP	Non-Polynomial
NSSF	Network Slice Selection Function
OS	Operating System
OSM	Open Source MANO
PCF	Policy Control Function
PDU	Protocol Data Unit
PFCP	Packet Forwarding Control Packet
PoP	Point of Presence
PPDR	Public Protection and Disaster Relief
PST	Privacy, Security & Trust
QCI	QoS Class Identifier
QoE	Quality of Experience
QoS	Quality of Service
RAM	Random Access Memory
RAN	Radio Access Network
RAT	Radio Access Technology
REST	Representation State Transfer
RF	Radio Frequency
RISC	Reduced Instruction Set Computer
RKE	Rancher Kubernetes Engine
RLC	Radio Link Control
RPC	Remote Procedure Call
RRC	Radio Resource Control
RSS	RDF Site Summary
RSU	Road Side Unit
RTT	Round Trip Time
SLA	Service Level Agreement
SMF	Session Mobility Function
SNR	Signal-to-Noise Ratio
SOTA	State Of The Art

SPP	Security and Privacy Protection
SSL	Secured Socket Layer
TCF	Thick Control Flow
ToC	Table of Content
TPU	Tensor processing unit
TSL	Transport Layer Security
UAV	Unmanned Aerial Vehicle
UC	Use-Case
UE	User Equipment (e.g., mobile phone)
UML	Unified Modelling Language
UPF	User Plane Function
URLLC	Ultra-Reliable Low Latency Communication
V2X	Vehicle to X
V2V	Vehicle to Vehicle
VANET	Vehicular Ad-hoc Networks
VEC	Virtual Environment Control
VIM	Virtual Infrastructure Manager
VM	Virtual Machine
VN	Vehicular Node
VNF	Virtual Network Function
vRAN	Virtual Radio Access Network
VRU	Vulnerable Road User
WMS	Warehouse Management System

List of Figures

List of Tables

Executive Summary

The envisioned requirements for 6G include, among others, lower environment impact, imperceptible end-to-end latency, high resiliency, security and privacy. They can only be achieved by combining both computation and communications capabilities of the whole network: core, edge, access, and even terminals. However, in 5G, it is lacking a unified platform that can leverage infrastructure programmability and AI techniques to meet the most stringent requirements of services.

The specific goals of WP3 (Mechanisms for supporting dynamic distribution of intelligence) include (T3.1) architectural techniques for supporting offloading, migration and distribution of computing and communication on processor, storage, and network levels, (T3.2) algorithms for migration and distribution of intelligence, and (T3.3) validation of the mechanisms for computation placement optimization also related to the high-level architecture and scenarios/use cases defined in WP2 and carried out in WP6, respectively.

This document is the second instalment of a series of three, describing the achievements in WP3. These achievements follow three dimensions:

Algorithms for Distribution of Intelligence (section 2)

The project has designed and implemented algorithms for the placement of intelligence (i.e. functional entities) while optimizing a set of given KPIs (including end-to-end latency, overall energy consumption, throughput, service reliability), in the particular cases of the use-cases defined in the project.

Experimentations were conducted in order to assess the relevance and performance of the proposed approaches. The results that were collected are presented in this deliverable. The results indicate that significant performance improvements can be achieved, as compared to state-of-the-art techniques.

Low-level Architectural techniques for Distribution of Intelligence (section 3)

We have studied architectural techniques for context switching, patterns of computation and communication, load balancing, movement of threads, reducing the state of computation, synchronization, programmability and placement of functionality. Early experimentations were conducted at the processor and server level.

Security and Trust (section 4)

The DEDICAT 6G security and privacy protection framework is based on a decentralized, blockchain powered data marketplace. It enables secure, automated monetization, processing and exchange of IoT sensors and digital assets data with technical and policy-based data verification. A description of this framework is given in section 4.

1 Introduction

The 6G network is expected to be deployed in the 2030s. Designing a communication technology for the 2030s relies on:

1. understanding future service needs on the 2030 horizon and beyond
2. investigating techniques improving performance versus the state of the art
3. combining different techniques to build a mobile communication system that addresses the identified needs and constraints.

The first two steps generally feed each other: new service requirements stimulate research while increased performance inspires new services.

The third step will start when there will be a sufficient understanding of the target services and technical capabilities to set initial design objectives or requirements. Consensus building will culminate with standardization, which will specify service requirements, architectures, interfaces, and protocols that should be addressed globally. Indeed, a common global standard will be key to enable affordable costs via economies of scale, interoperability and international roaming.

Network virtualization and slicing technologies enable service providers to have access to dedicated ISP's computational, storage, and communication resources. Cellular networks will integrate enterprise local networks, which, in turn, has the potential to increase even further the number of devices and networks connected to the internet.

In a Cloud Computing (CC) architecture, this explosion of the number of connected devices results in an overload of ISPs and Content Providers' resources. Consequently, users' Quality of Experience (QoE) may be drastically affected, and more demanding applications and services may even become invalidated. It has been extensively discussed in the literature that such an increasing demand may be addressed by shifting from the classic CC paradigm to the Edge Computing (EC) paradigm.

EC is a computing and data distribution paradigm in which computation and data storage are placed closer to the service end user. In a general EC model (cf. Figure 1), network nodes are split into three regions: (i) users (data sources), (ii) edge servers (access points and gateways), and (iii) edge cloud. Any network node of any region can host applications and the connection between user and application host is determined accordingly. By doing so, you offload the general resources utilization (bandwidth and response times (also call delay) and enable a more flexible range of applications. Specifically, because computations can be carried out at the edge, closer to where data is produced, response times have potential to be drastically reduced, which is particularly relevant to time-sensitive applications.

EC was introduced in 5G networks, however the control framework in 5G is not mature enough to support Intelligence Distribution as defined in the Dedicat-6G project, in Deliverable D3.1.

The general objective of WP3 is to support dynamic, optimal placement of intelligence (data, computation, storage) in heterogeneous B5G/6G networks with respect to Key Performance Indicators e.g., service creation time, latency and reliability, overall energy consumption and security.

The main incentive is to enable reliable service continuity with the target user mobility and given network and computation resources for the DEDICAT 6G use cases and system architecture defined in WP2.

Techniques for dynamic distribution of data, computation, and storage in B5G/6G networks are developed, and algorithms for the overall optimization task are formulated to support extended dynamic coverage (WP4) and services of WP6. Understanding the trade-offs between computation and communications is essential in the underlying concepts. While the developed solutions are initially evaluated and tested in this work package, their practical aspects are integrated in the use case pilots in WP6.

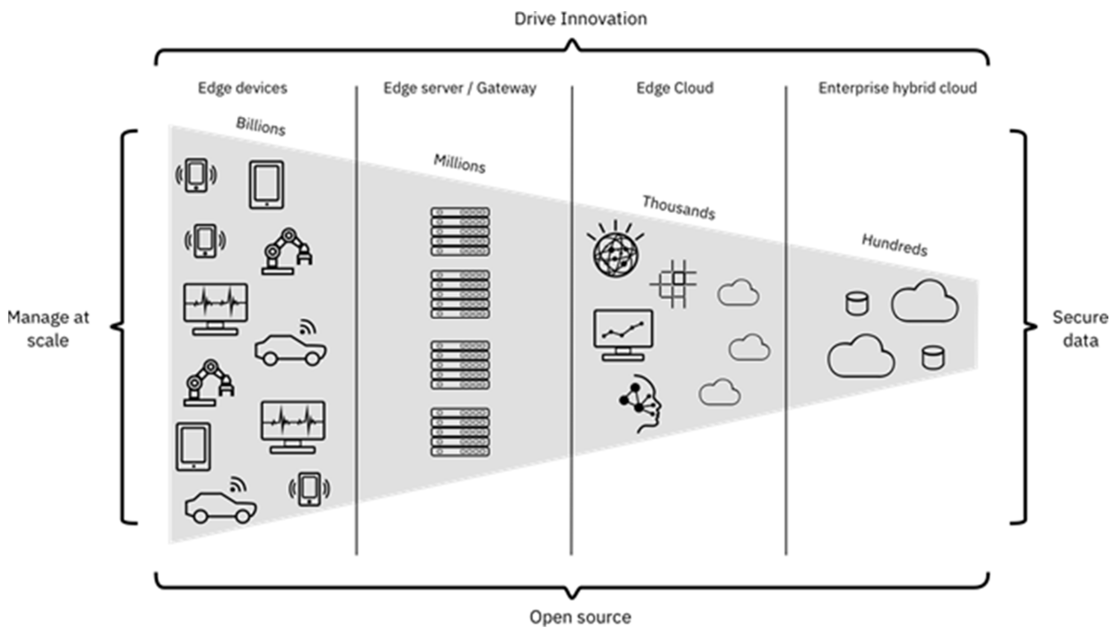


Figure 1. Edge Computing paradigm

1.1 Scope

The scope of this deliverable is to describe the achievements so far on the Mechanisms for Dynamic Distribution of Intelligence. The work described in this document is still in progress as the final outcomes of the work will be detailed in the next and last upcoming deliverable. This document is the second iteration out of the three planned over a period of 18 months.

1.2 Document Structure

The overview of the underlying architecture of the Dedicat-6G system is described in D2.2 [1], with a focus on the Functional Groups used in WP3 in D3.1 [6]. As a consequence, they are not repeated in this deliverable.

The state-of-art covering Intelligence Distribution Algorithms, technologies and frameworks for Distribution of Intelligence in Edge Computing, and orchestration of NFV (Network Function Virtualization) is provided in D3.1 [6].

Section 2 describes the algorithms for Distribution of Intelligence, first in a general case, and also in the particular instances of the use-cases defined and implemented in the project.

Section 3 details a set of low-level architectural techniques for improving the performance of Distribution of Intelligence.

Section 4 deals with security and trust issues.

Finally, the conclusion (Section 5) summarizes the achievements and describes the next steps to be carried out in WP3.

2 Algorithms for Distribution of Intelligence

Algorithms described in this section are designed to be executed in the Intelligence Distribution Functional Group and serve as a reference for the Context-Aware and Decision-Making components. These algorithms are key enablers for the objectives of meeting the specific requirements of verticals while lowering energy consumption.

2.1 Placement of Intelligence

2.1.1 Introduction

Intelligence placement (computation and content) intended for B5G/6G networks needs to take account of the increasing service requirements as well as the demand of power and delay sensitivity of user devices. In this section it is proposed an intelligence functions placement algorithm for dynamically distributing the functionalities to the various network nodes as part of intelligence distribution decision-making FC. This algorithm will support these kinds of systems by overcoming possible increase of network latency or possible unavailability of used edge nodes and more. It is provided a Mixed Integer Programming formulation of the problem and the proposed metaheuristic algorithm building upon the Genetic algorithm paradigm for solving it.

2.1.2 Updated Problem Statement/Formulation

In this subsection an updated formulation of intelligence functions placement problem described in [5] is provided. It is assumed a set of n Functional Entities (FEs) $F = \{F_1, \ldots, F_i, \ldots, F_n\}$ e.g., tasks, jobs, services with specific CPU requirements, $cpuFE_i$, and memory requirements, $memFE_i$. The possible communications between FEs are represented by a functional graph (Directed Acyclic Graph-DAG), denoted by $G_f = (\mathrm{F}, \mathrm{K})$. Each node F corresponds to a FE and each edge K connects interacting FEs and it is weighted, $k_{i,i'}$, according to the amount of data transferred between FEs F_i and $F_{i'}$ ($k_{i,i'} = 0$ when FEs F_i and $F_{i'}$ do not interact or when $i = i'$). Additionally, each edge has a maximum acceptable transmission delay, $l_{i,i'}$ (threshold).

Moreover, it is assumed a set of m Hosting Entities (HEs) $H = \{H_1, \ldots, H_j, \ldots, H_m\}$ e.g., edge nodes, core nodes, robotic units, end user devices, Virtual Machines, containers, with some capabilities. These are the maximum available CPU resources, $cpuHE_j$, memory resources, $memHE_j$, the battery level b_j if applicable and the functionality-wise E_j, which is a set that consists of FEs that the HE H_j can support ($E_j \subseteq F$) e.g., a robotic unit can support object recognition if camera is available, but cannot support grasping an object if robotic arm is not available. It is also considered a system layout graph $G_H = (\mathrm{H}, \mathrm{L})$ consisting of the available HEs and the communicational channels L among them. The communicational channel between the HEs H_j, $H_{j'}$, has a maximum link capacity $cap_{j,j'}$.

The objective is the allocation of FEs to HEs by ensuring efficiency of the system with low energy consumption and latency. Let A_j, denote the set of FEs that will be assigned to HE H_j, $A_j \subseteq F$. We are looking for the minimum cost allocation that satisfies a set of performance constraints.

For formulating reasons, we indicate the feasibility of assigning a FE to a HE in terms of functionality-wise, with the use of the constants $w_{i,j}$, where $w_{i,j} = 1$, if the HE H_j can support the FE F_i ($F_i \subseteq E_j$), and $w_{i,j} = 0$, otherwise. We assume that all FEs can be assigned to at least one HE in terms of functionality-wise ($\sum_{j=1}^{m} w_{i,j} \geq 1, \forall\ i \in \{1, \ldots, n\}$).

Furthermore, we introduce the set of decision variables y_j, where $y_j = 1$, if H_j is utilized (there is at least one FE assigned to it), and $y_j = 0$, otherwise. Also, we introduce the set of decision variables $x_{i,j}$ to describe the allocation of FEs to HEs. In particular, $x_{i,j} = 1$, if F_i is assigned to H_j and $x_{i,j} = 0$, otherwise. Finally, we define the set of decision variables $z_{i,i',j,j'}$, to describe the communication among HE. In particular, $z_{i,i',j,j'} = 1$, if H_j and $H_{j'}$ are communicating due to communicating FEs F_i and $F_{i'}$ assigned to them respectively, and $z_{i,i',j,j'} = 0$, if H_j and $H_{j'}$ are not communicating as no communicating FEs have been assigned to them.

The problem of obtaining A_j may be reduced to the following problem where the following Objective Function (OF) is minimized:

$$\min_{x,y,z} weight_1 \left(\sum_{j=1}^{m} y_j\, b_j \right) + weight_2 \cdot \left(\sum_{j=1}^{m} \left(W_{max}^{j} - W_{idle}^{j}\right) \cdot \frac{\sum_{i=1}^{n} cpuFE_i \cdot x_{i,j}}{cpuHE_j} + W_{idle}^{j} \right) + weight_3 \cdot \left(\sum_{j=1}^{m} \sum_{j'=1, j' \neq j}^{m} \sum_{i=1}^{n} \sum_{i'=1, i' \neq i}^{n} \left(z_{i,i',j,j'} \cdot \frac{k_{i,i'}}{cap_{j,j'}} \right) \right)$$

Subject to:

$\sum_{j=1}^{m} x_{i,j} = 1$, $\forall\, i = \{1, \ldots, n\}$, each FE can be allocated to only one HE,

$\sum_{i=1}^{n} [x_{i,j} * cpuFE_i] \leq cpuHE_j \text{ and } \sum_{i=1}^{n} [x_{i,j} * memFE_i] \leq memHE_j \;\; \forall\, j = \{1, \ldots, m\}$, the maximum available resources of the HEs are respected.

$\frac{\sum_{i=1}^{n} x_{i,j}}{n} \leq y_j$,$\forall\; j = \{1, \ldots, m\}$, all HE that are utilized ($y_j = 1$), have at least one FE assigned on them.

$x_{i,j} \leq w_{i,j}$,$\forall\, i = \{1, \ldots, n\}, j = \{1, \ldots, m\}$, the feasibility of assigning a FE to a HE ($x_{i,j} = 1$) in terms of functionality-wise is respected.

$z_{i,i',j,j'} \cdot \frac{k_{i,i'}}{cap_{j,j'}} \leq l_{i,i'}$, $\forall\; i, i' \in \{1, \ldots, n\},\; j, j' \in 1, \ldots, m$, where $i \neq i', j \neq j'$, the maximum transmission delay among two communicating HEs is respected.

$z_{i,i',j,j'} \geq x_{i,j} + x_{i',j'} - 1$, $\forall\;\; i, i' \in \{1, \ldots, n\},\; j, j' \in \{1, \ldots, m\}$, where $i \neq i'$, $j \neq j'$, $k_{i,i'} \neq 0$, the communicating HEs should have communicating FEs assigned to them.

The first term of the OF denotes the cost related to the battery level (if applicable) of utilized HEs. This cost takes higher values when battery is low, and it takes close to zero values when it is fully charged or when the HE is not battery-powered. The second term denotes the power consumption cost which is modelled as $\left(W_{max}^{j} - W_{idle}^{j}\right) \cdot Ucpu_j + W_{idle}^{j}$ based on [2], where $Ucpu_j$ is the CPU utilization rate on the HE H_j and W_{max}^{j}, W_{idle}^{j} are the power consumption when the HE H_j is fully loaded and idle, respectively. Finally, the third term denotes cost (transmission delay) imposed by the communication among HEs related to the amount of data transferred between FEs $k_{i,i'}$ and maximum capacity link $cap_{j,j'}$. Each term is normalized and is weighted depending on the use case. All notations can be found in Table 1.

Table 1. Intelligence functions placement notations.

Notation	Definition	Notation	Definition
n, m	Total number of FEs and HEs respectively	$G_H = (H, L)$	System layout graph with nodes HEs and computational channels L
i, i'	Indexes of FEs	$l_{i,i'}$	Maximum acceptable transmission delay between FEs F_i and $F_{i'}$
j, j'	Indexes of HEs	$cap_{j,j'}$	Maximum capacity of the links among HEs H_j and $H_{j'}$
F ={F_1, …, F_i, …, F_n}	Set of FEs	W^j_{max}, W^j_{idle}	Power consumption when HE H_j is fully loaded and idle, respectively
H ={H_1, …, H_j, …, H_m}	Set of HEs.	$w_{i,j}$	Binary constant showing if FE F_i can be assigned to HE H_j in terms of functionality-wise
$cpuFE_i$, $memFE_i$	CPU and memory requirements of FE F_i	A={A_1, …, A_j, …, A_m}	Collection of sets of FEs that will be assigned to the HEs
$cpuHE_j$, $memHE_j$	Maximum available CPU and memory resources of HE H_j	y_j	Decision variable that takes 1(0) depending on whether HE H_j is (is not) utilized
b_j	Battery level of HE H_j	$x_{i,j}$	Decision variable that takes 1(0) depending on whether FE F_i is (is not) assigned to HE H_j
$G_f = (F, K)$	Functional graph with nodes FEs and edges K	$z_{i,i',j,j'}$	Decision variable that takes 1(0) depending on whether HE H_j and $H_{j'}$ are (are not) communicating
$k_{i,i'}$	Weights of interacting FEs F_i and $F_{i'}$ (data transferred)		

2.1.3 Solution Approach

The above problem was initially solved with the use of a Mixed Integer Programming (MIP) python solver called GNU Linear Programming Kit (GLPK) provided by the open-source PuLP [7]. MIP solvers are known to provide the optimal solution but are computationally intractable, especially for large scale experimentation. For this reason, a metaheuristic Genetic algorithm is developed to approximate this solution. There are many studies proposing genetic algorithms for service and virtual machine placement problems providing good results [2][3][4].

In general, when using genetic algorithms to address an optimization problem, it is considered a population of individuals, known as "chromosomes", to encode a solution of a problem each. The chromosome, in turn, is a series of predetermined number of "genes", and each "gene" stands for a parameter that defines the solution for that individual. In our case, each "chromosome" is a series of HEs where each one represents the "proposed" HE for each FE and the length of each "chromosome" equals to the number of FEs. The algorithm firstly generates randomly a population of "chromosomes" and then it is applied a fitness function, which is the objective function, OF, of the optimization problem, to each chromosome. Each chromosome has a fitness score calculated from the OF. Over the course of a defined number of generations (in our case we use an upper limit of generations having the same best fitness score), a population of chromosomes evolves, and some operators (parent selection, crossover, mutation) are used to improve the population's overall fitness.

Parent selection operator is the process of selecting chromosomes in one generation to pass them to the next generation, these chromosomes are known as "parents". In our case we used tournament selection where each "parent" is the fittest out of a predetermined number of randomly chosen chromosomes of the population. Crossover operator is used for creating two "children" candidate solutions (new solutions) from two "parents". In our case we select a random split point on the chromosome of each "parent", then we create a "child" with the genes up to the split point from the first "parent" and from the split point to the end of the chromosome from the second "parent". This process is then inverted for the second "child". Finally, mutation is a change in a single gene of "child's" chromosome or in a group of genes for exploring new areas of the solution space. In our case we use reverse sequence mutation proposed in [5]. We randomly choose two positions in chromosome, and we reverse the gene order in the sequence between these positions. Crossover and mutation operators occur with a predefined probability. In our case we used 0.8 crossover and 0.15 mutation rate. Additional steps were introduced related to efficient initialization of chromosomes among others and more elaboration will follow.

2.1.4 Preliminary Results

In this subsection some initial results of the described model are presented. Figure 2 depicts a case of thirteen HEs of which nine are robotic nodes having camera or arm on them, with battery level of range 20% - 90%, and the rest are not battery-powered nodes.

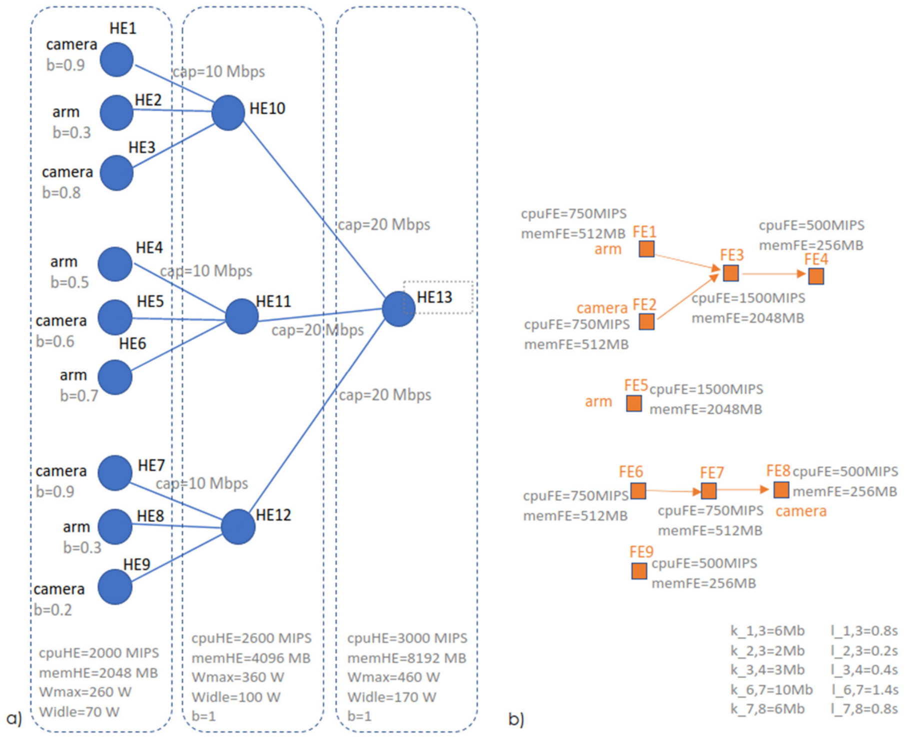
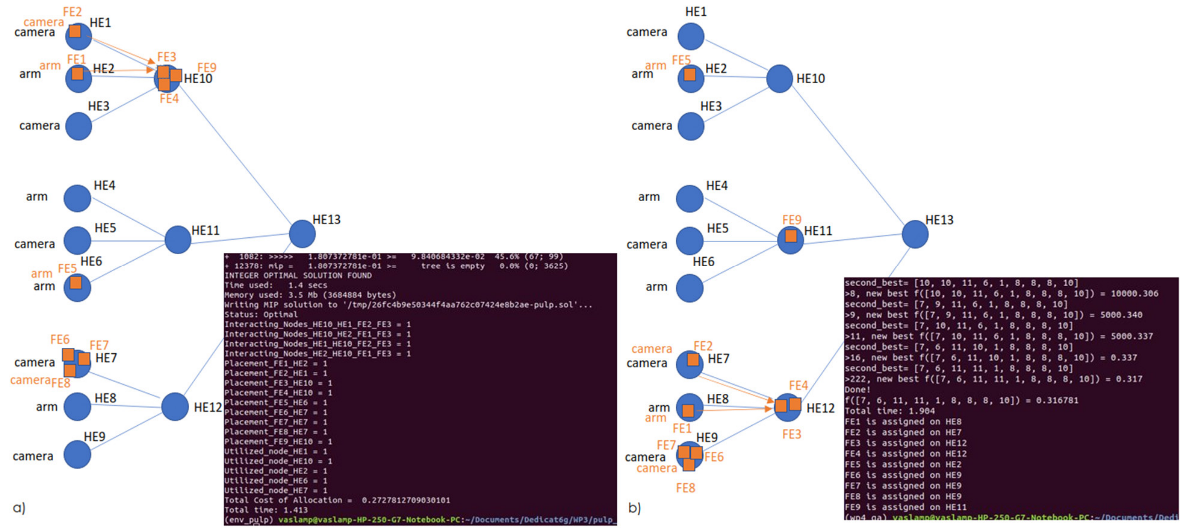


Figure 2. Example input to the algorithm: a) the schema on the left presents the HE's graph with their capabilities, b) the schema on the right presents the FE's graph with their requirements.

The levels of available CPUs are {2000, 2600, 3000} MIPS, the levels of available memory are {2048, 4096, 8192} MB, the levels of power consumption when fully loaded are {260, 360, 460} W and the levels of power consumption when idle are {70, 100, 170} W. The links between HEs have maximum capacity 10 or 20 Mbps. Figure 2b shows the nine FEs with their requirements. FE1 and FE5 can be placed in a robotic unit with an arm, FE2 and FE8 can be placed in a robotic unit with a camera and the rest can be placed anywhere fulfilling the CPU and memory requirements of levels {500,750,1500} MIPS and {256,512,2048} MB, respectively, considering the amount of data transferred 2-10 Mb and the maximum acceptable delay 0.2-1.4 s between two interacting FEs. In this example/experiment, we used a population of 40 chromosomes with maximum iterations having the same score (stopping criteria) being 400 for genetic algorithm. Subsequently, Figure 2 shows the output placement proposed by GLPK solver and Genetic Algorithm model. As it is shown Genetic Algorithm has a score (0.316781) close to the optimum (0.272781) with one extra HE utilized and similar category of nodes used in general.



```
+  1082: >>>>>   1.807372781e-01 >=   9.840684332e-02  45.6% (67; 99)
+ 12378: mip =   1.807372781e-01 >=     tree is empty   0.0% (0; 3625)
INTEGER OPTIMAL SOLUTION FOUND
Time used:   1.4 secs
Memory used: 3.5 Mb (3684884 bytes)
Writing MIP solution to '/tmp/26fc4b9e50344f4aa762c07424e8b2ae-pulp.sol'...
Status: Optimal
Interacting_Nodes_HE10_HE1_FE2_FE3 = 1
Interacting_Nodes_HE10_HE2_FE1_FE3 = 1
Interacting_Nodes_HE1_HE10_FE2_FE3 = 1
Interacting_Nodes_HE2_HE10_FE1_FE3 = 1
Placement_FE1_HE2 = 1
Placement_FE2_HE1 = 1
Placement_FE3_HE10 = 1
Placement_FE4_HE10 = 1
Placement_FE5_HE6 = 1
Placement_FE6_HE7 = 1
Placement_FE7_HE7 = 1
Placement_FE8_HE7 = 1
Placement_FE9_HE10 = 1
Utilized_node_HE1 = 1
Utilized_node_HE10 = 1
Utilized_node_HE2 = 1
Utilized_node_HE6 = 1
Utilized_node_HE7 = 1
Total Cost of Allocation =  0.2727812709030101
Total time: 1.413
(env_pulp) vaslamp@vaslamp-HP-250-G7-Notebook-PC:~/Documents/Dedicat6g/WP3/pulp_
```

```
second_best= [10, 10, 11, 6, 1, 8, 8, 8, 10]
>8, new best f([10, 10, 11, 6, 1, 8, 8, 8, 10]) = 10000.306
second_best= [7, 9, 11, 6, 1, 8, 8, 8, 10]
>9, new best f([7, 9, 11, 6, 1, 8, 8, 8, 10]) = 5000.340
second_best= [7, 10, 11, 6, 1, 8, 8, 8, 10]
>11, new best f([7, 10, 11, 6, 1, 8, 8, 8, 10]) = 5000.337
second_best= [7, 6, 11, 10, 1, 8, 8, 8, 10]
>16, new best f([7, 6, 11, 10, 1, 8, 8, 8, 10]) = 0.337
second_best= [7, 6, 11, 11, 1, 8, 8, 8, 10]
>222, new best f([7, 6, 11, 11, 1, 8, 8, 8, 10]) = 0.317
Done!
f([7, 6, 11, 11, 1, 8, 8, 8, 10]) = 0.316781
Total time: 1.904
FE1 is assigned on HE8
FE2 is assigned on HE7
FE3 is assigned on HE12
FE4 is assigned on HE12
FE5 is assigned on HE2
FE6 is assigned on HE9
FE7 is assigned on HE9
FE8 is assigned on HE9
FE9 is assigned on HE11
(wp4_ga) vaslamp@vaslamp-HP-250-G7-Notebook-PC:~/Documents/Dedi
```

Figure 3. Output of the example (terminal screenshot and schematical representation): a) on the left side it is the GLPK MIP model's (solver) output, b) on the right side it is the proposed Genetic Algorithm model's output.

Furthermore, Figure 3 shows some initial performance testing of Genetic algorithm compared to GLPK solver. For these plots we assume a fixed HE-schema, like the one shown in Figure 2a with the difference that the first category of HE (robotic units) consists of 36 HEs, the second category, consist of 6 HEs and the third category consists of one HE (43 HEs in total). We measured the scores and the execution time when increasing the FEs. The population of genetic algorithm is 60 for 2-16 FEs and 100 for 16-48 FEs and iteration threshold is set to 50 for 2 FEs, 100 for 4-20 FEs, 150 for 24-28 FEs and 200 for 32-48 FEs. As it is mentioned MIP solvers are computationally intractable in large experimentation, so we added an execution time limit of 200 s to MIP solver. As a result, the MIP scores appeared in Figure 4, exceeding 10 FEs, are not optimum since there was no time for all calculations to be completed. Additionally, we can see that there are not any MIP values for more than 36 FEs because within 200 s, GLPK solver could not find a feasible solution. As we can see from those graphs, Genetic algorithm has close to optimum scores within significantly less time than MIP model. Additionally, we can see that for a small amount of FEs (in this example less than 8 FEs), MIP solver is marginally faster than proposed Genetic algorithm, obtaining better scores. Hence, it may be preferable to use MIP model for small-scale problems and use the proposed Genetic algorithm model for large-scale problems.

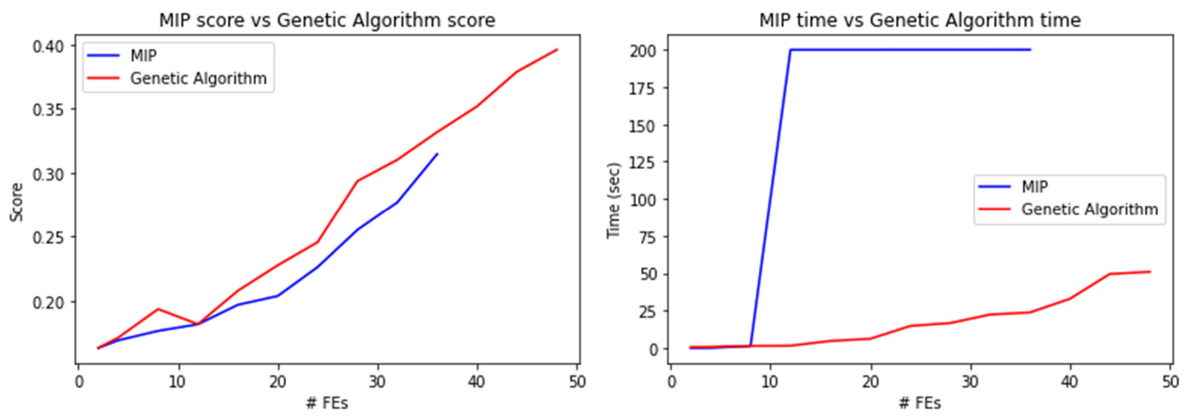


Figure 4. Score and time execution measurements of MIP and Genetic Algorithm models with increasing number of FEs.

2.1.5 Conclusion

In this subsection, the intelligence functions placement problem for distributing the intelligence/functionality to the various network nodes (edge, core nodes, robotic units etc.) is studied, taking into consideration among others the transmission delay and power consumption. It is provided the description and the formulation of the problem along with the description of the MIP and genetic algorithm implemented for approaching this problem. Additionally, preliminary results are provided. In future work, there will follow more evaluations and improvements of the proposed algorithm related to stopping criteria, population size, mutation probability, crossover probability and more.

2.2 Placement of Latency-Sensitive Tasks

In this Section, we consider the problem of finding the placement of the tasks, along with the needed routing between tasks and end-users, in the context of the Smart Warehouse Use-Case (UC 1).

2.2.1 Introduction

The performance of Edge Computing based systems depends on how efficiently the network resources are managed. In the Intelligence Distribution (ID) Problem, given a system (and its underlying network) with a fixed number of services to be consumed, we wish to (i) assign services to network nodes (service placement) and (ii) determine user-host communication path (service routing) in order to provide expected QoS to users at a minimum operation cost. In this section, we are interested in the ID problem for systems in which services are delay sensitive. Thus, in addition to finding an efficient service placement and routing, we want to ensure that services experience a tolerable end-to-end (E2E) delay.

In the context of Industry 4.0, Smart Warehouse applications bring together all the aforementioned challenges in an effort to promote digitization and automation of industrial processes. They have been considered a key use case in most next generation architectures envisioned designs, including the DEDICAT 6G project. In this work, we are particularly interested in real-time human-machine interaction services with strict latency requirements, e.g., automatically guided vehicles, timely computer-aided industrial operations (e.g., assisted with virtual and augmented reality technologies), etc. A MEC-powered network design is a key enabler of such services through back-end computation offload and opportunistic networking.

In this section, we approach the ID Problem by finding its optimal exact solution through a Branch-and-Cut algorithm. The separation problem is built with valid inequalities based on cover cuts (for capacity constraints). Additionally, we propose a greedy algorithm to efficiently approximate the problem with theoretical performance lower bounds.

2.2.2 System Model and Assumptions

Consider a MEC architecture, illustrated in Figure 1. The ***Devices*** layer comprises a set U of user equipment (UEs), each of which is associated with a single cellular Base Station (BS). Each BS is equipped with a MEC host for application-oriented data storage, processing, and routing, which we, henceforth, refer to as Points of Connection (PoCs) or "far-edge" hosts. For simplicity, we assume that PoCs are able to perfectly handle all transmissions from and to their associated UEs, so that UE-BS communication will be transparent to our model. In the transport network, BSs' traffic often converges at sink nodes/gateways powered with more abundant computational resources, which we refer to as "near-edge" MEC hosts. The ***Edge*** layer is composed of interconnected (e.g., through Mp3 interfaces) far- and near-edge MEC hosts, which we also refer to as the Edge network. We represent the set comprising all MEC hosts by H.

In our model, we do not make any distinction between applications and services and refer to them simply as ***tasks***. The set of all considered tasks is denoted by T. We assume that each MEC host has all tasks implemented a priori. We define this model in the context of the Decision-Making Functional Component (DMFC), as a centralized intelligence, with access to the information related to the entire network's infrastructure and available resources. The system operates periodically such that each period is split into two phases: observation and management. In the observation phase, UEs exchange their requested tasks' data and potentially place requests for more tasks, which will be served only in the observation phase of the next operation period. For the entire observation phase, UE-BS association remains static, and resources are dedicated to their assigned requests. In the management phase, the DMFC knows the set of all placed requests $\mathrm{R} \subseteq \mathrm{U} \times \mathrm{T}$ and decides how to manage the network's available resources in order to accommodate all requests.

In the management phase, depending on the availability of network and computational resources, any MEC host may be assigned by the DMFC to handle a request and to provide the related task's data to its UE, becoming the request's ***provider***. If the closest host to the UE, i.e., its associated PoC, is not available, then another MEC host may become the provider. In this case, the DMFC must also provide a ***route***, i.e., a sequence of links, connecting the request's provider and its UE. Additionally, the DMFC may also assign different priority levels, from a pre-determined set P, in order to force a higher transmission rate in exchange of more computational resources utilization. The resources and performance metrics related to the described operation are summarized as follows:

- When a request $r \in \mathrm{R}$ is assigned to a host $h \in \mathrm{H}$ at a priority $p \in \mathrm{P}$, it consumes a fraction D_p^r of host h's total available computational resources, C_h.
- The ***throughput*** T_p^r, i.e., the average achieved data transmission rate, to serve request r's data at priority p must be ensured by the provider host and each other host participating in the request's routing.

In the design of MEC systems that are able to meet strict latency requirements, we have to consider the trade-off between throughput and network latency. We assume error-free channels in the network, so the (average) throughput equals the data rate. Achieving higher throughput may provide UEs with smoother, uninterrupted experience, so we use it as a measure of Quality of Service (QoS). Moreover, we consider the network latency as the end-to-end delay, i.e., the time to transmit application packets from the UE to the host providing the requested task (or vice-versa). We assume that the end-to-end delay consists uniquely of queuing delay, which is fundamentally impacted by the total throughput. If, on one hand, we want to provide better QoS, on the other hand, it increases the network latency (queuing delay), becoming the bottleneck for timely applications.

In summary, the DMFC, aware of the request set R, must determine a *network setup*. We focus this Section on techniques to find such a network setup, which consists of:

(i) Priority assignment: Each request $r \in \mathrm{R}$ will be assigned a priority $p \in \mathrm{P}$. Higher priorities are associated to higher throughput and higher computation. We assume that the lowest priority is related to the minimum throughput to provide a task correctly.

(ii) Request allocation: Each request $r \in \mathrm{R}$ will be provided by an edge host $h \in \mathrm{H}$. Hosts may have different amounts of available resources, e.g., near-edge hosts usually enjoy more computation power than far-edge hosts.

(iii) Request routing: For each request $r \in \mathrm{R}$, if it is provided by a host other than its UE's PoC, then the DMFC must find a route through which task-related data will flow between UE and the actual provider.

We illustrate the operation of the considered system in Figure 5. In this scheme, a set of requests $\mathrm{R} = \{A,B,\dots,G\}$ is placed by the UEs of PoCs 1 and 2. The DMFC collects the requests and try to assign them to the MEC hosts in order to provide UEs with the highest throughput possible. Consider that Far-Edge hosts can only provide tasks at the lowest priority. Then, the DMFC will try to distribute the tasks among the Near-Edge hosts, i.e., hosts 3, 4, and 5. In this example, due to computational limitations, Near-Edge hosts can only provide up to 2 tasks at intermediate priority and 1 task at high priority. Similarly, latency requirements are only satisfied if arcs carry data traffic of up to 2 tasks at intermediate priority and 1 task at high priority. A possible assignment is shown in the figure, where tasks are placed on each MEC host and each colour is associated to a different priority level. The coloured arrows indicate the tasks' data route, flowing from its provider to the PoC.

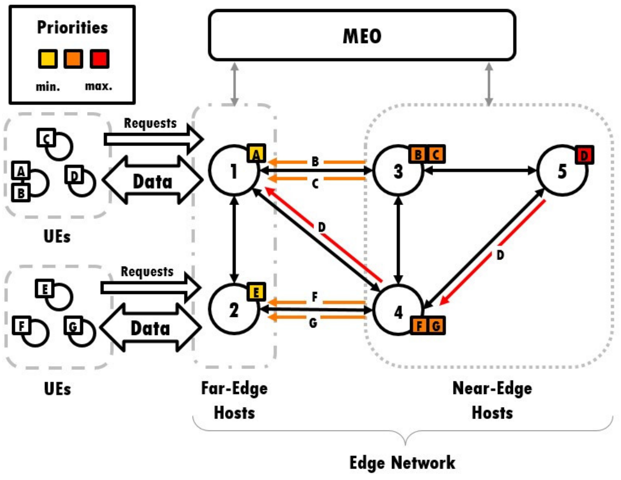
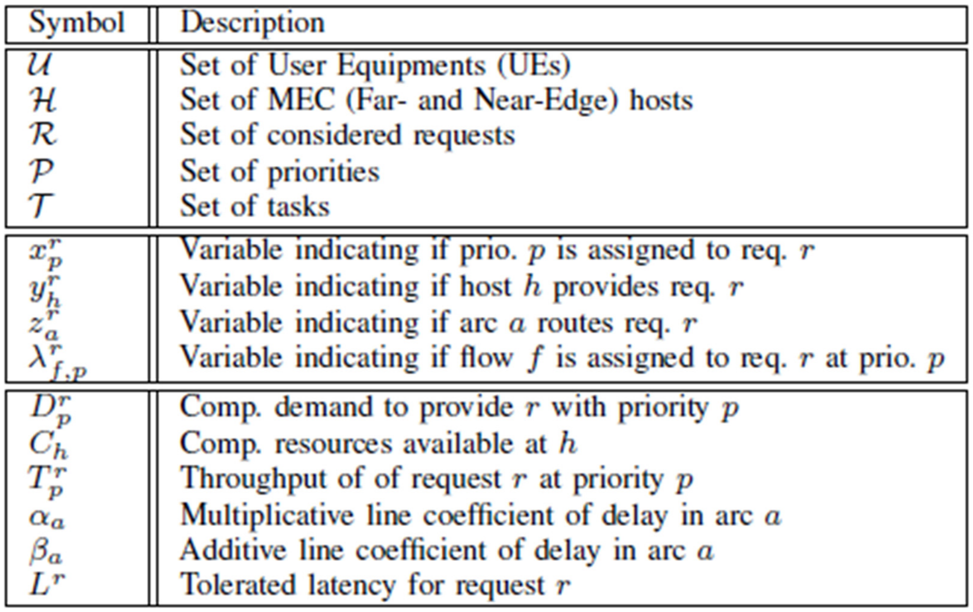


Figure 5. Operation Example: UEs place requests for tasks to their neighbouring PoCs, which are, in turn, forwarded to the DMFC. To each request, the DMFC assigns a provider host, a priority level, and a low-latency route between provider and PoC through the Edge network.

2.2.3 The Task Distribution Problem

Considering the system model and assumptions discussed previously, we introduce next the Task Distribution (TD) Problem and model it as an Integer Programming (IP) optimization problem.

Table 2. Summary of notations



Symbol	Description
$\mathcal{U}$	Set of User Equipments (UEs)
$\mathcal{H}$	Set of MEC (Far- and Near-Edge) hosts
$\mathcal{R}$	Set of considered requests
$\mathcal{P}$	Set of priorities
$\mathcal{T}$	Set of tasks
x_p^r	Variable indicating if prio. p is assigned to req. r
y_h^r	Variable indicating if host h provides req. r
z_a^r	Variable indicating if arc a routes req. r
$\lambda_{f,p}^r$	Variable indicating if flow f is assigned to req. r at prio. p
D_p^r	Comp. demand to provide r with priority p
C_h	Comp. resources available at h
T_p^r	Throughput of of request r at priority p
α_a	Multiplicative line coefficient of delay in arc a
β_a	Additive line coefficient of delay in arc a
L^r	Tolerated latency for request r

First, we represent the Edge network as a graph $G = (H,A)$, where the hosts compose the set of vertices and we consider a set of arcs A. If there is a communication link between two network hosts, we consider one arc in A in each direction. Note that arcs in G do not indicate transmission direction, they rather assist in the route creation process. Then, the route for serving request $r = (u,t) \in \mathrm{R}$ may be represented as a path on graph G from the UE to its provider host. Since we assumed that UE-BS is transparent to our model, it suffices to build a route from the UE's PoC to the request's provider. Therefore, from now on, we use index u to indicate the UE's PoC, i.e., $u \in \mathrm{H}$ is a (far-edge) MEC host.

In what follows, we introduce the elements of our optimization formulation.

1) Decision Variables: To each problem component described at the end of Section II, we associate a binary variable:

(i) We model the priority assignment with variable

$$x \in \mathbb{B}^{\mathcal{R}\, x\, \mathcal{P}} \qquad (1)$$

indicating whether priority $p \in \mathrm{P}$ is assigned to request$r \in \mathcal{R}$ (i.e., $x_p^r = 1$) or not (i.e., $x_p^r = 0$).

(ii) We model the request allocation with variable

$$y \in \mathbb{B}^{\mathcal{R}\, x\, \mathcal{H}} \qquad (2)$$

indicating whether request $r \in \mathrm{R}$ is provided by node $h \in \mathrm{H}$ (i.e., $y_h{}^r = 1$) or not (i.e., $y_h{}^r = 0$).

(iii) We model the request routing with variable

$$z \in \mathbb{B}^{\mathcal{R}\, x\, \mathcal{A}} \qquad (3)$$

indicating whether request $r \in \mathrm{R}$ data is forwarded through arc $a \in \mathrm{A}$ (i.e., $z_a^r = 1$) or not (i.e., $z_a^r = 0$).

Structural Constraints: First, to every request $r \in \mathrm{R}$, we must assign exactly one priority in P, i.e.,

$$\sum_{p \in \mathcal{P}} x_p^r = 1, \forall r \in \mathcal{R} \qquad (4)$$

Similarly, every request $r \in \mathrm{R}$ should be provided by one single node, i.e.,

$$\sum_{h \in \mathcal{H}} y_h^r = 1, \forall r \in \mathcal{R} \qquad (5)$$

Now, we define the constraints that will enforce the creation of flows (or paths) over the underlying graph *G = (H,A)* through which every request $r \in \mathrm{R}$ will be routed. This can be achieved by imposing the following set of *flow conservation* constraints

$$\sum_{h' \in \delta(h)} \left[z_{(h,h')}^r - z_{(h',h)}^r \right] = \mathbb{1}_{h=u(r)} - y_h^r, \forall r \in \mathcal{R}, \forall h \in \mathcal{H} \qquad (6)$$

where $u(r)$ is the PoC of the UE which placed request r and we define $\delta(h) \subseteq \mathrm{H}$ as the set of *neighbors* of host h, i.e., host h's communicating hosts, and 1_e indicate whether event e occurs (i.e., $1_e = 1$) or not (i.e., $1_e = 0$).

Computational Capacity Constraints: In what follows, we need to guarantee that a target setup can be implemented considering the available computational resources. We ensure that the *computational* utilization of each host meets its RAM/CPU capacity with

$$\sum_{r \in \mathcal{R}} \sum_{p \in \mathcal{P}} D_p^r \; x_p^r \, y_h^r \leq C_h, \forall h \in \mathcal{H} \qquad (7)$$

Where $D_p^r \in \mathbb{R}_+$ is the computational demand to provide the task associated to request r at priority p and $C_h \in \mathrm{R}_+$ is the total computational resources available at node h.

Latency Requirements Constraints: We model the latency requirements by imposing the following constraints

$$\sum_{\alpha \in \mathcal{A}} \left[\alpha_a \left(\sum_{r' \in \mathcal{R}} \sum_{p \in \mathcal{P}} T_p^{r'} \; x_p^{r'} \, z_a^{r'} \right) + \beta_a \right] z_a^r \leq L^r, \forall r \in \mathcal{R} \quad (8)$$

Where $T_p^r \in \mathbb{R}_+$ is the achieved throughput for request r at priority p and request r has tolerance $L^r \in \mathrm{R}_+$. We define the arc's *traffic load* as the total throughput achieved by all requests being routed through a. We approximate the network latency (i.e., the queuing delay) of a request r by a linear function of the traffic load on the links routing it, such that α_a and β_a are the line coefficients for arc a.[1] In summary, latency is impacted by two factors:

1) The number of links forwarding the request's data.
2) The total traffic load at those links, i.e., total throughput of all requests traversing each link.

Objective Function: Our goal is to maximize the QoS, which is simply the total throughput achieved in a given setup to serve all considered requests, i.e.,

[1] The system is stable if the total throughput over every link is bounded by the link's capacity [26]. We assume that all links have enough capacity to accommodate the maximum priority throughput.

$\text{QOS}(x) \triangleq \sum_{r \in \mathcal{R}} \sum_{p \in \mathcal{P}} T_p^r \; x_p^r$ (9)

Definition 1. *The Task Distribution (TD) Problem aims at finding a* valid *network setup, i.e., an assignment of* **x,y,z** *that satisfies constraints* (1)-(8)*, that maximizes the objective function* (9). *In other words, the TD Problem is to solve the following optimization problem:*

Problem 1 (Task Distribution (TD) Problem).

$$\begin{array}{lll} \textbf{(TD)} & \underset{\mathbf{x},\mathbf{y},\mathbf{z}}{\text{maximize}} & (9) \\ & \text{subject to} & (1)-(8) \end{array}$$

Proposition 1. *Problem 1 is* NP-*Hard.*

Proof. Consider the following special instance of Problem 1: Each request has sufficiently large latency tolerance so that all requests may share its route's arcs at the highest achievable throughput, i.e.,

$$L^r \geq \sum_{a \in \mathcal{A}} \alpha_a \; |\mathcal{R}| \, T_{max} \; + \; \beta_a \, , \; \forall r \; \in \mathcal{R}$$

Where $T_{\max} \geq T_p^r, \forall r \in \mathcal{R}, \forall p \in \mathcal{P}$. Notice that latency constraints can be relaxed and the DMFC is free to choose whatever route it judges to be convenient for each request. The DMFC still needs to assign each request to a host satisfying their computational capacity. Therefore, in this setup, the TD Problem can be translated to the classic Multiple Knapsack (MK) Problem, which is proven to be NP-Hard [28]. By reducing the MK Problem to this instance of the TD Problem, we show that, even in such a simple case, the TD Problem is NP-Hard. Therefore, the TD Problem is NP-Hard, so general instances cannot be solved in polynomial time, unless P = NP. □

Even though Problem 1 is NP-Hard, it can be linearized in order to be addressed via traditional IP solving techniques. This can be achieved by introducing auxiliary variables capturing the product of Problem 1's main variables.

Computational Capacity Constraints: We consider a first set of auxiliary variables

$u \in \mathbb{B}^{\mathcal{R} \, x \, \mathcal{H} \, x \, \mathcal{P}}$ (10)

such that computational capacity constraints (7) are written as

$\sum_{r \in \mathcal{R}} \sum_{p \in \mathcal{P}} D_p^r \; u_{h,p}^r \; \leq C_h \, , \; \forall h \; \in \mathcal{H}$ (11)

where $u_{h,p}^r \triangleq x_p^r y_h^r, \forall r \in \mathcal{R}, \forall h \in \mathcal{H}, \forall p \in \mathcal{P}$. We ensure that auxiliary variables **u** are coherently associated to the main variables **x,y** by enforcing the following set of constraints

$u_{h,p}^r \; \leq \; y_h^r \; , \forall r \; \in \mathcal{R}, \; \forall h \; \in \; \mathcal{H}, \; \forall p \; \in \; \mathcal{P}$

$u_{h,p}^r \; \leq \; x_p^r \; , \forall r \; \in \mathcal{R}, \; \forall h \; \in \; \mathcal{H}, \; \forall p \; \in \; \mathcal{P}$

$u_{h,p}^r \; \geq \; y_h^r \; + \; x_p^r \; - 1 \; , \forall r \; \in \mathcal{R}, \; \forall h \; \in \; \mathcal{H}, \; \forall p \; \in \; \mathcal{P}$ (12)

Latency Requirements Constraints: We consider a second set of auxiliary variables

$v \in \mathbb{B}^{\mathcal{R} x \mathcal{R} x \mathcal{A} x \mathcal{P}}$ (13)

such that latency requirement constraints are written as

$$,\sum_{a \in \mathcal{A}} \alpha_a \left(\sum_{r' \in \mathcal{R}} \sum_{p \in \mathcal{P}} T_p^{r'} \ v_{a,p}^{r,r'}\right) + \beta_a z_a^r \leq L^r \ , \forall r \in \mathcal{R} \qquad (14)$$

where $v_{a,p}^{r,r'} \triangleq x_p^{r'} z_a^{r'} z_a^r, \forall r, r' \in \mathcal{R}, \forall a \in \mathcal{A}, \forall p \in \mathcal{P}$. We ensure that auxiliary variables $\mathbf{v}$ are coherently associated to the main variables $\mathbf{x}$,$\mathbf{z}$ by enforcing the following set of

constraints

$v_{a,p}^{r,r'} \leq x_p^{r'}$, $\forall r, r' \in \mathcal{R}, \forall a \in \mathcal{A}, \forall p \in \mathcal{P}$

$v_{a,p}^{r,r'} \leq z_a^{r'}$, $\forall r, r' \in \mathcal{R}, \forall a \in \mathcal{A}, \forall p \in \mathcal{P}$

$v_{a,p}^{r,r'} \leq z_a^{r}$, $\forall r, r' \in \mathcal{R}, \forall a \in \mathcal{A}, \forall p \in \mathcal{P}$

$v_{a,p}^{r,r'} \geq x_p^{r'} + z_a^{r'} + z_a^r - 2$, $\forall r, r' \in \mathcal{R}, \forall a \in \mathcal{A}, \forall p \in \mathcal{P}$ (15)

The final formulation resulting from the linearization of Problem 1 is provided in Problem 2.

Problem 2 (Linear TD (LinTD) Problem). (LinTD)

$$\text{(LinTD)} \quad \underset{\mathbf{x},\mathbf{y},\mathbf{z},\mathbf{u},\mathbf{v}}{\text{maximize}} \quad (9)$$
$$\text{subject to} \quad (1)-(6),\ (10)-(15)$$

Remark 1. We note that we can provide an upper bound to Problem 1 if we solve the continuous relaxation of its linear version, i.e., relaxing the integrality of variables $\mathbf{x}$,$\mathbf{y}$,$\mathbf{z}$. Then, Problem 1 can be translated to a Linear Programming (LP) problem, which can be efficiently solved in practice. Nevertheless, it is important to emphasize that Problem 2's size grows quickly, as each new auxiliary variable introduces at least three new constraints.

2.2.4 Approximating the TD Problem

In this section, we discuss an approximation of Problem 1 based on *Decomposition*. In the Decomposition technique, we tackle the original problem using a two-stage approach. In the first stage, we solve an auxiliary problem which is simpler and captures only part of the original problem's constraints. The solution of the auxiliary problem is somehow used as an input to the master problem at the second stage. With the relaxed constraints related to the auxiliary problem, the master problem will find the best solution that suits its input.

In order to discuss how we can apply the decomposition technique to solve Problem 1, we need to introduce some additional notation. First, we define a *flow* $f_{s,d} = \{(s,h),...,(h',d)\}$ simply as a path on G, i.e., a sequence of arcs from a source node $s \in \mathrm{H}$ to a destination node $d \in \mathrm{H}$. Since multiple flows may exist connecting a pair of nodes s,d on G, we denote by $F_{s,d} > 0$ the total number of (s,d) flows. We denote the set of all flows in a graph G as

$\mathcal{F} \triangleq \{f_{s,d}(k) : k = 1, \ldots, F_{s,d}, \forall s, d \in \mathcal{H}\}$ (16)

where k is a unique index for each (s,d) flow. If $s = d$, then we consider that $F_{s,d} = 1$ and $f_{s,d}(1) = \emptyset$.

2.2.4.1 The Flow Formulation – Master Problem

Given that the set of flows F was obtained in advance, we propose to reformulate Problem 1 in terms of flows. Roughly speaking, the DMFC's resource management can be reduced to assigning flows to requests. Then, we integrate all problem's components into the following decision variables

$$\lambda \in \mathbb{B}^{\mathcal{R} x \mathcal{F} x \mathcal{P}} \qquad (17)$$

indicating whether request r's data is routed using flow $f \in$ F at priority p (i.e., $\lambda^r_{f,p} = 1$) or not (i.e., $\lambda^r_{f,p} = 0$). Note that, when selecting a flow for a request, we are implicitly selecting (i) the provider host and (ii) the sequence of arcs in G through which request's data is sent.

We enforce that a single flow, whose source node is the request's PoC, is assigned to each request at exactly one priority by imposing constraints

$$\sum_{f \in \mathcal{F}} \sum_{p \in \mathcal{P}} \mathbb{I}_{s(f)=u(r)} \lambda^r_{f,p} = 1 , \forall e \in \mathcal{R} \qquad (18)$$

where $s(f)$ denotes the source node of flow f. The computational capacity constraints can be translated to

$$\sum_{r \in \mathcal{R}} \sum_{p \in \mathcal{P}} \mathbb{I}_{d(f)=h} D^r_p \lambda^r_{f,p} \leq C_h , \forall h \in \mathcal{H} \qquad (19)$$

where $d(f)$ denotes the destination node of flow f.

Similarly, the latency requirements constraints are redefined as follows

$$\sum_{f \in \mathcal{F}} \sum_{p \in \mathcal{P}} \lambda^r_{f,p} \sum_{a \in f} \left[\alpha_a \left(\sum_{r' \in \mathcal{R}} \sum_{f' \in \mathcal{F}} \sum_{p' \in \mathcal{P}} \mathbb{I}_{a \in f'} T^{r'}_{p'} \lambda^{r'}_{f',p'}\right) + \beta_a\right] \leq L^r \forall r \in \mathcal{R} \qquad (20)$$

Finally, the objective function is

$$\text{QOS}_{\mathcal{F}}(\lambda) \triangleq \sum_{r \in \mathcal{R}} \sum_{f \in \mathcal{F}} \sum_{p \in \mathcal{P}} T^r_p \lambda^r_{f,p} \qquad (21)$$

We formulate the TD Problem in terms of flows as follows:

Problem 3 (Flow-Based TD (FTD) Problem).

$$\textbf{(FTD)} \quad \underset{\boldsymbol{\lambda}}{\text{maximize}} \quad (21)$$
$$\text{subject to} \quad (17) - (20)$$

Proposition 2. If F contains all possible flows on graph G, then problems 1 and 3 are equivalent, i.e., if (x∗,y∗,z∗) and λ∗ are their respective optimal solutions, then

$$QOS(x *) = QOS_{\mathcal{F}}(\lambda *)$$

The intuition behind Proposition 2 is that every flow in F satisfies the flow conservation constraints (6). Therefore, by building a solution for Problem 3 we are implicitly finding a valid assignment for variables **z** from the original formulation.

Proposition 3. *Problem 3 is* NP-*Hard.*

The proof of Proposition 3 follows the same lines as the proof for Proposition 1. We consider a setup for Problem 3 with sufficiently large latency requirements and note that the MK Problem can be reduced to such a particular instance. Therefore, Problem 3 is NP-Hard.

Remark 2. Even though Problem 3 can be linearized (similarly to what was done for Problem 1), the size of set $\mathcal{F}$ grows rapidly with the size of the network G, which may easily lead to a computationally intractable problem size. However, we remark that the structure of Problem 3 suits the Column Generation framework, which may be able to find good approximations in practice.

We discuss next how to further exploit the flow formulation to approximate the TD Problem.

2.2.4.2 Shortest-Flow Approximation

The main idea of the Shortest-Flow Approximation (SFA) is to consider that the optimal solution of the TD Problem is primarily built upon shortest flows. This is reasonable because the number of links routing a task's data is the first factor that impacts the latency (recall the discussion on Latency Constraints in Section III).

For a given network G, we define the set of shortest flows from source node s to destination node d as follows

$$\mathcal{SF}_{s,d} \triangleq \{f_{s,d} \in \mathcal{F} : \forall f'_{s,d} \in \mathcal{F}, |f_{s,d}| \leq |f'_{s,d}|\} \tag{22}$$

where $|f|$ is the size of set f, i.e., the number of arcs in flow f. Additionally, we consider that the set of shortest flows can be limited to a maximum number $K \geq 1$ of equivalent flows between every pair of source-destination nodes. We define the set of K-shortest flows between nodes s,d as $\mathcal{SF}^K_{s,d}$. Notice that, because a pair of source-destination nodes may have a number of shortest flows smaller than K, then $|\mathcal{SF}^K_{s,d}| = \min(K, |\mathcal{SF}_{s,d}|)$. Finally, we denote the set of the K-shortest flows for all pairs of nodes in G as

$$\mathcal{SF}^K \triangleq \{\mathcal{SF}^K_{s,d} : \forall (s,d) \in \mathcal{H}\}$$

In SFA, we build and solve Problem 3 using the set $\mathcal{SF}^K$ of K-shortest flows instead of the entire set of flows $\mathcal{F}$.

Remark 3. There are two immediate consequences of applying SFA: (i) Set $\mathcal{SF}^K$ can be efficiently obtained, e.g., via Dijkstra algorithm [30] and (ii) we consider a significantly smaller problem. However, by limiting set $\mathcal{F}$, we provide the solver with less flows to integrate solution candidates, which often results in sub-optimal solutions. In summary, if λ^* and λ^*_{SFA} are the optimal solutions of Problem 3 considering the entire set of flows $\mathcal{F}$ and considering set $\mathcal{SF}^K$ of the K-shortest flows, respectively, then,

$$\text{QoS}_{\mathcal{F}}(\lambda^*) \geq \text{QoS}_{\mathcal{SF}^K}(\lambda^*_{\text{SFA}}).$$

Henceforth, we refer to SFA limited by the K-shortest flows as K-SFA. The K-SFA's ability to achieve solutions close to the optimal (considering all flows) depends primarily on two factors, which we discuss next.

1) *Selecting K:* The larger is K, the better may be the approximation. For large enough values of K, the closer set $\mathcal{SF}^K$ gets to $\mathcal{SF}$, which provides the maximum number of shortest-flow candidates and, in turn, the best approximate solution. However, it is still not guaranteed that the real optimal solution can be achieved, given that optimal flows are not necessarily the

shortest ones (i.e., optimal flows may reside within $\mathrm{F} \setminus \mathrm{SF}$). Moreover, the optimization problem's size grows rapidly with K. Therefore, we must find the sweet spot in this trade-off for K, in order to obtain satisfactory approximations from computationally treatable optimization problems.

2) Selecting the K-shortest flows: In the case where $|\mathrm{SF}_{s,d}| > K$, the strategy used to choose the K shortest flows among this subset to constitute set SF^K may also impact the quality of K-SFA's solution. If, in a given set SF^K, flows tend to overshare the same arc, then sub-optimal assignments will integrate the solution in order not to violate the latency requirement constraints.

3) The idea is that we must choose the K-shortest flows that provide the largest number of feasible assignment candidates possible, in order to reduce the optimality gap (i.e. the relative distance to the optimal). Inspired by [31], we capture the notion of potential traffic load of a given arc by using its edge-betweenness centrality. In short, the edge-betweenness of a link a is the fraction of shortest flows traversing it. We propose to rank flow f's "idleness" level according to the following metric

$$B(f) \triangleq \sum_{a \in f} \left(1 - \frac{1}{|\mathcal{SF}|} \sum_{f \in \mathcal{SF}} \mathbb{1}_{a \in f'}\right) \qquad (23)$$

As the total traffic load traversing each arc is the second factor impacting latency, choosing the K-shortest most-idle flows provides a space of solution candidates with higher degree of freedom, potentially reducing the optimality gap.

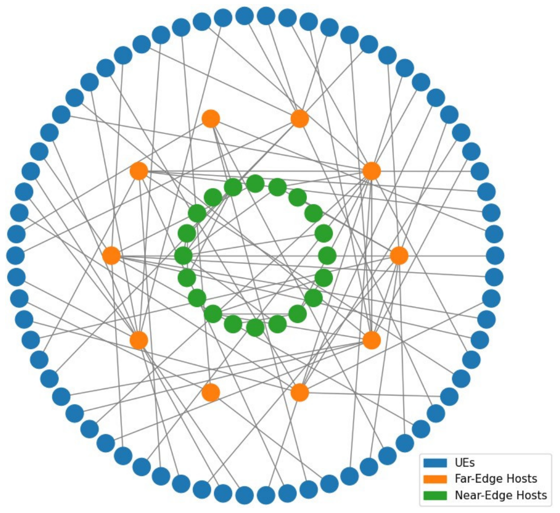


Figure 6. Experimental network topology consisting of 100 nodes: 70 UEs, 10 Far-edge nodes (PoCs) and 20 Near-edge hosts.

2.2.5 Experimental Results

In this section, we study the performance of K-SFA for specific values of K and how the selection strategy based on edge-betweenness may affect the quality of the approximation. In our experiments, we consider the Berlin topology: a cellular network consisting of 10 PoCs located according to the positions of T-Mobile BSs in Berlin extracted from [33]. Moreover, we consider that the PoCs communicate to a randomly generated (connected) network of 20 near-edge hosts. There is a set of 70 UEs, each of which randomly connected to a PoC. We plot in Figure 3 the network topology used in our experiments.

We consider that values of computational utilization and throughput only depends on the priority, i.e.,$D_p^r = D_p, \forall r \in$

R, $\forall p \in$ P and $T_p^r = T_p, \forall r \in \mathcal{R}, \forall p \in \mathcal{P}$. There are 3 priorities, such that, $\forall r \in$ R, computational demands are $D_1^r = 1.0$ GB, $D_2^r = 2.0$ GB, and $D_3^r = 4.0$ GB, and the throughput levels are $T_1^r = 10.0$ Mbps, $T_d^r = 20.0$ Mbps, and $T_3^r = 30.0$ Mbps. Far- and Near-edge hosts have C_h =32.0 GB and C_h = 64.0 GB of RAM, respectively. Requests' UEs are chosen uniformly at random, and their related latency is also uniformly selected from the interval [50.0,150.0] ms. All these values are consistent with the literature (e.g., [15]). For each arc in the network, the latency model parameters are randomly chosen from a fixed interval. Specifically, $\forall a \in$ A,

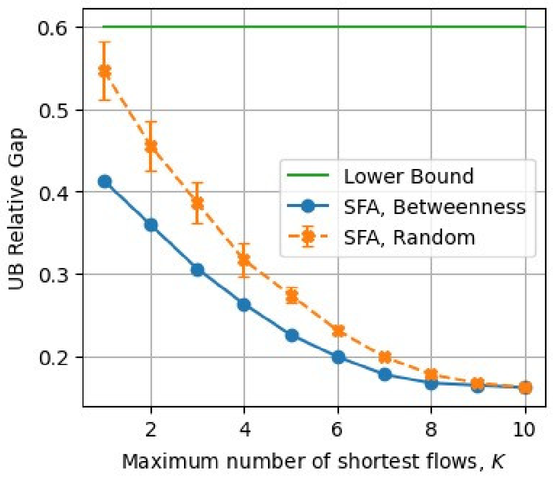


Figure 7. Upper bound gaps relative to (i) lower bound, (ii) K-SFA (Betweenness), (iii) K-SFA (Random), versus the maximum number of flows K.

Table 3. Experimental Parameters

Computational Capacity	Far-Edge Hosts	Near-Edge Hosts	
	C_h = 32.0 GB	C_h = 64.0 GB	
Latency Tolerance	$L^r \sim$ UNIFORM[50.0, 150.0] ms		
Priorities Attributes	p = 1	p = 2	p = 3
	T_p = 10.0 Mbps	T_p = 20.0 Mbps	T_p = 30.0 Mbps
	D_p = 1.0 GB	D_p = 2.0 GB	D_p = 4.0 GB
Arcs' Latency Parameters	$\alpha_a \sim$ UNIFORM[0.0, 5.0]	$\beta_a \sim$ UNIFORM[0.0, 200.0]	

α_a is chosen from interval [0.0,5.0] and β_a is selected within [0.0,200.0]. We summarize the parameters in Table 2.

Given the described experimental setup, we generate the request set such that PoCs can provide all their UEs' requests at the lowest priority possible. This guarantees that the optimization problem always has a feasible solution, which we refer to as the *trivial* solution. We propose to compare the approximation of K-SFA with different values of K and with diverse selection strategies. We also define the lower bound (LB) as the trivial solution where all requests are deployed at the UEs' PoCs at the lowest priority. We compare K-SFA and LB with Problem 2's upper bound (UB), i.e., the solution of its continuous relaxation.

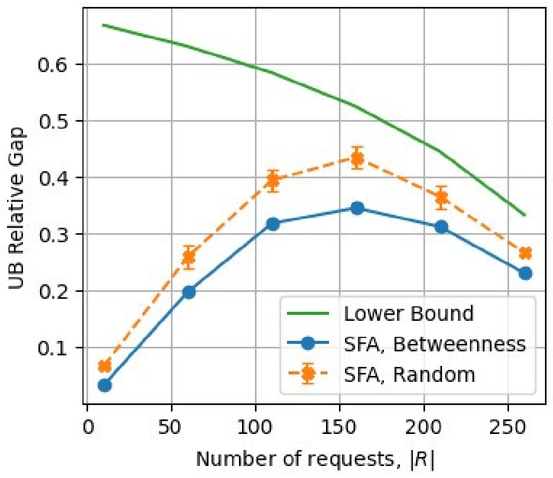


Figure 8. Upper bound gaps relative to (i) lower bound, (ii) K-SFA (Betweenness), (iii) K-SFA (Random), versus the total number of requests |R|.

In our first set of experiments, we propose to observe the impact of K and the different shortest flows selection strategies for a set of 150 requests. For each $K \in \{1,...,10\}$, we solve[2] an instance of Problem 3 with a set of K-shortest flows filtered (i) uniformly at *random*, $\mathcal{SF}_{R}^{K}$, and (ii) using *betweenness* strategy, SF^{K_B}. For each K, we generate 30 problem instances with randomly chosen $\mathrm{F}_{K,R}$ and average their relative UB gaps. Finally, we calculate the relative UB gap for the betweenness strategy. We show the first experimental results in Figure 8.

Table 4. Run time for different solution approaches.

	Original Formulation	K-Shortest Flow Approximation				
		K=2	K=4	K=6	K=8	K=10
Auxiliary Problem Time	-	912	1122	1507	2355	2880
Main/Master Problem Time	-	18352	38355	67355	80561	110803
Total Solution Time	7458	19264	39477	68862	82916	113683

We first note that the performance gap between random and betweenness (of around 27%) has its largest value when $K = 1$. At this point, both approximations have large UB gaps and as K increases, (i) K-SFA tends to provide better results (i.e., closer to the UB), in general, until it reaches a convergence point at $K = 8$, and (ii) the performance gap between random and betweenness decreases. Notice that KSFA (regardless of the selection strategy) may achieve results up to 10% far from the problem's UB for when $K = 10$. At this point, the selection strategy has little impact because, for $K = 10$ in this network, we are considering almost all shortest flows. This hypothesis is also supported by the fact that the standard deviation for the random

[2] The optimization model was developed with PyOMO and solved using IBM CPLEX solver.

selection is close to zero for $K > 5$. These facts altogether corroborate our conclusion that the approximation provided by K-SFA is at most 10% from the optimal solution in this setup. This means that the optimal solution is composed of some non-shortest flows.

We show on Table 4 the completion time, in seconds, for different stages of our solutions. In general, the auxiliary problem's solution time is considerably smaller than the master problem's solution time. As we briefly mentioned in Section IV, the shortest paths can be efficiently obtained. The largest overhead is due to betweenness calculation time, which is still a negligible increase in comparison with the master problem solution time. Experiments were run on a computer with an Intel(R) Xeon(R) CPU E5-2650 v2 running at 2.60 GHz using 256 GB of RAM and Linux Debian 6.3.0 operating system.

In our second set of experiments, we fix $K = 2$ for K-SFA and solve the problem for requests sets of different sizes. Once again, we plot K-SFA's (both Betweenness and Random) and LB's gap relative to the problem's UB. We observe that, for $|\mathrm{R}| = 10$, the network can accommodate almost all the request at the highest priority and K-SFA was capable of approximating this solution. As we increase the number of requests, the larger is the gap with the UB. This trend can be explained by the fact that having to handle more requests require more complex solutions not only based on shortest paths. In other words, having less busy shortest flows improves the result (Betweenness is still performing better than Random), but it gets a less significant advantage, given that non-shortest flows are more present in the solution as the problem grows in complexity (with more requests to handle). Interestingly, after reaching its peak at $|\mathrm{R}| = 160$, the curves tend to decrease and to get close to each other. We assign this trend to the increasing excess of requests (i.e., those which cannot be accommodated in the network at all), which end up having trivial assignments (PoC with smallest priority). This excess tends to dominate the portion of requests that are optimally assigned as $|\mathrm{R}|$ grows large, which, in turn, makes the relative UB gap reduce.

2.2.6 Conclusions

The TD Problem captures fundamentally the primary requirements of latency-sensitive systems based on MEC architectures. In this work, we propose a linear integer programming model for the TD Problem. We model requests' latency as the classic queuing delay and approximate it as a linear function of the total achieved throughput in the participating communication links. We approach the problem using the K-SFA and we introduce a shortest flow selection strategy based on edge-betweenness centrality. In our experiments, we could observe our technique's performance and conclude that K-SFA can achieve satisfactory results even for relatively small values of K. The results presented in this Section will pave the way to more sophisticated modelling and more efficient approximations, for example, considering randomized rounding algorithms [36] and other shortest path selection strategies. Furthermore, the framework can be expanded to consider a mathematical modelling for the robust optimization of the expected QoS under random requests and accounting for uncertainties, such as chaotic deployment and volatile, mobile UE association. In this scope, we can even use the resulting stochastic model as a comparison baseline for Realtime problems based on dynamic and distributed policies.

2.3 Link Optimization in Smart Highway

2.3.1 Introduction

This section addresses the problem of optimizing the location of the Road Side Unit (RSU) for determining optimal link flow which is used to determine the minimum set of links that is equipped with traffic monitoring devices to identify vehicle paths in a connected vehicles environment. Network connectivity for RSU deployment focuses on reducing link flow connection efficiency and disconnection interval between RSUs and connected vehicles, and there are observable and non-observable links in RSU.

In V2X, RSU plays a significant role in not only detecting VRUs on the road from mounted sensors (e.g., camera), but also exchanging data for VRUs detected through wireless communication, relaying messages between VRUs, and integrating information. Advanced sensor technology attached to the vehicle can collect information about the vehicle's speed, location on the road, and VRUs detected around the vehicle. The vehicle periodically exchanges this information with connected vehicles and RSUs installed on the road through wireless communication.

RSU can also collect information about traffic-based infrastructure (e.g., traffic lights), and the collected infrastructure information is transmitted to nearby connected vehicles and traffic control integration centers. Based on the information collected by RSU, integrated traffic guidance and control is possible to optimize traffic conditions. Therefore, it is important not only to detect VRUs on the road, but also to optimize RSU location for smooth link flow with surrounding VRUs.

2.3.2 Problem Formulation

The RSUs can observe the position and velocity of the connected vehicle. As shown in Figure 9, RSUs obtain traffic counts from multiple links via V2X, each RSU can simultaneously collect information for multiple links via V2V communication.

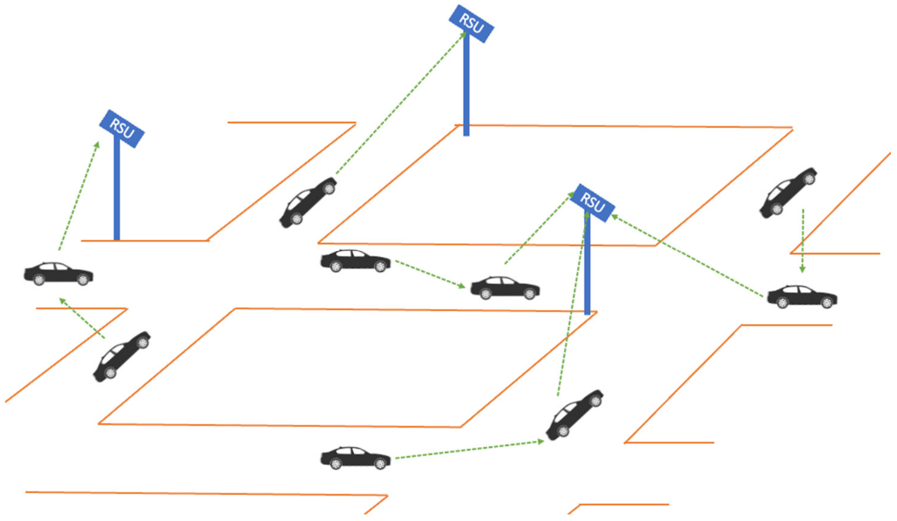


Figure 9. RSUs obtain traffic counts from multiple links via V2V

From the RSU point of view, the flow of a link is an observed flow of a link and a flow of an unobserved link. The flow of an observed link can be estimated using shock wave theory and a car-following model at that link [8]. And the flow of the unobserved link is to be inferred using the node link flow conservation equation based on the observed link flow.

Communication delay is a major cause of measurement error related to data collected by the RSU, so measurement error is defined as communication delay related to the observed link flow. The communication delay is assumed to be the sum of the propagation delay and the data packet queuing delay during V2X communication, and the data packet queuing delay is the amount of time the RSU waits for a data packet sent by the connected vehicle to be executed.

It is to define the RSU location formulation model to achieve link flow determination. It is assumed that RSUs should only be located on nodes in the transportation network.

The decision variables x_{kj},y_k are binary variables. If the RSU is located at node $k \in K$, then $y_k = 1$. Otherwise, $y_k = 0$, where j is index for links in a transportation network, k is index for nodes in a transportation network, K is set of nodes in a transportation network, and y_k is also binary variable. If link $j \in J$ is allocated to the RSU located at node$k \in K$, then $x_{kj} = 0$. Otherwise, $x_{kj} = 1$ where J is set of links in a transportation network.

The objective function [9][10] aims to minimize the total measurement error associated with observed link flows and unobserved link

$$\text{minimize } F(x,y) = \min\ (F_1(x,y) + F_2(x,y))$$

$F_1(x,y)$ and $F_2(x,y)$ are defined as:

$$F_1(x,y) = \sum_k \sum_j x_{kj} t_{kj} + \sum_k \sum_j x_{kj} t_{kj} \frac{1 - Cv_k^2}{2} \cdot \frac{p \sum_j x_{kj} f_j}{\mu_k (\mu_k - p \sum_j x_{kj} f_j)} + \sum_k \sum_j x_{kj} \frac{1}{\mu_k}$$

$$F_2(x,y) = \sum_{n \in N} \left(\sum_{j \in J} \delta_j^n - \sum_{j \in J} \sum_k x_{kj}\, \delta_j^n \right) \cdot \left(\sum_{j \in B_n} - \sum_{j \in B_n} \sum_k x_{kj} \right)$$

where $F_1(x,y)$ represents the sum of the measurement errors associated with each observed link flow. $F_2(x,y)$ gives the cumulative quantity of unobserved links connected to non-central nodes with one unobserved new link, Cv_k is the coefficient of variation of service time associated with the RSU located at node $k \in K$, p is the packed size of each data packed sent from a connected vehicle to RSU, t_{kj} is the expected information travel time from link $j \in J$ to node $k \in K$, f_j is the number of vehicles on link $j \in J$, μ_k is capacity of RSU located at node $k \in K$, B_n is the set of new links connected to non-centroid node n, and δ_j^n is the node-link index: $\delta_j^n = 1$ if link j is connected to node n, else $\delta_j^n = 0$.

And there are constraints as follow:

$$\sum_{j \in B_n} - \sum_{j \in B_n} \sum_k x_{kj} \le 1, \qquad \forall n \in N$$

$$\sum_k x_{kj} \le 1, \qquad \forall j \in J$$

$$\sum_k f_j x_{kj} \le \mu_k y_k, \qquad \forall k \in K$$

$$x_{kj} \le y_k, \qquad \forall k \in K,\ \ j \in J$$

2.3.3 Simulation

The above formulation is a multi-objective optimization as a multi-criteria decision-making domain containing two objective functions to be optimized simultaneously. The epsilon-constraint method, which is one of the representative methods for solving multi-objective optimization problems, is used, and the simulation is based on python and utilizes the pyomo [11] and PuLP [12] packages for optimization problems to obtain results.

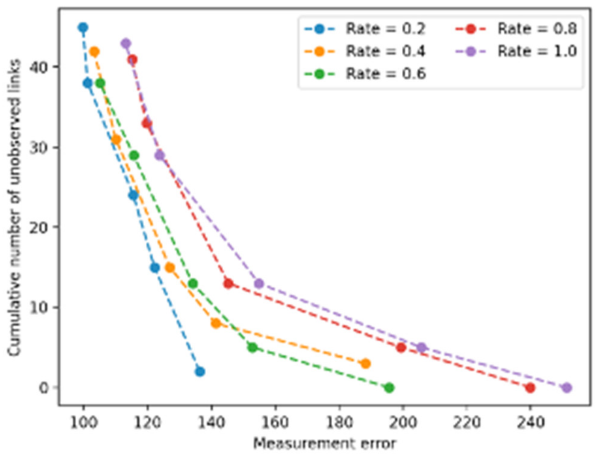


Figure 10 Pareto front generated from the RSU location model

Figure 10 shows the pareto front generated from the RSU location model according to different rates. As shown in Figure 11, the maximum cumulative quantity of unobserved links that guarantee a complete link flow determination is 45, 42, 38, 41, and 43 at rates of 0.2, 0.4, 0.6, 0.8, 1.0, respectively. When the accumulation of unobserved compounds reaches a maximum, the measurement error reaches a minimum along the pareto boundary. It indicates that the reduction in inference error associated with the inferred link flow requires an increase in the measurement error associated with the observed link flow.

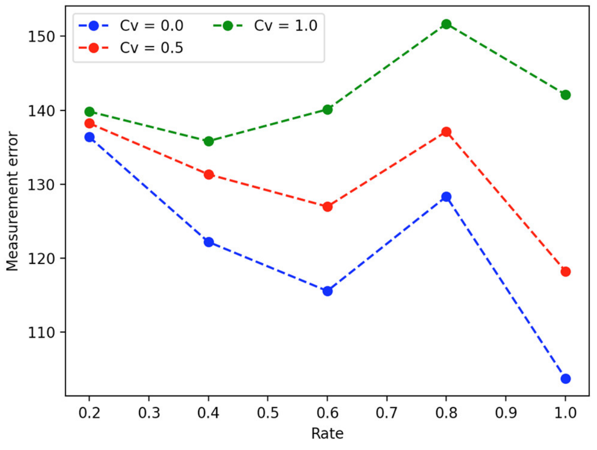


Figure 11 The relationship between the measurement error and the penetration rate of the connected vehicle

Figure 11 shows how the measurement error, which is the sum of the propagation delay and the data packet queuing delay, changes as the penetration rate of connected vehicles according to the coefficient of variation of service time value increases. When Cv is 0, the measurement error decreases as the penetration rate of connected vehicles increases from 0.2 to 0.6, and the measurement error increases as the penetration rate of connected vehicles increases from 0.6 to 0.8. This is also the case when Cv is 0.5.

This is mainly due to an increase in the penetration rate, which significantly reduces delivery delays, while data packet queuing latency is not significantly affected. This is consistent with the fact that Cv does not significantly affect the measurement error when the penetration rate is low. However, when Cv is 1, as the penetration rate increases, the overall measurement error continues to increase (excluding from 0.2 to 0.4), which can be inferred that the data packet queuing delay is sensitive to the penetration rate. On the other hand, more detailed research is needed in the future to analyze the phenomenon that the measurement error decreases again regardless of the size of Cv when it is 0.8~1.0.

In conclusion, the RSU location to achieve link flow determination is modelled as a multi-object optimization problem based on measurement error and inference error, and analysis is attempted with the pareto optimal solutions. As the measurement error increases along the pareto front, it is possible to estimate the relationship that the inference error may not decrease even if the cumulative amount of unobserved links decreases. A complete analytic problem formulation is not possible because the inference error is formulated indirectly as the cumulative number of unobserved links, but the analysis is still meaningful in pareto optimization in terms of measurement error and inference error between links. Further studies are needed from several perspectives in the future.

2.4 Intelligence Placement and Migration in Smart Highway

In D3.1, we presented our research about state-of-the-art algorithms for distributed intelligence. We described the benefits and the downsides of the centralized and distributed algorithms approach. In this section, we present some first studies on service placement problem setting and a mathematical formulation for the scenario. Our objective is to use the work presented in this section and adapt to vehicular networks.

2.4.1 Introduction

Edge computing are designed to support vehicular applications by road services to nearby vehicles to support more compute-intensive, latency-aware, and even energy-aware applications while suppressing the latency impact caused by cloud communication while not overloading the network. Providing these services in the edge requires effective service placement, as placing them far away from related services can induce latency costs, and placing all the services on a single device will consume more device resources than are available, reducing service reliability.

To this service allocation problem, we propose a general Deep Q-Network (DQN) methodology for service placement, which considers the placement optimization of multiple intercommunicating services on a network. This placement optimization considers the improvement of the network performance by minimizing the total impact on the network, while also improving energy efficiency. These are generally conflicting objectives, defining the problem as a Multi-Objective Optimization (MOO) problem. Valid placements should satisfy device constraints, such as available memory, network constraints, such as available bandwidth, and application constraints, such as maximally allowed latency between two services.

In this section, we use several State-of-the-Art (SotA) Multi-Objective Reinforcement Learning (MORL) methods to solve the service allocation problem, using the strengths of DQNs to effectively place the services across the network. Our proposed methodologies use scalarization and support dynamic weight changes throughout the network lifespan, providing support for a higher-level control mechanism.

2.4.2 Problem Setting

The service allocation problem is an expansion of the General Quadratic Assignment Problem (GQAP). The service allocation problem expands on this by putting constraints on both the devices and the network, whereas GQAP does not consider transport constraints. GQAP has been shown to be solvable for up to 22 devices, showcasing the problem complexity

[13]. Additional complexities arise due to applying it to edge computing: the available run-time of the placement algorithm is generally very low, as it takes seconds to find a solution, depending on the context. Similarly, the available resources for fog devices are generally limited, requiring algorithms with low computational complexity. This further bounds the methodologies applicable to the problem. We additionally extend GQAP to consider MOO, as the service allocation problem considers multiple competing objectives, defined further below.

2.4.3 Multi-Objective Optimization

MOO considers the optimization of multiple, possibly conflicting, objectives. This is formulated in the Equation 1.

$$\begin{aligned} \min\ & \mathbf{F}(\mathbf{x}) = [F_1(\mathbf{x}), F_2(\mathbf{x}), \ldots, F_o(\mathbf{x})] \\ \text{s.t.}\ & \mathbf{x} \in X \\ & g_i(\mathbf{x}) \geq 0, i = 1, 2, \ldots, k \\ & h_j(\mathbf{x}) = 0, j = 1, 2, \ldots, l \end{aligned}$$

Equation 1. Multi-Objective Optimization

The goal is to traverse the search space x ∊ X to simultaneously minimize the cost of the objective function vector F for o different objectives, while considering k different inequality constraints, gi , and l different equality constraints, hj . These objectives conflict, thus there is no single optimal solution, rather a front of non-dominated solutions. This front is the Pareto Front (PF), where each solution is at least better for one objective while being worse for at least one other objective. Figure 12 shows an illustration of a Pareto front, where F_1 and F_2 represent arbitrary minimization objectives. Circles represent sub-optimal solutions in the search space, whereas the squares represent the optimal solutions, or the Pareto front. The red colour distinguishes solutions which would become invalid if certain constraints are considered and shows how the Pareto front would reshape under said conditions. Solutions A and B are non-dominated solutions, signifying that they are worse on at least one objective while being better on at least one other objective. Both solutions dominate C, which is equal to B concerning objective F2, but worse for objective F1. However, the search space for the placement problem is constrained. The solutions which do not satisfy the constraints are shown on Figure 12 as dark red nodes. By removing these from the search space, it becomes clear that solution C becomes part of the PF, as B becomes infeasible. However, the entire PF is not necessary for the service allocation problem, only one solution is required from this front. The selection of this solution is done by a Decision Maker (DM), which defines the preferences for the set of objectives. This definition is often done using scalarization. By summing the objectives together, the utility of a solution can be defined as a linear combination of the objectives. This scalarization is done using a weighted function, where the weights represent the importance of each objective. Popular techniques for this approach include the weighted sum approach, which purely uses the weights defined by the DM, and the Chebychev scalarization, which also incorporates a reference vector, giving a search direction to the algorithm [14]. Scalarization is one of the simplest approaches for solving MOO problems, but it is often difficult for the DM to find the optimal set of weights. This has an especially large impact on Machine Learning (ML) scenarios, where the training of weights for a neural network can consume a large amount of time and resources.

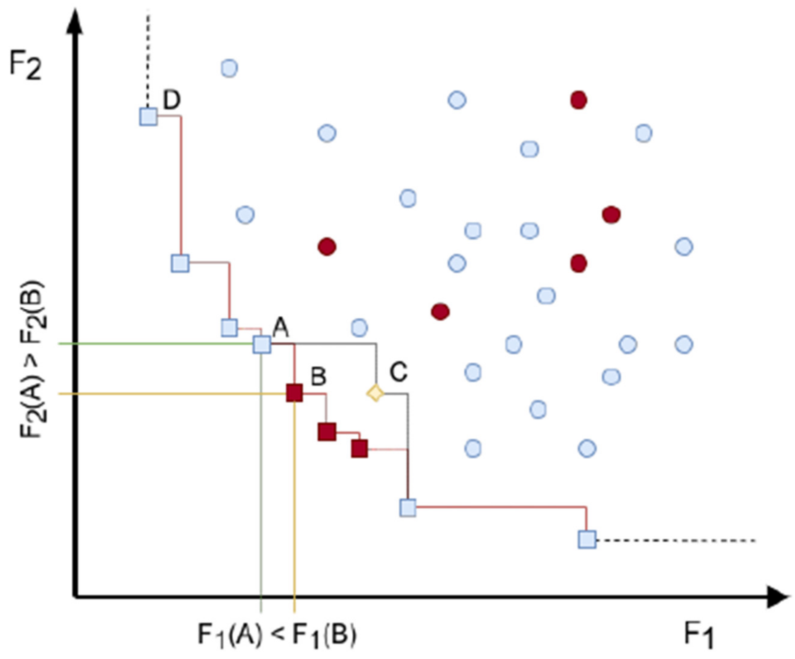


Figure 12. Illustration of a Pareto front, F_1 and F_2 represent arbitrary minimization objectives.

2.4.4 Approach

2.4.4.1 Linear Scalarization

Linear scalarization is the baseline approach for solving generic MOO problems. It uses the strength through simplicity approach of the weighted sum, where objectives are multiplied with a predefined weight and summed together, as defined in Equation 2. This approach has been applied in the SotA already, including Tang et al. [15]. While this approach enables tackling MOO problems with single-objective approaches, it suffers from the inability to handle a change in the weights of the objectives. To this end, we propose expanding on the scalarization approach by intelligently managing the weights of the objectives, as outlined below.

$$u(r) = \sum_{i=1}^{n} r_i * \omega_i$$

Equation 2. Weighted Sum

2.4.4.2 Deep Optimistic Linear Support

One approach proposed by Mossalam et al. is the Deep Optimistic Linear Support Learning (Deep-OLS) algorithm [16]. This algorithm supports bi-objective optimization through scalarization. The approach first trains two Deep Neural Networks (DNNs) on the weight extrema, one optimized for each objective, and then uses the two DNNs to predict the utility of a solution. This is then used to create a Convex Coverage Set (CCS) and apply Optimistic Linear Support (OLS) to select the weights for which a new policy could gain the most improvement in finding a new optimal policy. This results in multiple trained agents, each one optimal for a range of weights. These agents can then be swapped at runtime based on the weights required. The approach aims to minimize the amounts of networks trained to cover the weight space. This approach has been expanded upon for application on the placement problem. Due to the current limitations of the Deep-OLS algorithm, only two objectives are used. To keep the objectives as conflicting as possible, a focus was put on both a device and a network optimization objective. The network objectives were scalarized together and normalized to the range [0, 1]. This was subsequently also done for the device objectives. This approach reduces the reward vector to two objectives.

2.4.4.3 Conditioned Network

Our final approach uses a Conditioned Network (CN), removing the need of having multiple pre-trained agents completely, by introducing the weights into the observation space of the agent. As proposed by Abels et al. [17], the approach is based on a Universal Value Function Approximator (UVFA), where a network learns state and goal embeddings, using a distance-oriented metric to combine both. These goal embeddings are represented in MORL problems as the weight vectors. Training happens end-to-end, with the state and goal embedding as input and the multi-objective Q-values as output. In addition, a Diverse Experience Replay (DER) methodology is applied to the CN, which is a replay buffer which focuses on diversity. This is especially important, as the network should memorize the impact of the different weight vectors as well as the impact of the state-action pairs. Their approach is shown to have improved results compared to various other methodologies, including the multi-network approach.

2.4.5 Results

For evaluation, a use-case was crafted of 10 devices and 10 tasks, providing 10^10 possible different placements. The networks and additions were built using RLLib. Four objectives were evaluated: Energy, Worst Case Execution Time (WCET), Latency and Bandwidth. For the Deep-OLS approach, Energy and WCET were scalarized as device objective, and Latency and Bandwidth scalarized as network objective. Due to instability and slower convergence, the vector Q-values, proposed by Mossalam et al. were not used [16]. We expanded on the existing approaches of the Deep-OLS and Conditioned networks by building them using a Double Dueling DQN, which improves general stability and convergence. The hyperparameters used are found in Table 5.

Table 5. Hyperparameters

Hyperparameter	Value
γ	0.95
Learning Rate	0
ε	150 000
Batch Size	32
Buffer Size	20 000
Weight Change Interval	10 000 Steps

The results were compared with a Non-dominated Sorting Genetic Algorithm II (NSGA-II) approach, as proposed in previous research [18]. This algorithm was configured with a population size of 100, running for 1000 iterations. Additionally, a comparison was made with a standard random search, iterating over 1000 possibilities before finishing.

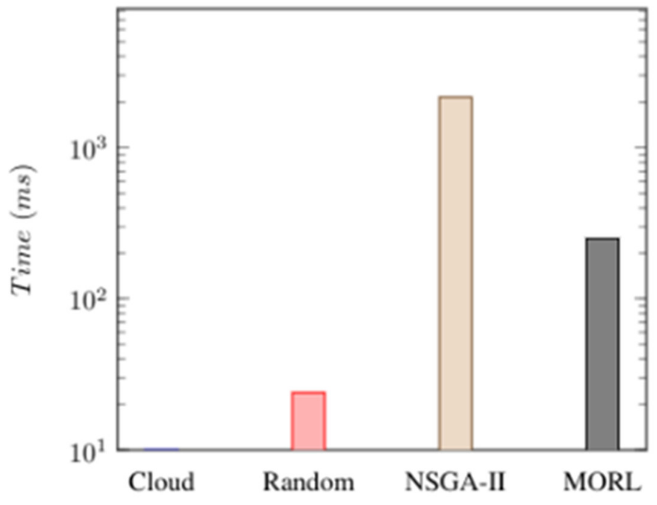


Figure 13. Execution Time

All algorithms were generally able to find solutions that satisfied all constraints. Figure 13 represents the average time required to find a single solution. We pooled the MORL algorithms, as they had similar networks and consequently similar inference time. The Cloud solution refers to placing all possible tasks on the cloud, showcasing the traditional approach. The log scale showcases that the proposed MORL approaches outperform the traditional NSGA-II algorithm by a factor 5. Note, however, that both NSGA-II and Random Search depend on the number of iterations to determine timing, whereas the proposed MORL algorithms have static timing and resource usage in nature. More interesting results are found on Figure 14, which shows the average reward over 50 runs. The x-axis shows the weight for the network objective, where a 0 is the corner weight focusing on device objectives and 1 is the opposite corner weight focusing on network objectives. Note that the cloud solution does not satisfy the latency solution and is invalid, being purely shown as reference. It is clear that the NSGA-II algorithm finds the optimal solution. This is at the trade-off of consuming considerably more time and resources. The bi-objective Conditioned Network approach comes quite close to the NSGA-II algorithm, which showcases that a trained network is a valid approach in resource-constrained service placement. Interestingly, the neural network generally also finds better solutions in 50 timesteps than the random algorithm does in 1000. This is partially accredited to a light skew in the normalization, making network objectives slightly more valuable. In addition, if a corner weight of the Deep-OLS fails to converge, the subsequent search becomes infeasible. We notice that the conditioned network trained on four objectives succeeds at finding useful solutions but is outperformed by nearly all other approaches. This is likely due to the large jump in complexity between solving for two and four objectives. The results showcase the brittleness of applying Deep-OLS in practical scenarios. The approach depends on finding the policies for the weight extrema first, but if these values are far apart, the algorithm stops working as expected. In addition, the approach suffers from search space complexity differences. In our scenario, it is considerably easier to optimize for network objectives, by putting all services on the same device, than it is to optimize for device objectives. This mismatch makes it difficult to build an automated Deep-OLS search methodology, as the network objective policy converges considerably faster. We recommend to instead train individual policies with individual hyperparameters per weight and apply OLS on top.

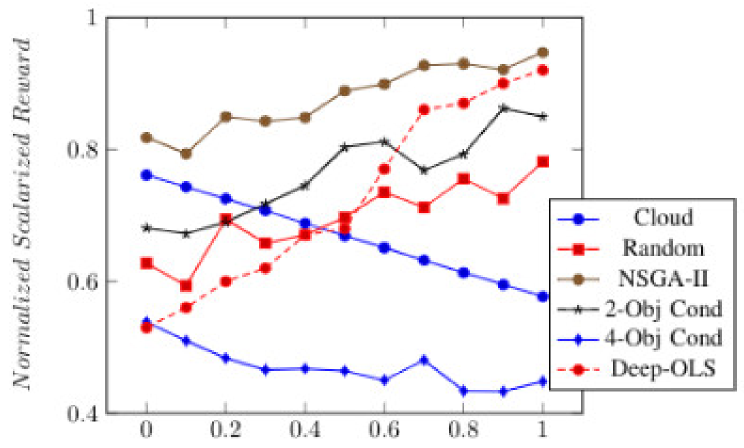


Figure 14. Average Scalarized Reward

2.4.6 Future Steps

Using the MORL techniques described, further objectives, such as privacy and security, and constraints, such as software requirements, can easily be added. The impact of these added objectives and constraints should be evaluated, and the scalability of the proposed techniques validated. The trained models could be further improved to reduce resource consumption and inference time, using network pruning, as proposed by Balemans et al. [19]. Major improvements will most likely also be achieved when evolving towards a distributed RL methodology, where the problem is reduced to multiple sub-problems, improving scalability but reducing the quality of the found solution.

2.5 Delay-aware offloading task association for networked computing

The networked computing approach represented in D3.1 [6] involves several network and resource management decisions to be made before the access to a suitable computing server can be provided by the network operator. One of the most important decisions is to decide the association of the computing tasks to access points and attached edge servers. The association decision is nontrivial because there are a number of factors affecting the end-to-end offloading delay. In general, the transmission delay effect of bi-directional offloading communication links as well as computing delay of the servers must be included in the decision-making process.

In this section, we study the offloading of computing tasks of users within the network area of interest. Typical computationally involved offloading tasks involve e.g., 3D video encoding for virtual reality and object detection from a video frame for augmented reality purposes. The main aim of the deployed decision-making framework is to maximize the network delay coverage probability which corresponds to a fraction of users that experience target offloading delay deadline. We apply both centralized and distributed delay-aware association approaches and evaluate their delay coverage probability in a network simulator. The target high-level server access model is illustrated in Figure 15. It involves on-site users participating some event and willing to upload computing tasks to proximity servers, heterogeneous wireless access point network, and edge servers at a wired proximity to the closest access point.

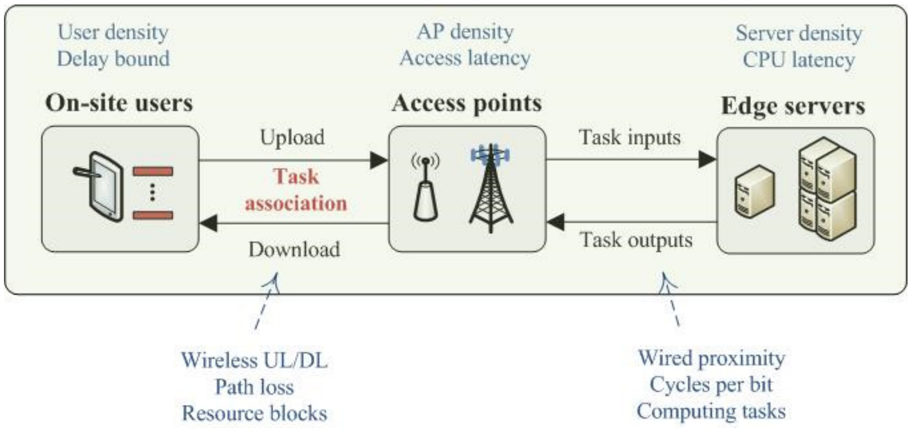


Figure 15. Target heterogeneous server access model for computation offloading.

2.5.1 Problem

We consider *N* users randomly deployed in a network which involves *M* heterogeneous access points (APs) in a target region of interest. Each user has a computing task that it wishes to offload to a computing server (CS) and then receive the result within a target delay deadline. The *n*th computing task is associated with the *m*th AP-CS pair if the association decision variable x_{nm} is 1, and 0, otherwise.

Let C_m and S_m be the maximum capacity of the *m*th AP and CS, respectively. The former is measured as a sum over allocated resource blocks q_{nm} whereas the latter is the number of computing tasks that can be processed in the given server. Let U_{nm} be the user utility of the *n*th user associated to *m*th AP-CS pair which is selected to be the inverse of the sum of the mean delay caused by the uplink D_{nm}^{ul}, computation server D_{nm}^{cs}, and the downlink D_{nm}^{dl}. The constrained optimization problem at hand is given by

$$\max_{x_{nm}} \left\{ \sum_{m=1}^{M} \sum_{n=1}^{N} U_{nm} x_{nm} \right\}$$

$$\text{subject to}$$

$$U_{nm} = \frac{1}{D_{nm}^{\text{ul}} + D_{nm}^{\text{cs}} + D_{nm}^{\text{dl}}}, \forall n, m \ \text{(offloading utility)}$$

$$x_{nm} \in \{0,1\}, \forall n, m \ \text{(binary decision variables)}$$

$$\textstyle\sum_{m=1}^{M} x_{nm} \le 1, \forall n \ \text{(association to one AP)}$$

$$\textstyle\sum_{n=1}^{N} q_{nm} x_{nm} \le C_m, \forall m \ \ \text{(max AP capacity)}$$

$$\textstyle\sum_{n=1}^{N} x_{nm} \le S_m, \forall m \ \ \text{(max CS capacity)}$$

where the applied constraints are explained in the respective parentheses. The uplink and downlink delays for a computing task are calculated via division of number of transmitted bits and achieved link bit rate while the server delay is calculated via division of number of needed computation clock cycles and circuit clock rate.

2.5.2 Applied Solution Frameworks

In order to resolve the above integer linear programming problem, we first relax the binary task association variable x_{nm} to be a continuous-time fractional decision variable which has zero lower bound and upper bound of one. After finding the fractional decisions variables, the binary decision variables are found by rounding to the nearest integer.

We then apply two different approaches to resolve the relaxed problem based on both centralized and distributed control mechanisms. In the former approach, it is assumed that there is a central controller which obtains the utility information from each candidate task association, makes all the decisions, and then informs the respective AP-CS pairs which tasks are allocated to them. In this case, the interior point approach is applied to solve the resulted centralized linear programming problem (cf. [31]).

Regarding the distributed approach, the main goal is to allow each user offloading a task to decide which AP-CS pair to associate. In this case, we modify the Lagrange decomposition approach originally proposed in [32] which does not consider offloading end-to-end delay with server offloading computing. Specifically, the Lagrange decomposition of the problem with two coupled constraints becomes $L(\mathbf{a}, \mathbf{b}) = \sum_m \sum_n U_{nm} x_{nm} - \sum_m \sum_n a_m q_{nm} x_{nm} - \sum_m \sum_n b_m x_{nm} + \sum_m a_m C_m + \sum_m b_m S_m$ where $\mathbf{a}$ and $\mathbf{b}$ are the Lagrange vector multipliers. After some manipulations, the coupled constraints become separable so that each user can decide the computing task allocation so that the mth AP-CS pair is selected for the nth task that maximizes the function $\arg\max_m (U_{nm} - a_m q_{nm} - b_m)$ while respecting the target constraints The gradient decent method is used to find the Lagrange multipliers at each AP-CS pair which then broadcast the values to assist the user-induced task association decisions.

2.5.3 Results

The main aim of the simulation study is to reveal the delay coverage probability performance as function of different delay targets for the computing task association concepts presented in the previous subsection. Recall that the delay coverage probability corresponds to the fraction of users that achieve the target offloading delay performance in a network region of interest. It is assumed that if no association is possible for some tasks, the task immediately causes a delay outage event.

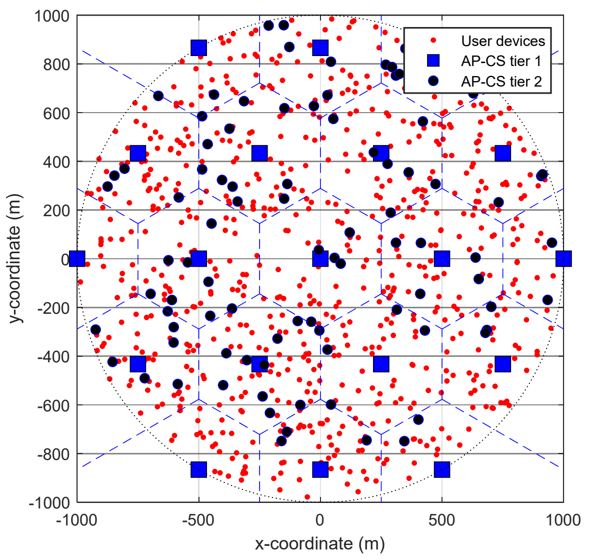


Figure 16. A snapshot of target heterogeneous network topology

In the target simulation network, we deploy a two-tier network architecture in the region of interest with the radius of 1000 m. A snapshot of the network topology is shown in Figure 16 and the main simulation parameters and used values are provided in Table 6. Matlab toolboxes are utilized where possible.

Table 6. Main simulation parameters and values.

Parameter	Value
Network radius	1000 m
Network architecture	Heterogeneous 2-tier
Path loss model	UMi_A
Shadowing	Lognormal; 3dB
Transmission powers	43 dBm (Tier 1 AP) 27 dBm (Tier 2 AP) 23 dBm (Device)
Node densities	Tier 1: 19 APs (deterministic) Tier 2: 90 APs (uniform) Users: 900 users (Poisson)
Number of task bits	46 kbit
Bandwidth per task	5 MHz
Noise power	-174 dBm/Hz
Carrier frequency	2 GHz
AP bandwidth	60 MHz
CS task capacity	10
Mean cycles per bit	1000 (exponential)
CPU rate	3 GHz

In Figure 17, we first illustrate the relative contributions of different parts of offloading delay regarding the uplink for uploading the video frame, server for finishing the computing task, and downlink for returning the computing result within the uploaded video frame. In this case, we have used the standard distributed received power aware association method which favours the performance of the downlink delay contribution when selecting the task association. It is seen that, as expected, the downlink performs the best followed by that of the server and uplink delay coverage sub-contributions.

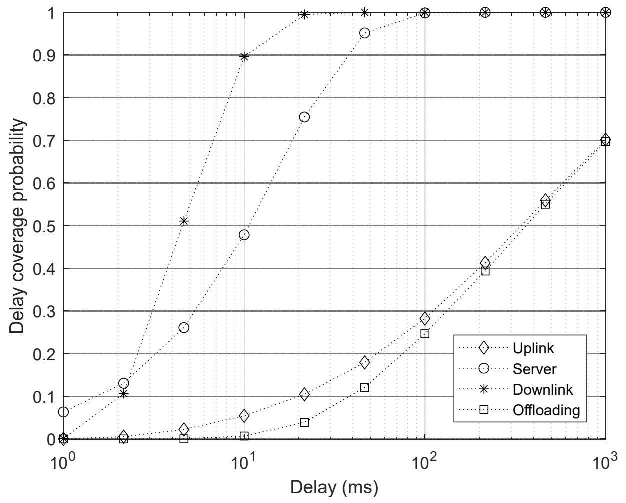


Figure 17. Comparison of different parts contributing on overall offloading delay coverage as function of set delay target for power-aware association.

We next move to evaluate the comparison of the centralized and distributed delay-aware task association with a standard power-aware association approach discussed above. In Figure 18, we first present the delay coverage probability for the end-to-end offloading delay for the case where the capacities of the APs and CSs are unlimited. It is seen that after certain delay threshold, the delay-aware approaches start to clearly outperform the power-aware method as it is able to select the associations more efficiently using the delays of all subparts. Since the capacity constraints of APs and CSs are not activated in this case, the performance of centralized and distributed approaches for the delay-aware method are quite similar. In Figure 19, we then activate the capacity constraints, and it is visible that the performance gets worse because the association opportunities are reduced with capacity limitations. Moreover, the difference between delay-aware and power-aware approaches start to increase. This is because, unlike the delay-aware approaches, the power-aware approach does not use the information on the capacities of the APs and CSs. Furthermore, it becomes more difficult for the distributed approach to keep up with the centralized approach because the distributed approach is not directly aware of other task association decisions.

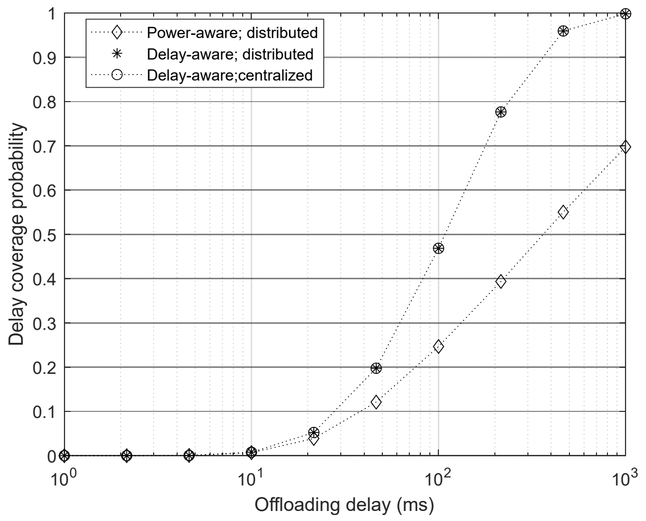


Figure 18. Comparison of different task association approaches with unlimited AP and CS capacities as function of target offloading delay.

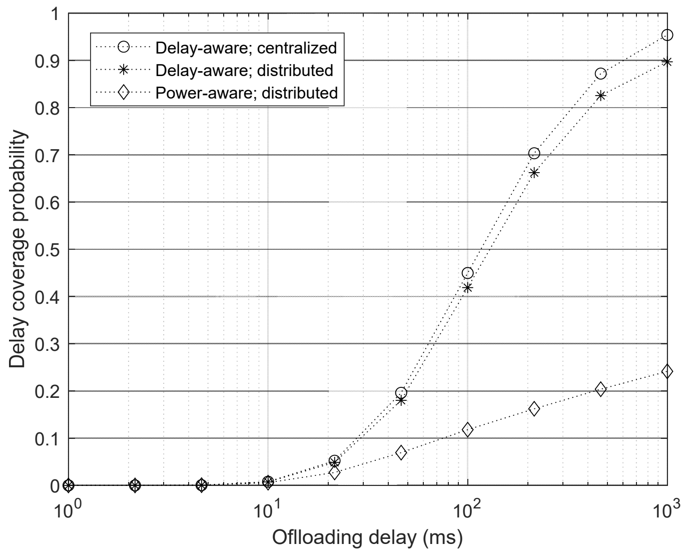


Figure 19. Comparison of different task association approaches with limited AP and CS capacities as function of target offloading delay.

2.5.4 Conclusions

In this subsection, delay-aware computation task association methods are studied under a networked computing framework where both communication links and server computation affect the end-to-end delay. It is demonstrated how delay-awareness can be beneficial in comparison with power-awareness with regards to delay coverage performance. Moreover, the performance relationships between centralized and distributed association methods are illustrated. In future work, the inclusion of computation allocation methods for each task will be examined to further reduce the overall delay of edge servers.

3 Architectural Techniques for Distribution of Intelligence

The overall goals of DEDICAT 6G include transforming beyond 5G networks into a smart connectivity platform. The platform needs to be highly adaptive, ultra-fast, and dependable/resilient for supporting securely innovative, human-centric applications tentatively by combining communication infrastructure with the provision of computation and storage resources at the edge to allow for the flexible realization of the envisaged perceived very low latency, ultra-fast response time, energy-efficiency, high productivity, and low cost. The projected numerical key performance indicator improvements are hard to achieve via the old/current means of increasing the clock rate of processing units, speed of communication links and incremental modifications to the methodology due to the complexity of the networked system and approaching fundamental limits of computation and communication technology [42][45]. The most promising methods left are architectural and algorithmic improvements addressing the performance, productivity, utilization and energy-efficiency issues of computation and communication in the network.

This section summarizes the work being done for *distribution of intelligence* (DoI) via investigation of low-level architectural techniques and engines utilizing them as well as employing the higher-level algorithms and conventions discussed in the previous sections of this document and WP2 deliverables.

3.1 Introduction

Intelligence can be characterized here as any computation and related communication that changes the state of the network in a meaningful way.

From a point of view of computation, the network can be seen as an entity consisting of a high number of computing nodes connected together via communication links. The nodes are not homogeneous but anything from high-performance data centers and edge servers down to user equipment and miniature IoT computing devices. Some of them have a more complicated internal architectural structure, featuring multiple processor cores, CPUs, processor cards and clusters of them with internal communication channels, as well as memory and input/output devices. Also, communication links are typically heterogeneous. In addition, neither the computational workload of the network nor its hardware components are constant or homogeneous but alter more or less frequently all the time, depending on the prevailing user data traffic distributions and acts of service providers. Let us call computation here *parallel* if it happens simultaneously in multiple units within a node and *distributed* if multiple nodes are involved.

In order to get the best performance out of the network, the computation needs to occupy intra-node units and be distributed among the network nodes so that the overall computing capacity would be maximized and the resource usage, e.g., time, energy, design effort, building and operating costs, would be minimized while ultimately supporting the designated use cases. A typical network node alone has more than a thousand design parameters not to mention the plurality of nodes and fitting the subsets of the computation into them. It is therefore evident that determining the optimal distribution is a very demanding multi-target *nondeterministic polynomial* (NP) complete problem, which cannot be solved with the current technology [33]. The current 5G systems approach distribution by relying on simple offloading functionality from UE to the network, virtual machines allowing flexible allocation of resources in the cloud, containers reducing the state of computation, multicore CPUs allowing (constrained) parallel processing and a programming paradigm relying mostly on independent sequential components and asynchronous execution of threads.

To boost the 6G network-level architectural design decisions, the possibilities of low-level architectural techniques and integration of them to entities are considered. For flexible and efficient placement of computation, an architecture utilizing the following techniques, such as (i) swift context switching, (ii) specific patterns of computation and communication, (iii) balancing the computational and communication load, (iv) moving computation, (v) reducing the state of computation, (vi) cost-efficient synchronizations, (vii) easy programmability and (viii) efficient placement of computation, overcoming the limitations of 5G network and processing solutions could be potentially applied (see Figure 20).

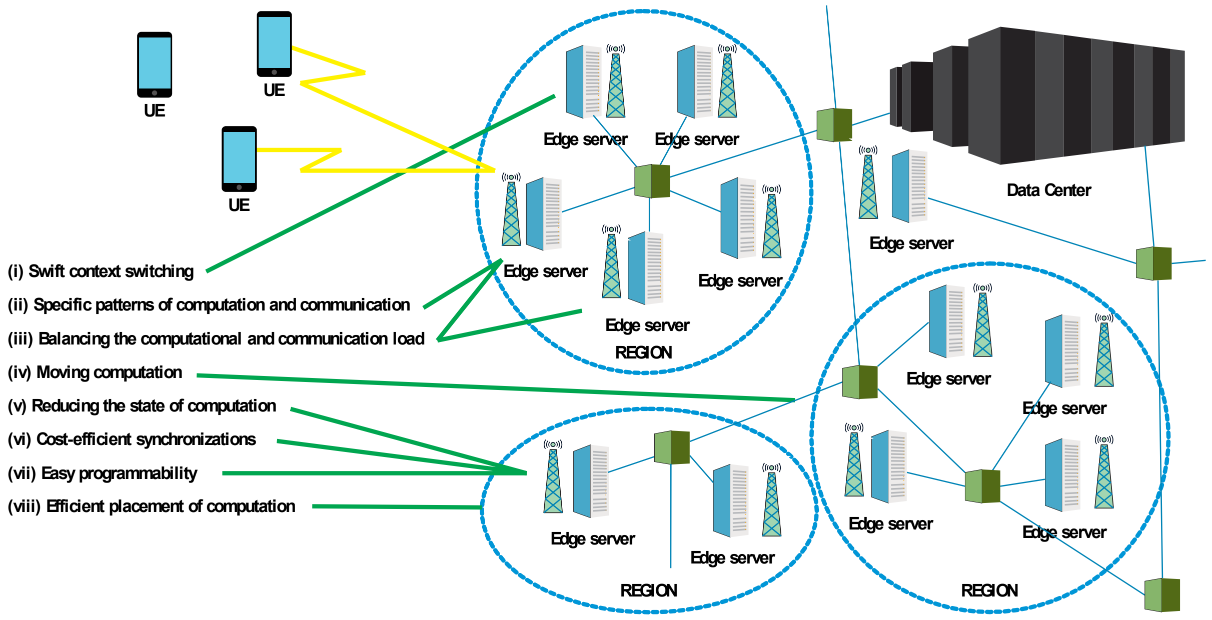

UE
UE
UE
Edge server
Edge server
Edge server
Edge server
Edge server
Edge server
REGION
Data Center
Edge server
(i) Swift context switching
(ii) Specific patterns of computation and communication
(iii) Balancing the computational and communication load
(iv) Moving computation
(v) Reducing the state of computation
(vi) Cost-efficient synchronizations
(vii) Easy programmability
(viii) Efficient placement of computation
Edge server
Edge server
Edge server
Edge server
Edge server
Edge server
REGION
REGION

Figure 20. Architectural techniques for distribution of intelligence.

In the following subsections, we discuss about using VTT's parallel processor framework and the ideas presented in this project to form a powerful edge processor. We go through preliminarily at low level, how such processor would perform against the existing industry standard solutions. The commercial exploitation of this technology will be investigated as a part of our effort (involving likely an external industrial partner) to make it gradually available in the markets.

Higher-level improvements are searched also from an Orchestration Engine that will serve as an interface between the DEDICAT 6G Decision-Making Functional Components and the NFV Orchestrator to properly instantiate ad-hoc network slices and assist the Decision-Making Functional Group (FG) in the application of the instantiation, scaling, or migration network service/slices orchestration procedures according to the outputs generated by the algorithms in this FG.

3.2 Discussion on Edge Processor Integrating the low-level Architectural Techniques

In order to solve the performance, resource efficiency and usability problems behind the low-level architectural techniques listed above, discussed in more detail in D3.1 (cf. [6]) and meet the ambitious DEDICAT 6G KPI targets T 6G, let us consider utilizing VTT's multiprocessor framework [41] and the findings of this project to outline an efficient edge processor architecture.

A processor utilizing such architecture could be used as a key computing component for building efficient networked computing devices, such as edge servers and cloudlets, supporting efficient distribution of intelligence. VTT's aim is to study exploitation possibilities of the edge processor as a part of larger commercialization effort of general-purpose parallel processor IP funded by VTT and Business Finland. One concrete possibility here is to find a spin-off company to take care of further development and bring processor products to the market with suitable industrial partners since manufacturing leading edge processors is highly expensive and requires a lot of manpower and expertise.

The *Thick Control Flow Processor Architecture for Edge* (TPA-E) is a scalable multiprocessor architecture that can be configured at design time for various constellations lending widely solutions from the original TPA [39] and VTT's processor framework [41] aiming generally at addressing the performance and programmability issues of current general-purpose multicore architectures. The framework defines principles how to build efficient *Emulated Shared Memory* (ESM) processors utilizing the *Thick Control Flow* (TCF) abstraction; how to make them flexible and expandable with accelerators; how to achieve backwards compatibility with existing commercial product lines; includes a number of processors, interconnect and memory system architectures and designs needed for that purpose and outlines the methodology to develop program. A TCF is an abstraction of parallel computation that merges self-similar threads into a single computational entity that is independent of the number of threads [44]. Self-similarity refers here to properties of flowing through the same control path and having homogeneous operations. We call the component threads of a TCF *fibers* to distinguish them from ordinary threads having their own control. The fibers within a TCF are executed synchronously with respect to each other in order to simplify parallel programming. In ESM, the latency of the memory system is hidden via multithreading and sufficient bandwidth, the synchronization cost is virtually eliminated using wave synchronization and low-level parallelism exploitation is optionally improved by chaining of *Functional Units* (FU) assuming there is enough parallelism in the functionality at hands [46][43][35]. Instead of multithreading, TPA-E uses a similar technique for fibers, called interleaved multifibering. This lets a fiber to execute other fibers while it is making a memory reference. If the executed program contains enough fibers the latency of the shared memory system can be completely hidden. Sufficient interconnection bandwidth is provided by using an *M*-way multimesh network. The synchronization wave is used to separate memory references belonging to consecutive steps of execution by issuing all fibers followed by a synchronization message. Synchronization messages are routed in the network through all possible paths so that when a synchronization message arrives to a router, it blocks the message (and related paths) until a synchronization message can be found in all inputs. Then the router fetches all the incoming messages and sends out a synchronization message via its outputs. Low-level parallelism is supported in TPA-E by organizing FUs as a sequential chain rather than in parallel so that consecutive instructions can be executed regardless of possible interdependencies within a step.

A TPA-E multiprocessor consists of *F Frontend* (FE) processing units and *B Backend* (BE) processing units, intercommunication networks and a memory system (see Figure 21). FEs take care of fetching instructions from the memory and executing the common parts of TCFs, such as control of the flow and base address computation. In turn, BEs handle execution of individual fibers. The memory system consists of two parts: Depending on the FE architecture of choice, FEs are connected to either a traditionally organized *Symmetric MultiProcessor* (SMP) or *NonUniform Memory Access* (NUMA) memory system and BEs are attached to an ESM system employing a multimesh interconnect. The latter supports synchronous operations and parallel-computing specific access patterns such as concurrent reads and writes, reductions and multiprefix computations as well as powerful atomic compute-update operations. The FE and BE parts of the memory system are also connected together.

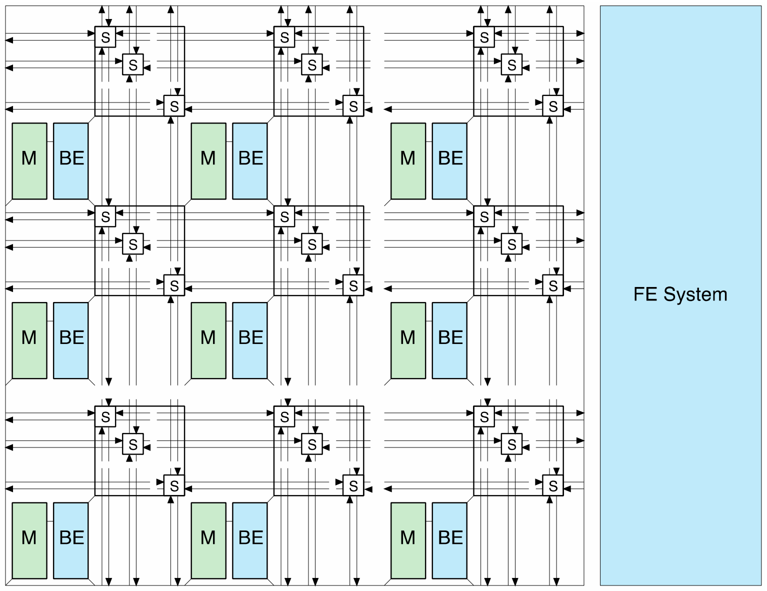


Figure 21. High level organization of TPA-E (FE=frontend, BE=backend, M=memory, S=switch).

A TPA-E FE system resembles an ordinary multicore CPU and in fact the VTT's multiprocessor framework allows a processor designer to use a variant of existing commercial CPU core as FE. In this project, however, we use the framework default, the *Minimal Pipeline Architecture* (MPA) *Very Long Instruction Word* (VLIW) processor [34] as the FE architecture. MPA features a number of FUs commanded by dedicated sub-instruction fields in a single (compound) instruction word. The original version of MPA features a special minimal pipeline with only two stages—fetch and execute with no pipeline delays in the case of control transfer. There are some modifications to the pipeline to allow multi-TCF operation but delay-free operation of the pipeline also for control operations was retained.

A BE is a special processing unit resembling a MultiBunched/Threaded Architecture with Chaining (MBTAC) ESM processor core [40] and containing logic for operand selection, chain of FUs, latency compensation unit and write back logic. However, a TPA BE does not include an instruction fetch unit and sequencer. In TPA-E these belong to the FE system.

For efficient edge computing, TPA-E is aimed to support distribution of intelligence in both processor and network edge region levels.

Here we present a high-level view of the processor only since TPA-E utilizes VTT's multiprocessor framework not belonging to this project. In addition, VTT aims to commercialize its processor technology and wants to protect the low-level IP independently of this project.

To figure out, how well TPA-E compares to current commercial solutions and addresses the problems behind the low-level architectural techniques of D3.1, we wrote a number of parallel test kernels in C/pthreads and equivalent TCF-aware programming language, performed execution time and program code length measurements on two industry standard processors (4-core Intel Core i7 6820HQ and 18-core Xeon W 2191B) and an entry-level 1-FE 16-BE TPA-E. TPA-E execution time was measured with a help of TCF simulation software of VTT's processor multiframework. Since there is no silicon implementation of TPA-E, we assumed that it would run at the same 3.2 GHz as Core i7 and Xeon W. The measurement included here represent processor/computer level solutions whereas region and network level effects are aimed to be studied for D3.3. In the following we present results of the measurements grouped similarly as in D3.1:

Swift context switching — Context can be defined as the minimal set of data used by a computational task that must be saved to allow task to be interrupted and later continued from the same point. Swift context switching is important to support multitasking, i.e., running multiple computational functionalities in the same processor. In principle, all CPUs capable of executing multitasking operating systems supporting switching of threads but the latency of thread switching is typically a few hundred clock cycles. This is not a problem when executing independent threads that are switched infrequently enough. However, if the threads of the functionality at hands require dense intercommunication, the performance can be catastrophically poor as indicated by our measurements of memcopy kernel as a function of the number of threads on Core i7 and Xeon W processors (see Figure 22). The behaviour of both processors is similar—the execution time decreases as the number of cores increases until there is one thread per processor core. After that, the execution time stays the same until there are two threads per core since Intel processors feature a two-way multithreading called hyperthreading. Finally, as the number of threads exceeds the number of hardware threads the execution time instantly jumps up by almost three orders of magnitude. This is caused by thread switching time that cannot anymore be hidden with hardware threads, synchronization costs and especially the operating system scheduler, which allocates a way too long time slice for each thread performing the final synchronization—the only place where there are dependencies in the memcopy kernel. For comparison purposes we also show the execution time level of TPA-E performing the same task with a single TCF having maximal parallelism. It avoids this problem with its zero-switch latency multifibering mechanism and allows a programmer to easily get the full performance out of the hardware.

Specific patterns of computation and communication——Patterns of parallel and distributed computation and communication refer to situations where multiple computational threads interact in a regular way that can be seen as a pattern. The most popular patterns include parallel execution, reduction, spreading and permutation. These are used, e.g., in parallel processing and communication, collection of data, multicasting as well as in certain mapping tasks.

As a part of our development work, we studied five alternative techniques for efficient multioperations in TPA—a fast single-instruction multioperation (FS), a symmetric two-instruction multioperation (S2), a backend-frontend multioperation (BF), an optimized two-instruction multioperation (O2) and a multioperation load (ML). For this WP, we measured the execution time of memory-to-memory reduction patterns in TPA-E supporting the variants and compared the results to sequential algorithm without any of these operations.

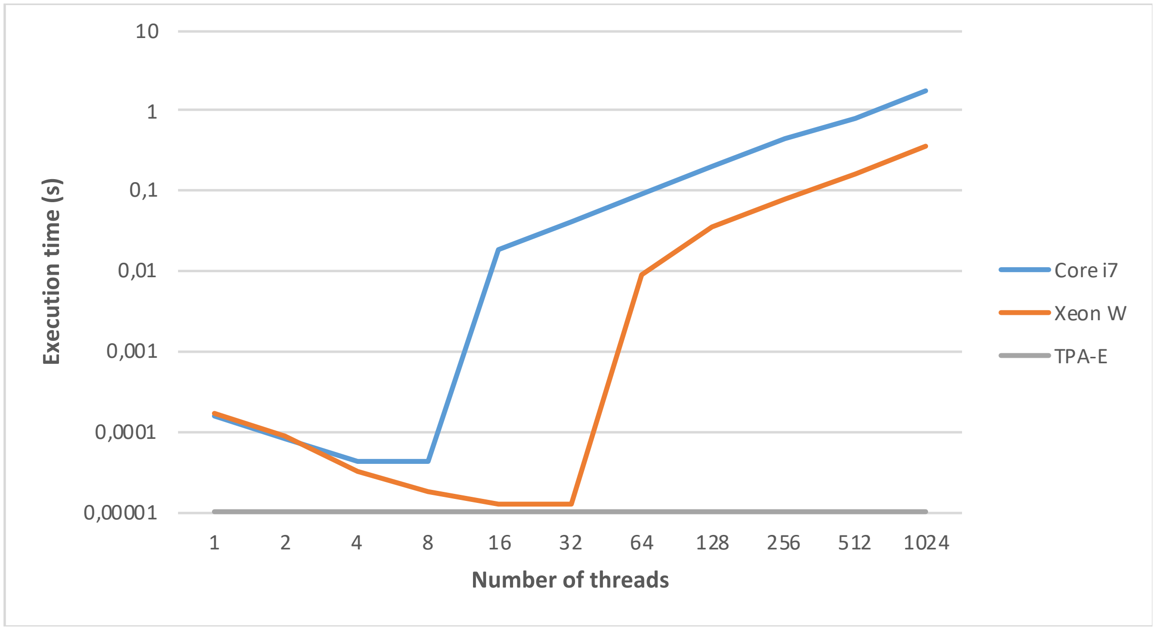


Figure 22. Execution time of blocked version of the memcopy benchmark (log scale).

According to our measurements, the multioperation techniques can speed up execution of memory-to-memory reductions by a factor of 15.57 w.r.t. sequential execution and by a factor of 43.82 w.r.t. 16-processing unit baseline machine (see Figure 23).

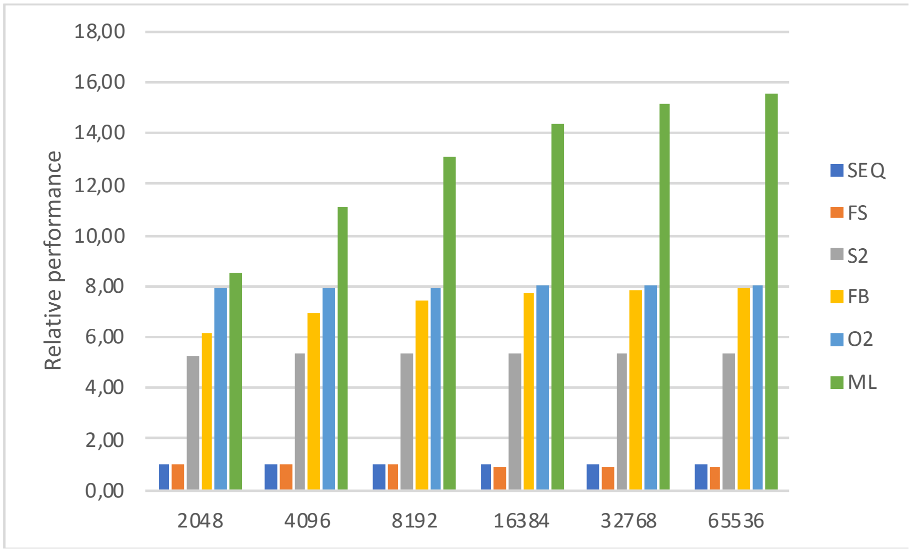


Figure 23. Relative performance of a multioperation reduction in TPA-E as a function of the input data array size.

To demonstrate how well practical low-level patterns are supported in TPA-E with respect to available resources, we determined the fraction of the memory bandwidth for six memory access patterns—blocked, interleaved, random, matrix multiplication row style, matrix multiplication column style and concurrent memory access. The results, comparing TPA-E to Core i7 and Xeon W, are shown in Figure 24.

According to the measurements, Core i7 and Xeon W utilize their memory bandwidth only for blocked access pattern while TPA-E is able to retain virtually the maximum bandwidth regardless of the access pattern. The fact that 18-core Xeon W shows significantly weaker results than 4-core Core i7 may indicate that there is a serious intrinsic scalability problem in current architectural approach of multicore processors and systems built on top of them as we have been predicting [38].

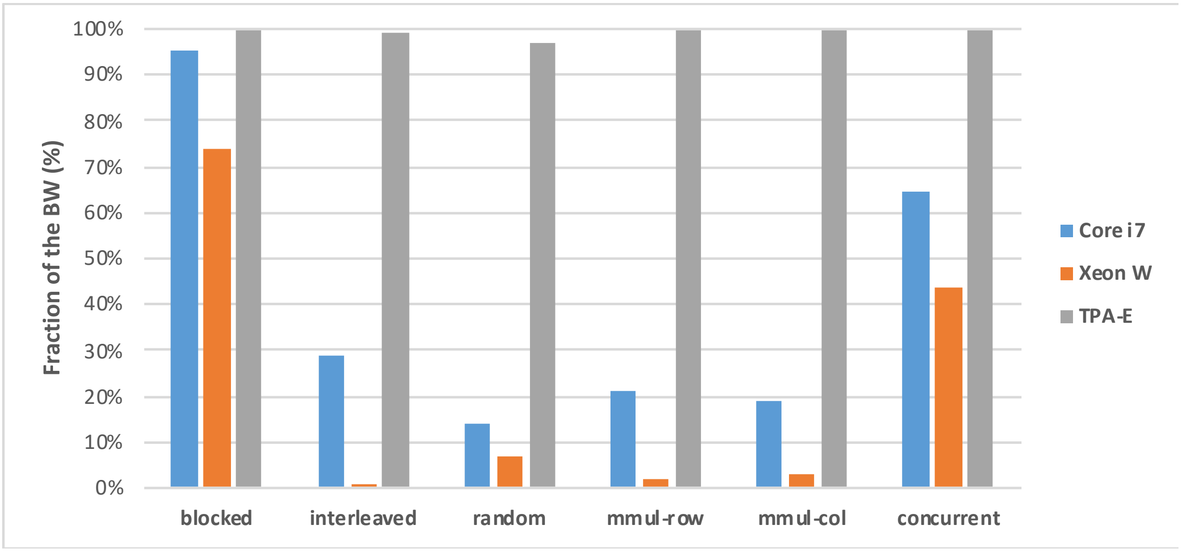


Figure 24. Fraction of the bandwidth utilized for shared memory access patterns.

Cost-efficient synchronizations—Synchronization is the key mechanism to ensure the correct behaviour of parallel and distributed software at hands in the case of inter-thread dependencies. Unfortunately, in current multicore systems the cost of synchronization can be very high. The main reason for this is the asynchronous nature of execution in multicore CPUs, computers with multiple processor sockets, clusters of computers and especially in the network. A notable fact is that the need for fast and efficient synchronizations is much more stringent in fine- grained parallel computing than in coarse-grained distributed computing that is not sup- posed to be able to execute fine-grained parallel algorithms efficiently.

We compared the execution time of barrier synchronization in Core i7 and Xeon W processors to wave synchronization employed by TPA-E and observe that TPA-E synchronizes at least 1000 times faster for the cases in which there is large enough number of threads for full performance (see Figure 25). In addition, the synchronization cost in Intel processors makes a jump of multiple magnitudes up as the number of hardware threads is exceeded.

Easy programmability—Programmability is said to be good if the functionalities can be expressed compactly and naturally without unnecessary architecture-dependent constructs. A key factor is also portability and ability to retain performance with respect to the number of execution units among a family of processors using the same approach but having a different hardware configuration. The main challenges of current systems include the asynchronous nature of execution and sensitivity to non-trivial memory access patterns. Distributed systems, such as regions of edge servers, pose further challenges to programmability since the latencies are much higher, and throughputs lower than those within parallel machines. Programmability is directly proportional to productivity of software development, and thus cost of the software. TPA-E solutions for this come from the ESM architecture and TCF abstraction.

The TCF versions the kernel functionalities are written as a single TCF, maximally parallel, synchronous programs utilizing available primitives of parallel computing where relevant. There is no need for explicit synchronizations in the tested TCF algorithms. Consider three alternative ways of implementing functionalities in parallel for Intel processors: *The straight-forward*

Pthreads versions are written similarly as the TCF versions except that synchronizations are added to the end to be able to determine when all the threads have completed their tasks and wherever they are needed to guarantee the correct execution order of operations assigned to different cores. The *matched parallelism* versions limit the number of threads to a given maximum, which in our case is the number of processor cores *P*. The matching is done by employing loops that process at most *P* elements (and threads) at the time. We expect matched parallel versions to be substantially faster than straight-forward ones. This is because matching eliminates interference between time slots defined by the operating system scheduler and actual computation as well as the thread management overhead, especially in the case of fine-grained parallel functionality. The *blocked* versions divide the processed data elements to blocks that are executed in the processor cores in parallel. This kind of mapping and implied partitioning should also improve the performance over the matched parallelism versions due to increased locality and reduced inter-processor communication.

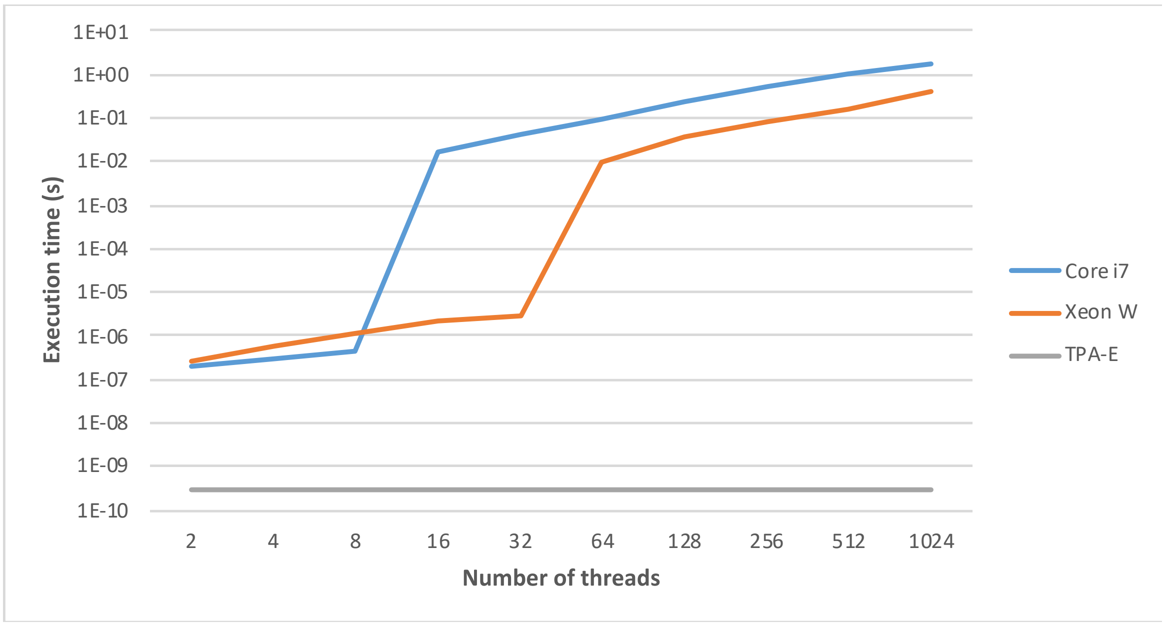


Figure 25. Execution time of a barrier synchronization (log scale).

In order to measure the complexity of programming, we implemented three versions of matrix addition algorithm A:=A+B for Intel multicore CPU systems with C/pthreads and a single TCF version for a system utilizing ESM and TCF with a C/pthreads-style parallel language. From the programs, we determined the number of active code lines. Three program versions for the Intel system were included since the simplest straight-forward pthreads version interferes in a very ugly way with the operating system scheduler and gives 27.8 million times slower execution time in Core i7 and 6.1 million times slower execution time in Xeon W than in TPA-E. The matched parallel pthreads version in Core i7 and Xeon W gave 14.5x and 99.1x slower performance than in TPA-E, respectively. Finally, the blocked version comes closer to TPA-E performance. In Core i7 it executes 4.5 times and in Xeon W 1.2 times slower (or 36% per processor core slower) than that in TPA-E.

Figure 26 shows implementations of as active code lines. Note that the number of active code lines for pthreads algorithms increases as the execution time decreases and that the ESM version is 2, 3 and 6 times shorter, respectively. In addition, pthreads needs initialization, thread creation and termination code. Note also that the trade-off between the performance and software complexity in Intel processors that forces programmers to employ complex and error-prone programming techniques to have a decent performance whereas TPA-E is not affected.

// madd Intel straight-forward pthreads

```
i   pthreads_attr_init(&attr);                        Initialization and thread
ii  pthreads_attr_setdetachstate(&attr,               creation needed.
      PTHREADS_CREATE_JOINABLE);
iii for (t=0; t<NUM_THREADS;t++)
iv       rc=pthreads_create(&thread[t],
              &attr,Add_Array,(void *)t);
```

```
1 A[id]+=B[id];         Core i7: 27800000x       Textbook algorithm
2 Synchronize;          Xeon W:  6100000x        Synchronizations needed.
                        slower than TPA-E
```

```
i   pthreads_attr_destroy(&attr);                     Termination needed.
ii  for (t=0; t<NUM_THREADS;t++)
iii      rc=pthreads_join(thread[t],&status);
```

// madd Intel matched parallel pthreads

```
i   pthreads_attr_init(&attr);                        Initialization and thread
ii  pthreads_attr_setdetachstate(&attr,               creation needed.
      PTHREADS_CREATE_JOINABLE);
iii for (t=0; t<NUM_THREADS;t++)
iv       rc=pthreads_create(&thread[t],
              &attr,Add_Array,(void *)t);
```

```
1 for (id=tid; id<N; id+=NUM_THREADS)                 Textbook algorithm leads to
2   A[id]+=B[id];                                     poor performance. Instead
3 Synchronize;          Core i7: 14.5x                limit parallelism
                        Xeon W:  99.1x                Synchronizations needed.
                        slower than TPA-E
```

```
i   pthreads_attr_destroy(&attr);
ii  for (t=0; t<NUM_THREADS;t++)
iii      rc=pthreads_join(thread[t],&status);         Termination needed.
```

// madd Intel blocked pthreads

```
i   pthreads_attr_init(&attr);                        Initialization and thread
ii  pthreads_attr_setdetachstate(&attr,               creation needed.
      PTHREADS_CREATE_JOINABLE);
iii for (t=0; t<NUM_THREADS;t++)
iv       rc=pthreads_create(&thread[t],
              &attr,Add_Array,(void *)t);
```

```
                                                      Textbook algorithm leads to
1 blocksize=SIZE/NUM_THREADS;                         poor performance. Instead
2 start = tid*blocksize;                              divide data into core-wise
3 stop = start + blocksize;                           blocks and process them in
4 for (id=start; id<stop; id+=gap)                    parallel.
5   A[id]+=B[id];
6 Synchronize;          Core i7: 4.5x                 Explicit loop and
                        Xeon W: 1.2x                  synchronizations needed.
                        slower than TPA-E
```

```
i   pthreads_attr_destroy(&attr);                     Termination needed.
ii  for (t=0; t<NUM_THREADS;t++)
iii      rc=pthreads_join(thread[t],&status);
```

// TPA-E version

```
                                                      No initialization nor
                                                      component creation
                                                      needed.
```

```
                                                      Straight-forward textbook
1 A_[$]+=B_[$];                                       algorithm Full performance
                                                      via a parallel statement. No
                                                      loops, blocking nor explicit
                                                      synchronizations needed.
```

```
                                                      No termination needed.
```

Figure 26. Parallel matrix addition kernel for Intel CPUs and an TPA-E.

Efficient placement of computation—The best performance is achieved when the right data is in the right place at the right time since moving both data and computation, i.e., execution of operations take time. Additional complications come from the fact that the farther away data is from the place where it is needed, the longer time it takes to obtain it and the more dependencies there are, the longer it takes to execute if there are resource limitations. Additional complications can come from possible contention of traffic in the network caused by non-optimal placement of data and functionality in the network, reliability issues potentially requiring resubmissions, protocol issues, deadlocks, livelocks, race conditions, sequentialization, physical defects, noise etc.

Current multicore systems are highly sensitive to data and functionality placement. These phenomena are augmented in the distributed computers such as cloudlets and regions of edge servers due to high latencies and limited bandwidth.

We measured the execution time of the matched parallelism and blocked versions of the memcopy program as a function of the number of threads in systems with 4-core Intel Core i7 and 18-core Xeon W processors. Both processors feature two way-multithreading (or hyperthreading as Intel calls it) where a processor core can execute up to two threads without typical 100+ clock cycle thread switching overhead.

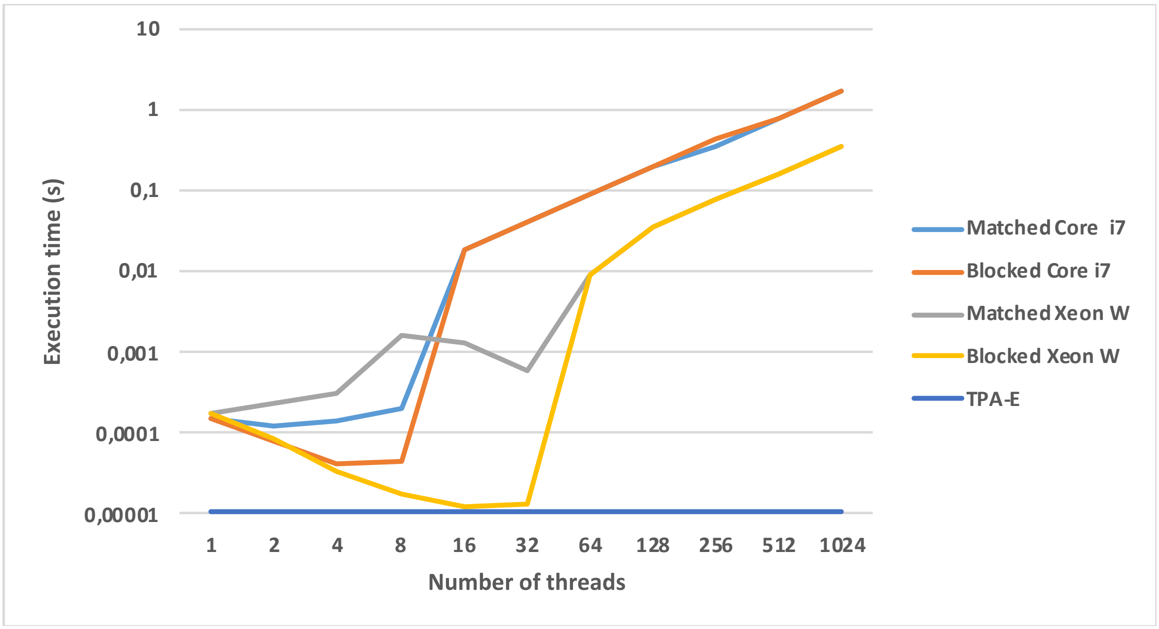


Figure 27. Execution time of matched parallel and blocked versions of the memcopy benchmark (log scale).

We figured out how these improvements in low-level architectural techniques show up as performance and software complexity in applications utilizing the patterns of parallel and distributed computation and communication. For Intel processors all three programming styles (straight-forward, matched parallel and blocked) are included while the TPA-E versions utilize just one TCF. The following six Figures present the results of our initial performance and code length measurements.

According to our measurements, TPA-E executes straight-forward benchmarks in average 56.2 million times faster than Core i7 and 11.9 million times faster than Xeon W with three times shorter programs (counted as active program lines). These massive speedups are caused by the joint effect of slow synchronization and context switching as well as operating system scheduler allocating a way too long slices for the threads. In matched parallel versions the speedups for TPA-E drop to 26.6x and 164x, respectively. The Intel processor program length overhead increases to 3.5x. These programs eliminate the extremely slow context switching in barrier synchronizations but suffer from last level cache line sharing. Finally, in blocked tests, TPA-E is in average 9.32 times faster than Core i7 and 3.35 times faster than Xeon W with one sixth of the active program lines.

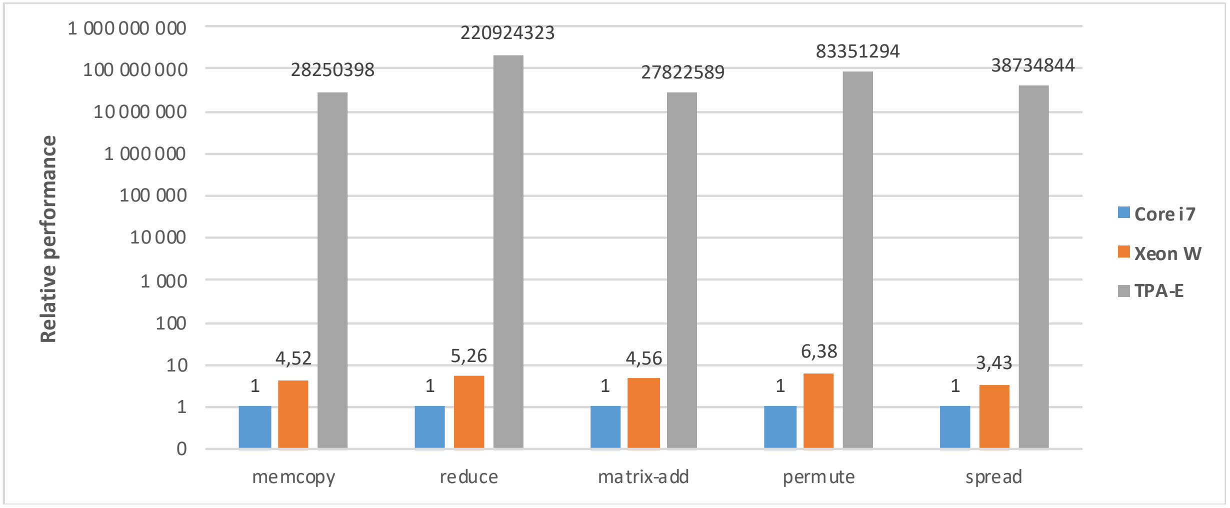


Figure 28. Relative performance of straight-forward parallel versions of the kernels (log scale).

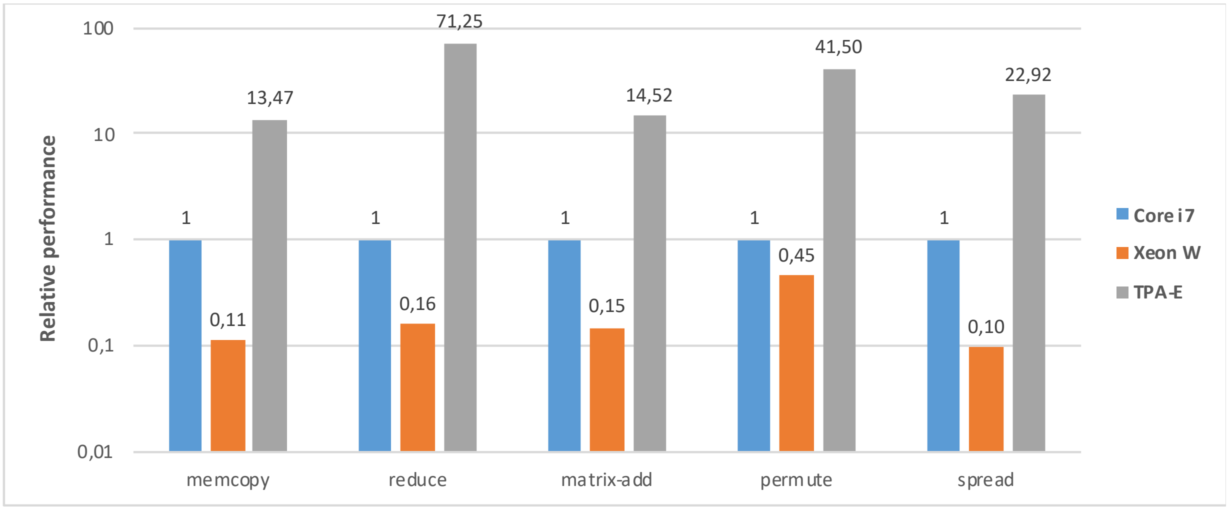


Figure 29. Relative performance of matched parallel versions of the kernels (log scale).

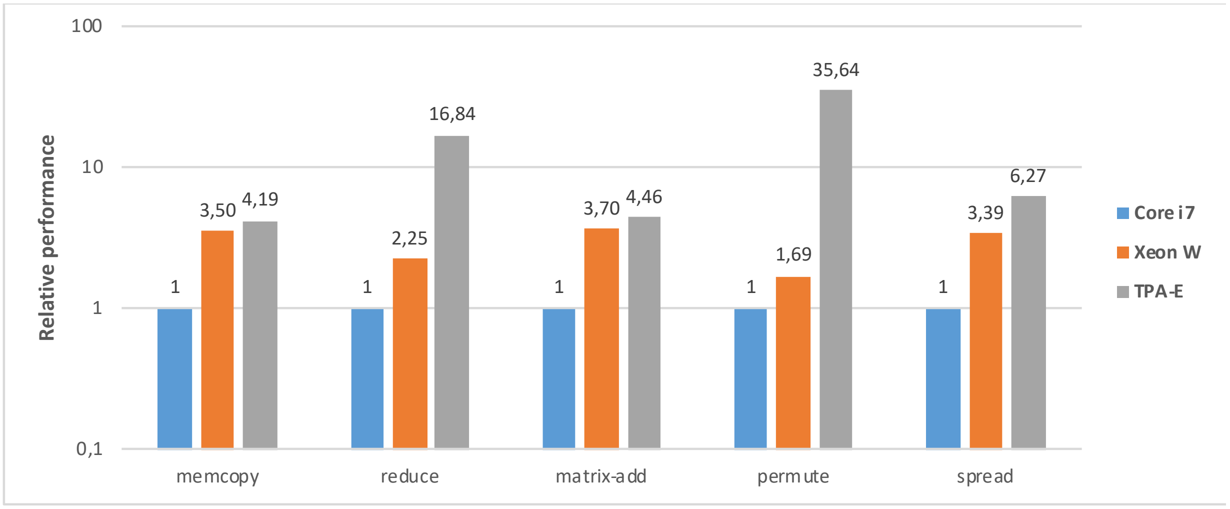


Figure 30. Relative performance of blocked versions of the kernels (log scale).

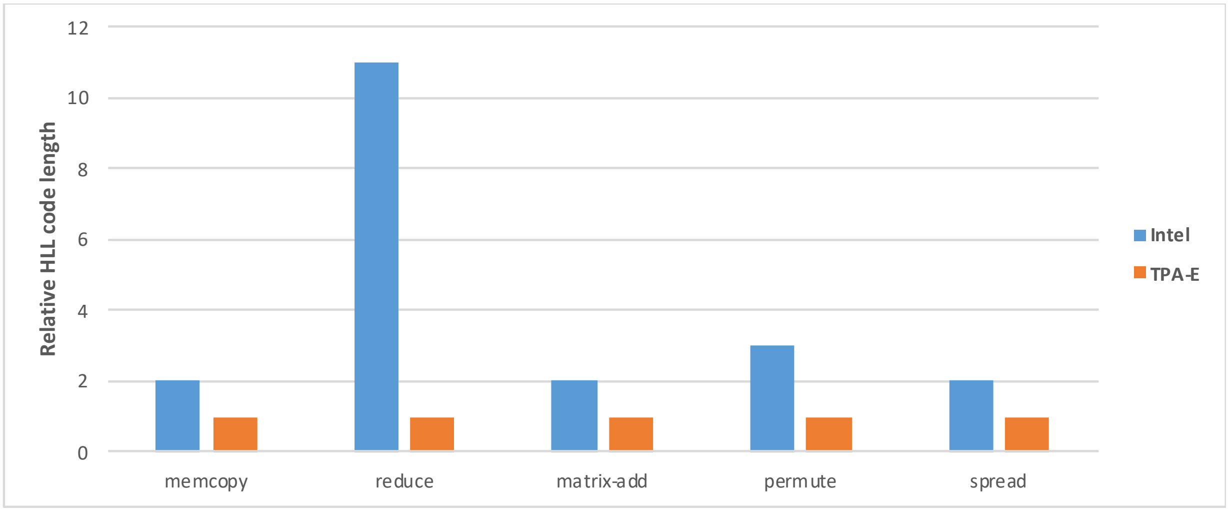


Figure 31. Relative active code line count of straight-forward versions of the kernels.

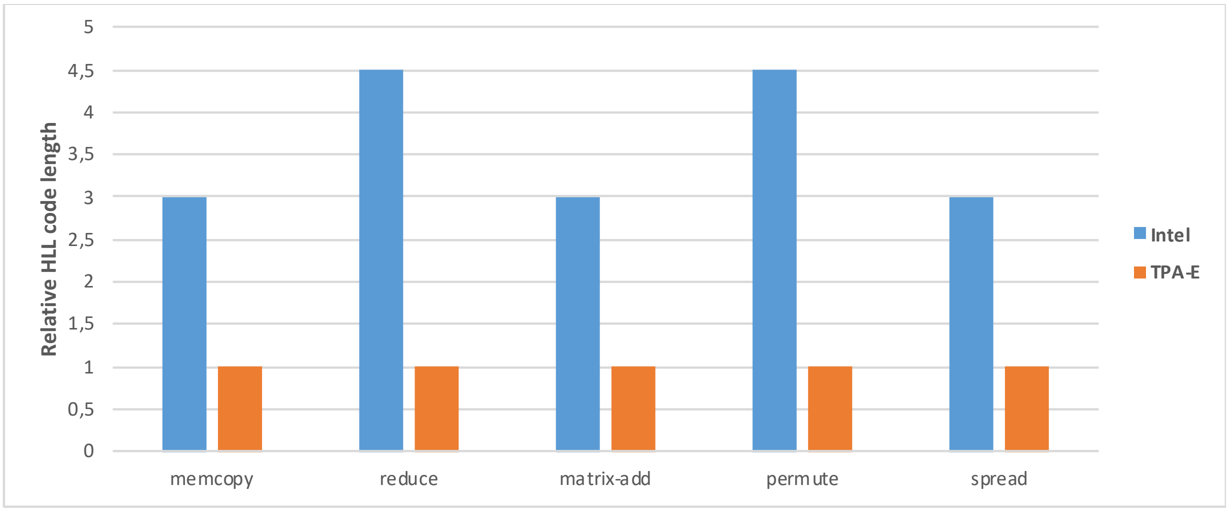


Figure 32. Relative active code line count of matched parallel versions of the kernels.

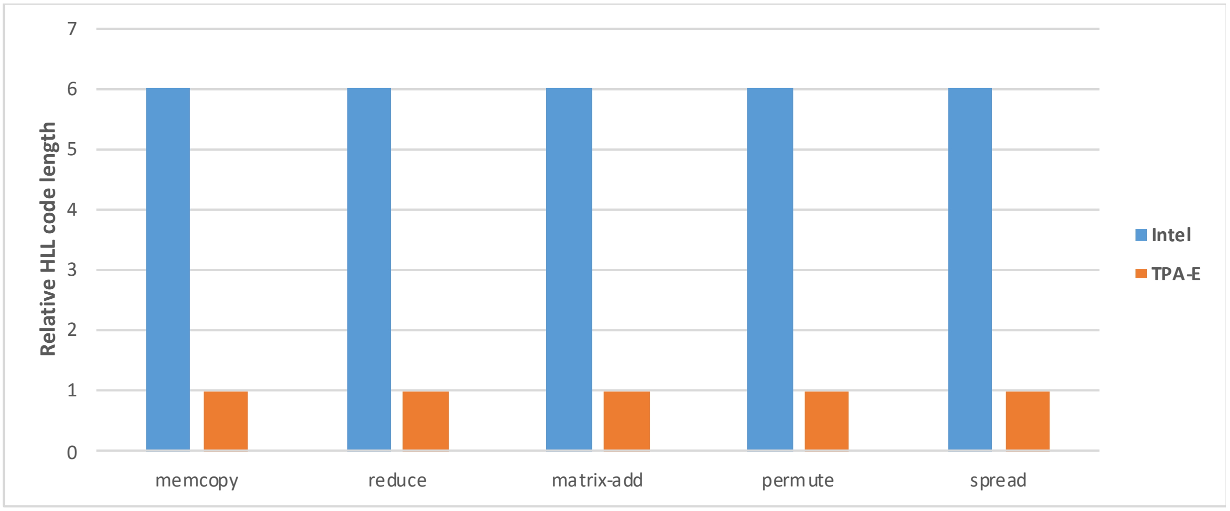


Figure 33. Relative active code line count of blocked parallel versions of the kernels.

Based on these initial measurements with an entry-level configuration of TPA-E, it seems preliminarily possible that a processor like TPA-E, integrating the low-level architectural techniques listed in D3.1 and VTT's multiprocessor framework, could make it possible to build networked computing devices providing substantial speedup and giving a substantial software development productivity boost with respect to industry standard solutions. This would help to in part achieve the 10x latency improvement and efficiency targets of DEDICAT 6G.

3.3 Orchestration Engine

The orchestration engine will be part of the (Service) Orchestration FC defined in WP2's architecture and will serve as an interface between the Decision-Making FCs and the NFV Orchestrator to properly instantiate ad-hoc network slices, thus enabling the provisioning of a unique 5G/B5G-based virtualized network space to run vertical's apps under the requested conditions. Moreover, the orchestrator engine will assist the Decision Making FG in the application of the instantiation, scaling, or migration network service/slices orchestration procedures according to the outputs generated by the algorithms in this FG.

To play such role within the DEDICAT6G platform the orchestration engine must receive instructions and concrete information coming from several internal FCs and with the external NFV Orchestrator. The overall functional overview and their interactions are represented in Figure 34 and summarized as follows:

- **CEDM FC:** It is the FC responsible to take decisions about coverage extension. Its output will feed the orchestration engine with information about the existing MAPs deployment where the network provisioning is required.
- **IDDM FC:** The IDDM will send to the orchestration engine some recommendations to ensure optimality in the management of the computational resources consumed by the NFV-related intelligence (e.g., VNFs).
- **NODM FC:** This FC will assist the orchestration engine to properly configure the network slices by providing network-related information.
- **µS/FC Registry FC:** It can serve information about the computational requirements of to either µS or FC when needed.
- **µS/FC Repository FC:** The orchestration engine can access this repository when information about the container/VM images of the FC/µS is required.
- **NFV Orchestrator:** This external entity is the one responsible to manage the NFV infrastructure (NFVI) resources to establish network slices. The orchestration engine will command the NFV-O according to the decisions made within the DEDICAT6G platform.

For the interest of WP3, in this deliverable we will put the light on the parts related to the orchestration of the intelligence and computational resources in the system to be managed by the orchestration engine to configure and instantiate network slices, not the networking and other configuration mechanisms. However, it is worth mentioning that the nature of the orchestration engine component tackles both WP3 and WP4 domains, thus, some parts in D3.2 and D4.2 are common in both documents to clearly understand this work in a standalone way. See D4.2 for more information on the orchestration engine from a NFV perspective to complement the vision provided in this deliverable.

Furthermore, it is expected that this orchestrator engine will expand its functionalities to enable orchestration mechanisms related to the communication with the Edge Orchestrator (e.g., Kubernetes) in a similar way than with the NFV Orchestrator.

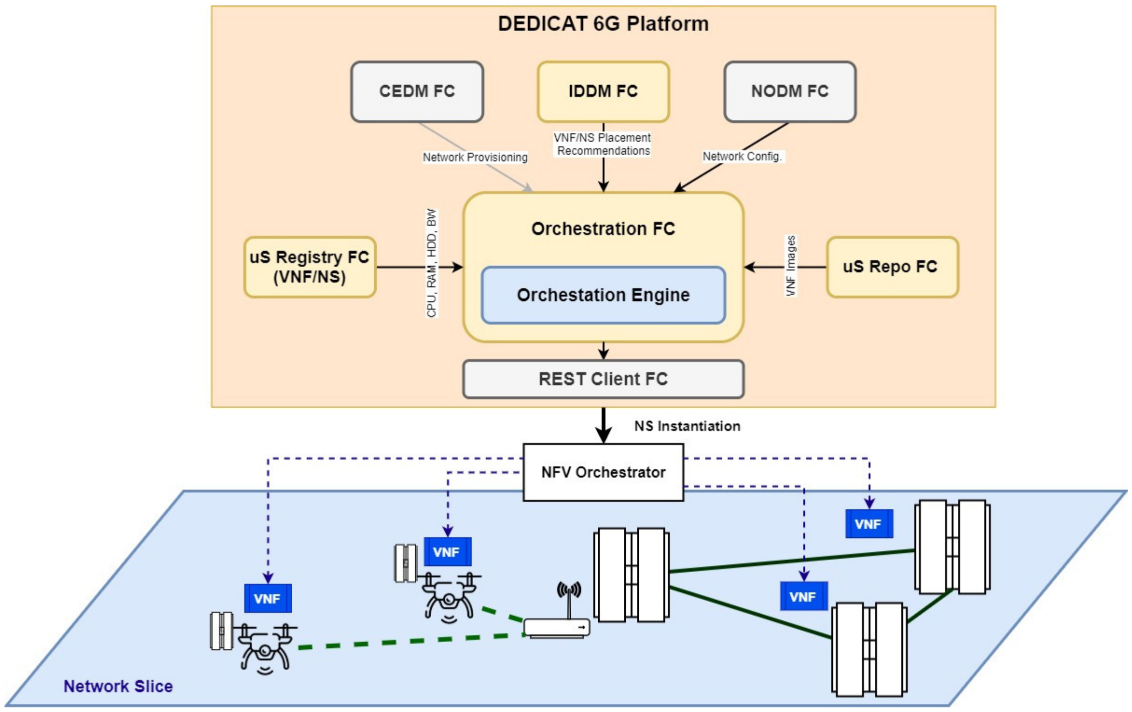


Figure 34. Orchestration engine schema in the DEDICAT6G platform.

3.3.1 Orchestration Engine Design

In this work, we present a cloud-native modular design for the orchestration engine component ready to work in a distributed microservice environment. This design is focused to be implemented in a lightweight and scalable implementation suitable for deployment both in the cloud and at the edge domains. It is worth mentioning that this design is targeted to communicate with standards and trending external tools.

The Orchestration Engine is focused on assisting the Decision Making FG to translate and apply recommendations in the external 5G/B5G system, mainly focused on the NFV side by means of the NFV-O. Our design (shown in Figure 35) assumes the ETSI Open-Source MANO (OSM) as NFV Orchestrator. In this iteration, the orchestration engine design comprises 5 main sub-modules:

- **Engine Manager:** This module works with all the other components to provide the main functionalities of the orchestration engine and as a link of the rest of the modules. It is in charge of managing all internal modules in a coordinated way according to the requests coming from the Decision Making FCs.
- **Blueprint/Descriptor creator:** This entity is responsible for automatically creating or modifying descriptors based on incoming requests. A descriptor defines what is understood as a VNF, a network service and a network slice. These templates are based on the ETSI SOL006 data model [47], which are supported by OSM.
- **OSM Client:** This component is in charge of implementing OSM Client to be able to communicate with OSM through the native OSM API. It also implements some new features that are not currently available in OSM Client, for instance, post processing messages received by OSM to extract more valuable and concrete information to manage the instances which are running.
- **gRPC Server:** It exports the set of functionalities offered by the orchestration engine to the rest of FCs in the DEDICAT6G platform. Also, the gRPC-based server to enable rapid

and efficient communications channels among the different FCs. More information about gRPC and its benefits for microservice communications in the next subsection.

- **Cache DB:** This database will store temporal and persistent data, like information about the network and received messages, necessary for the orchestration engine tasks.

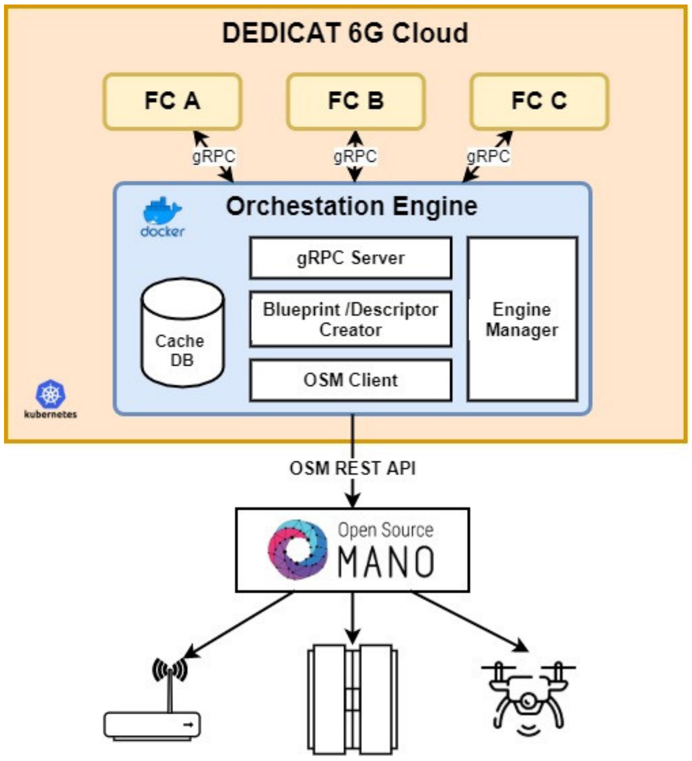


Figure 35. Orchestration Engine design

3.3.2 Experiment Set-up

For the purpose of implementing, testing, and validating the functionalities of the orchestration engine, details of a preliminary set of experiments aimed at that end are shown here.

3.3.2.1 Particular WP3 Goals

As exposed before, the orchestration engine is a component that implements some key functionalities of the WP2-defined Service Orchestration FC. The main objective of this experiment is to demonstrate the key functionalities of such component within a realistic B5G-driven environment under the DEDICAT6G umbrella. Despite, the main objective can be shared between WP3 and WP4, here we have defined a set of WP3-oriented sub-goals to be shown in this work:

- Demonstrate an agile and efficient microservice-based communication between the IDDM FC and the orchestration engine.
- Show a preliminary working implementation of some WP3-related FCs suitable to be potentially extended to the rest of the DEDICAT6G platform.
- Preliminary implementation of the IDDM FC output WP2-defined data model.
- Show the impact of the network slice instantiation in the intelligence distribution and computational resource consumption in the edge nodes of the system.

3.3.2.2 Related Technologies

We now introduce the set of open-source software tools employed to perform the corresponding functions presented in the orchestration engine design chosen to meet the abovementioned objectives:

- **Docker [48]:** Docker is an open-source tool that automates the deployment of applications running inside software containers, thereby bringing an additional layer of abstraction and automation to the virtualisation of applications. It is currently the most widely used tool in microservices-based deployments. In this work, Docker is used to create the containers where the different components are implemented and deployed.
- **Kubernetes [49]:** is an open-source platform for managing container deployment, as already presented in D3.1. The role of Kubernetes in this implementation is to manage and host the Docker containers related to the DEDICAT6G FCs and also to host the OSM instance.
- **gRPC [50]:** it is a high-performance, lightweight, and RPC-based communication protocol used in microservices environments and hosted by the Cloud Native Computing Foundation. A prominent feature of gRPC is that its data is structured using strict and lightweight rules defined by Protocol Buffers (Protobuf [51]), a framework for serving structured data. The data in Protobuf, at the same time as in gRPC, is specified and defined by an Interface Description Language (IDL). It can create language-independent interfaces to support communication between servers or clients written in different languages. The present work generates gRPC libraries to enable the communication among the DEDICAT6G FCs. Additionally, the messages of the DM-related FCs (outputs) have been implemented by using the data models defined in D2.4.
- **SQLite [52]:** this lightweight and fast SQL-driven database is the chosen one to represent the role of the cache DB in the orchestration engine.
- **OSM [53]:** Open Source MANO is the ETSI-hosted NFV orchestrator fully aligned with ETSI NFV standards. As explained extensively in D4.1, it is capable of managing and orchestrating virtualized resources in NFV-based ecosystems to enable the instantiation of 5G/B5G network slices. This is the tool selected for these purposes as NFV-O in the present work.
- **OpenStack [54]:** is one of the most popular open-source cloud infrastructures used to manage and host third-party services in virtual machines, bare metal and containers. In this implementation, OpenStack plays the role of a Virtual Infrastructure Manger (VIM) that represents an edge node with the computational resources to host the VMs of the VNFs in a network instantiation. In our case, we consider two different OpenStacks as follows in the next sub-section.

3.3.2.3 Implementation Schema

The trial has been carried out on the ATOS Telecommunications test bed, following the implementation scheme described in Figure 36. The testbed features a Kubernetes cluster (K8s) representing the cloud domain intended to host and control the Docker containers, in this case the DEDICAT6G FCs and the orchestration engine, all of them written in Python. In addition, in this same cluster, we have deployed an instance of OSM, playing the role of NFV-O. Finally, the edge domain is modelled with two Open Stacks (VIM) instances, physically separated, to consider two different edge nodes. We assume both VIMs as part of the NFV infrastructure to enable the establishment of multi-site network slices.

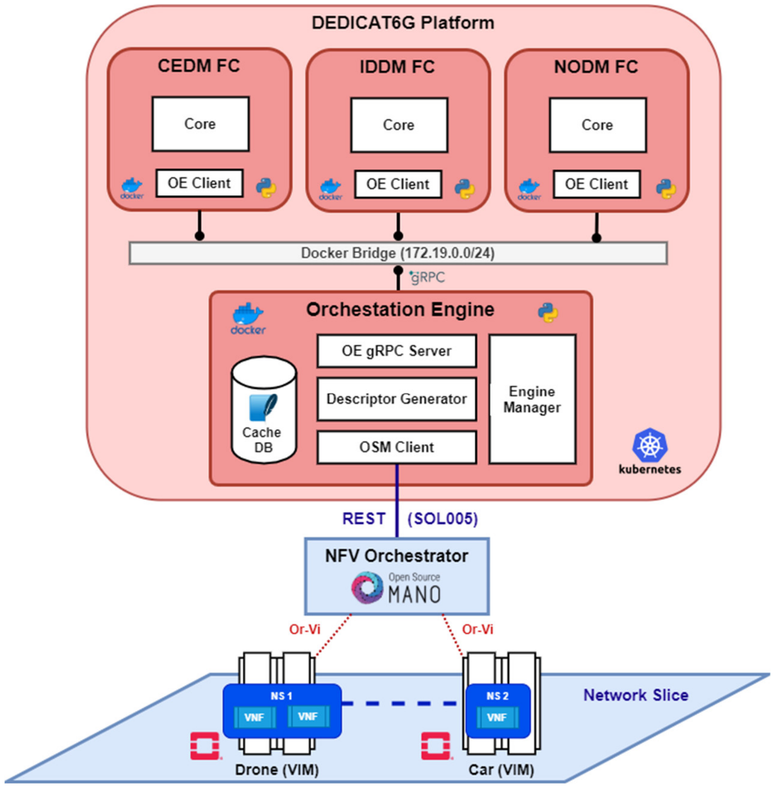


Figure 36. Orchestration Engine Implementation Schema

Regarding connectivity among existing entities, we consider two types of communication, internal to DEDICAT6G and external to the rest of the system. A Docker bridge (a virtual network) has been configured to allow communication within the DEDICAT6G platform. This bridge is used to enable the establishment of gRPC channels, which is the microservices-based communication chosen for data exchange between the FCs and the orchestration engine. In this work, the orchestration engine is considered as the anchor of the DEDICAT6G platform to link to external entities. In this case, the orchestration engine has an OSM client to allow leveraging the by-default OSM REST interface. It is noteworthy that this interface is defined in the ETSI GS NFV-SOL 005 specification [47]. Finally, OSM recognizes the two Open Stacks as VIMs, and uses the Or-Vi interface for the instantiation of the VNFs in an automatic manner [Or-Vi].

For the sake of simplicity and to cover the objectives in this work, we have assumed a Network Slice with two network service, one of them with two VNFs and the other with one.

3.3.2.4 Workflow

Here, in Figure 37, we present the workflow followed in this work. It is important to clarify that this experiment starts with the interaction between the IDDM FC and the orchestration engine. Thus, we assume that the IDDM have previously calculated the output. In this work, we focused on the orchestration engine side, not in the optimization of the intelligence distribution. Also, due to the relation to coverage extension mechanisms, the parallel interactions among the orchestration engine and the CEDM and NODM are shown in D4.2. The steps in this experiment are as follows:

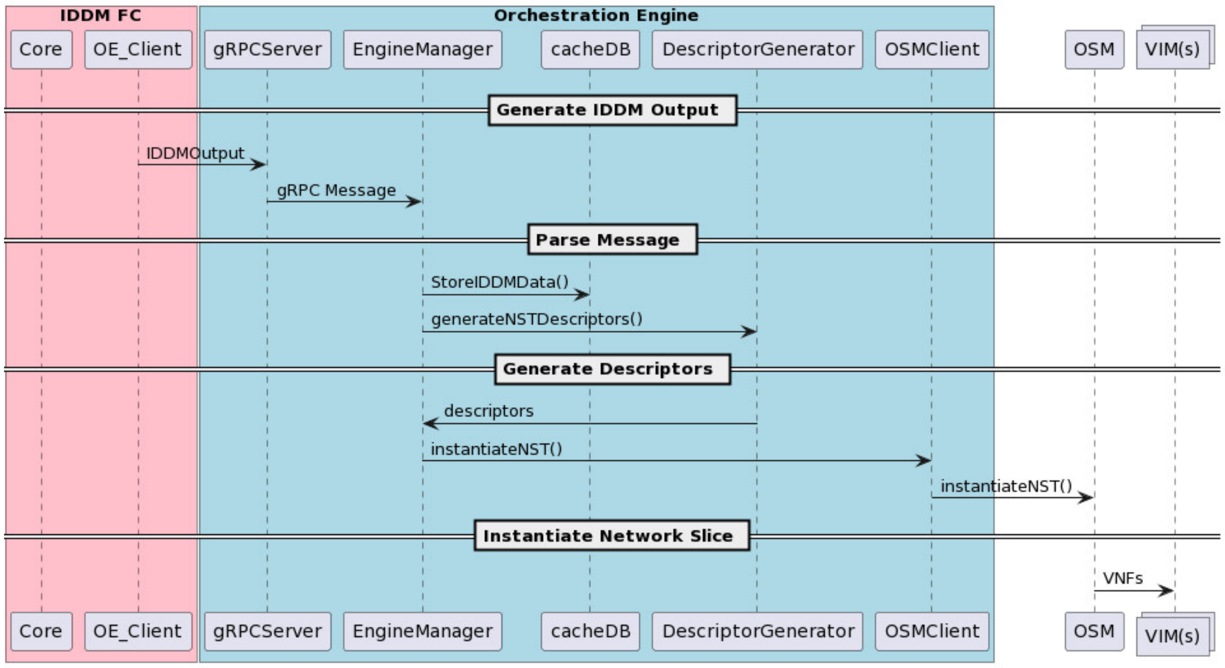
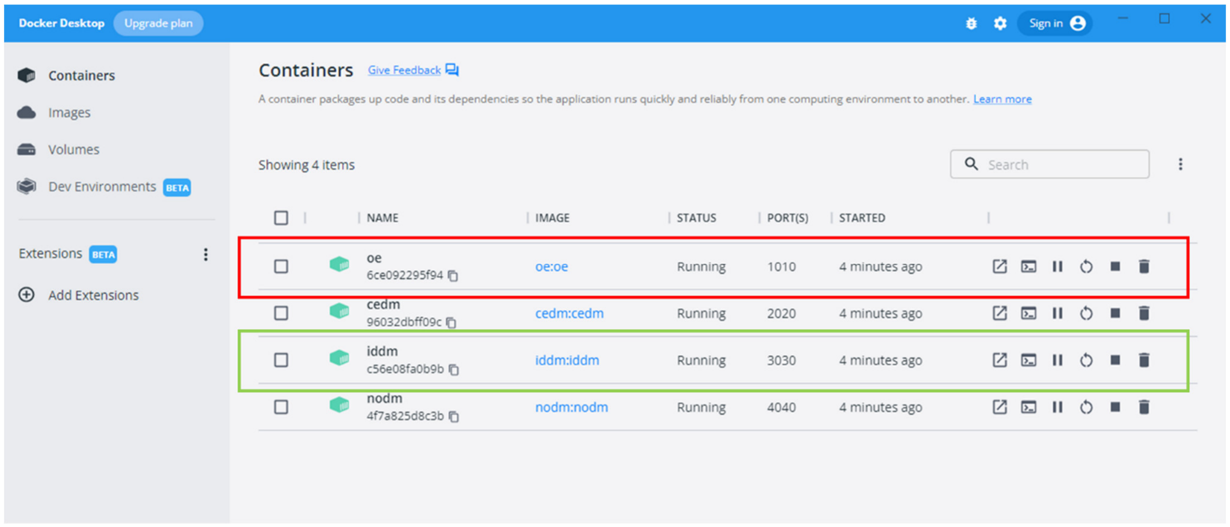


Figure 37. Sequence Diagram of the workflow

1. The Core of the IDDM FC generates the output by using the structure defined in the corresponding WP2 data model, and encapsulate this info in the shape required and sends it to its OE_Client
2. The OE_Client, opens a gRPC channel to send the generated message (IDDM output) to the Orchestration Engine gRPCServer by using a concrete rpc.
3. Now, the IDDM Output is in the orchestration engine domain, and the gRPC forwards the message to the EngineManager
4. Then, the EngineManager is in charge of parsing the data and structure it according to the internal structure of the cacheDB, and store it
5. Once the IDDM data is stored and processed, the EngineManager commands the DescriptorGenerator to automatically create the corresponding descriptors (VNFDs, NSDs and NSTD) with the IDDM information in the correct fields according to the ETSI SOL006 data model.
6. After this stage, the network slice is properly modelled in this set of descriptors are it is ready to be instantiated. Thus, the EngineManager requests the OSMClient to instantiate the NST (Network Slice Template in ETSI SOL006 terminology).
7. The OSMClient sends a request by using the OSM REST interface to instantiate a network slice based on the descriptor attached in that request.
8. Finally, OSM process this request and orchestrate the instantiation of the network slice in the VIMs.
9. After a few seconds, the network slice is properly configured and available to be used, with the VNF associated to the NSs consuming computational resources in each VIM.

3.3.3 Preliminary Results

As can be seen in Figure 38, four different docker containers have been built and instantiated, three that emulate the CEDM, IDDM and NODM FCs, and an additional one that contains the orchestration engine. Each of these containers, have preconfigured ports to enable gRPC communication by using the Docker bridge, thus, allowing the transmission and reception of messages compliant with the data models defined in D2.4.



Containers

A container packages up code and its dependencies so the application runs quickly and reliably from one computing environment to another. Learn more

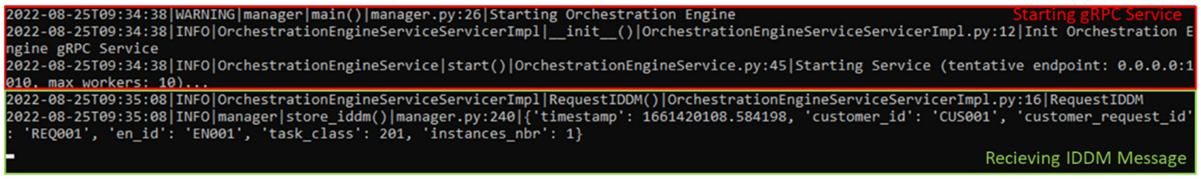
Showing 4 items

NAME	IMAGE	STATUS	PORT(S)	STARTED
oe 6ce092295f94	oe:oe	Running	1010	4 minutes ago
cedm 96032dbff09c	cedm:cedm	Running	2020	4 minutes ago
iddm c56e08fa0b9b	iddm:iddm	Running	3030	4 minutes ago
nodm 4f7a825d8c3b	nodm:nodm	Running	4040	4 minutes ago

Figure 38. Snapshot of the Docker environment with all the containers up and running

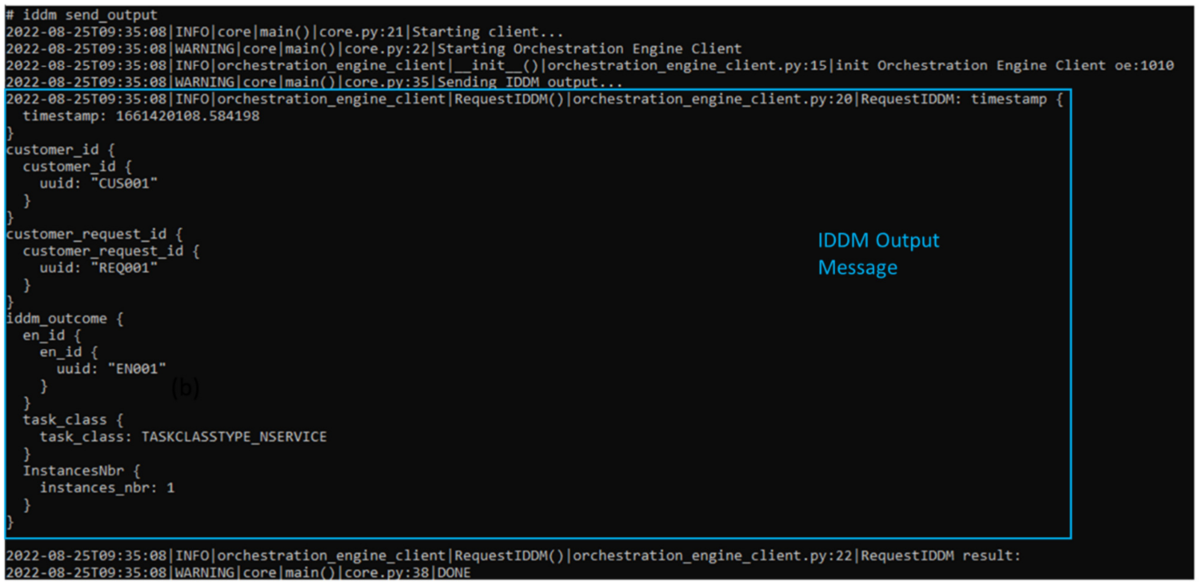
In this scenario, our main target is to instantiate a network slice, called *NST001,* which includes two NSs, the first one called *drone_ns*, emulating a NS to be instantiated in the drone IT resources, and the latter called *car_ns*, with a similar meaning. The *drone_ns* is composed by one VNF which will be instantiated in the VIM associated to the drone, while the *car_ns* has two VNFs which will be instantiated in the VIM corresponding to the car. The first step once all containers are running, is to start the gRPC server of the orchestration engine, as can be seen in Figure 39a. Once the gRPC server is ready, the FCs can send all output messages, by using a specific gRPC Orchestrator Engine client, that will be received and stored by the OE. In this case, we will focus on the IDDM side, as CEDM and NODM will be further detailed in D4.2.

The information of such gRPC message, built by the core of the IDDM FC representing its recommendation after executing concrete algorithms to distribute the intelligence, is received by the OE gRPC server and stored in the cacheDB in the orchestration engine as can be seen in Figure 40. For instance, in this figure it can be appreciated how the IDDM indicates the OE about how to distribute the intelligence associated to the request "REQ001" between the edge nodes "EN001" and "EN002", which means that the intelligence associated to the network services (VNFs) must be deployed in the edge nodes attached to the Drone VIM and the Car VIM correspondingly. The Task Class Type 201 indicates that the nature of the request is for Network Services.



```
2022-08-25T09:34:38|WARNING|manager|main()|manager.py:26|Starting Orchestration Engine
2022-08-25T09:34:38|INFO|OrchestrationEngineServiceServicerImpl|__init__()|OrchestrationEngineServiceServicerImpl.py:12|Init Orchestration E
ngine gRPC Service
2022-08-25T09:34:38|INFO|OrchestrationEngineService|start()|OrchestrationEngineService.py:45|Starting Service (tentative endpoint: 0.0.0.0:1
010, max workers: 10)...
2022-08-25T09:35:08|INFO|OrchestrationEngineServiceServicerImpl|RequestIDDM()|OrchestrationEngineServiceServicerImpl.py:16|RequestIDDM
2022-08-25T09:35:08|INFO|manager|store_iddm()|manager.py:240|{'timestamp': 1661420108.584198, 'customer_id': 'CUS001', 'customer_request_id'
: 'REQ001', 'en_id': 'EN001', 'task_class': 201, 'instances_nbr': 1}
```

(a)

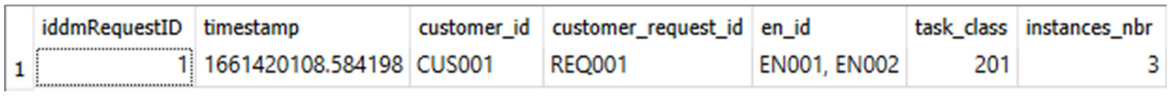


```
# iddm send_output
2022-08-25T09:35:08|INFO|core|main()|core.py:21|Starting client...
2022-08-25T09:35:08|WARNING|core|main()|core.py:22|Starting Orchestration Engine Client
2022-08-25T09:35:08|INFO|orchestration_engine_client|__init__()|orchestration_engine_client.py:15|init Orchestration Engine Client oe:1010
2022-08-25T09:35:08|WARNING|core|main()|core.py:35|Sending IDDM output...
2022-08-25T09:35:08|INFO|orchestration_engine_client|RequestIDDM()|orchestration_engine_client.py:20|RequestIDDM: timestamp {
  timestamp: 1661420108.584198
}
customer_id {
  customer_id {
    uuid: "CUS001"
  }
}
customer_request_id {
  customer_request_id {
    uuid: "REQ001"
  }
}
iddm_outcome {
  en_id {
    en_id {
      uuid: "EN001"
    }
  }
  task_class {
    task_class: TASKCLASSTYPE_NSERVICE
  }
  InstancesNbr {
    instances_nbr: 1
  }
}

2022-08-25T09:35:08|INFO|orchestration_engine_client|RequestIDDM()|orchestration_engine_client.py:22|RequestIDDM result:
2022-08-25T09:35:08|WARNING|core|main()|core.py:38|DONE
```

(b)

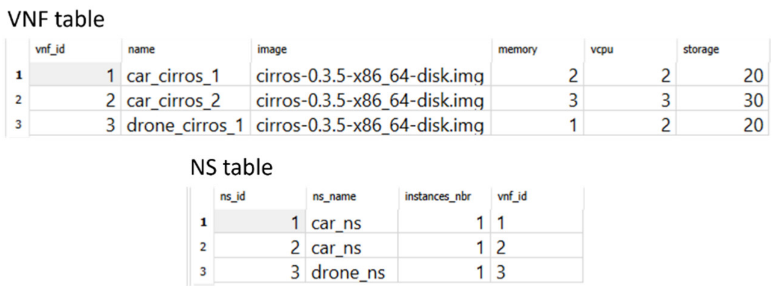
Figure 39. Terminal snapshot with logs messages of the containers corresponding to: a) Orchestrator Engine; b) IDDM.



	iddmRequestID	timestamp	customer_id	customer_request_id	en_id	task_class	instances_nbr
1	1	1661420108.584198	CUS001	REQ001	EN001, EN002	201	3

Figure 40. Snapshot of the SQLite GUI showing the IDDM information stored

When the OE receives enough information to process the new request, including some information sent by CEDM and NODM, the OE starts processing the information to instantiate a network slice containing the necessary NSs for the drone and the car. These NSs, are associated with one or several VNF images, also including resource information like RAM, VCPU and Storage to be consumed by the VNFs. As can be seen in Figure 41, this mapping between the NSs and VNFs is stored in the cache DB (and managed by the Engine Manger), this information can be obtained via the µS/FC registry and repository FCs. In our experiment, this functionality is modelled by the internal registries of OpenStack, as we will see later in this section.



VNF table

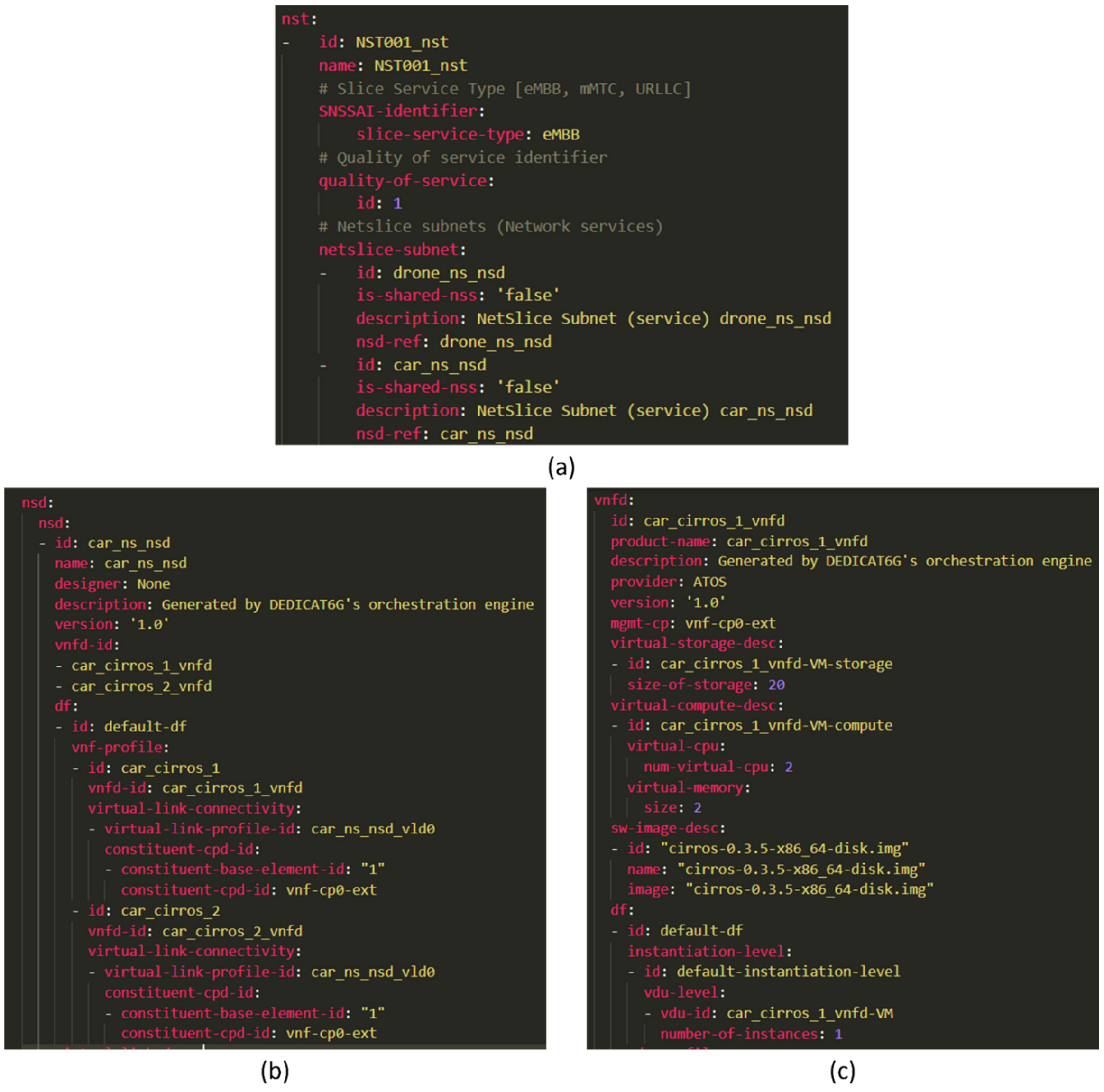
	vnf_id	name	image	memory	vcpu	storage
1	1	car_cirros_1	cirros-0.3.5-x86_64-disk.img	2	2	20
2	2	car_cirros_2	cirros-0.3.5-x86_64-disk.img	3	3	30
3	3	drone_cirros_1	cirros-0.3.5-x86_64-disk.img	1	2	20

NS table

	ns_id	ns_name	instances_nbr	vnf_id
1	1	car_ns	1	1
2	2	car_ns	1	2
3	3	drone_ns	1	3

Figure 41. VNF internal mapping to associated to the NSs in OE cache DB.

Once the information regarding VNFs for every NS is extracted, the next step is to prepare the necessary descriptors to translate the information received by the FCs, to a Data model understood by OSM (ETSI SOL006). Three types of descriptors are generated, including VNF descriptors (VNFD), network service descriptors (NSD) and network slice templates (NST).



```
nst:
-   id: NST001_nst
    name: NST001_nst
    # Slice Service Type [eMBB, mMTC, URLLC]
    SNSSAI-identifier:
        slice-service-type: eMBB
    # Quality of service identifier
    quality-of-service:
        id: 1
    # Netslice subnets (Network services)
    netslice-subnet:
    -   id: drone_ns_nsd
        is-shared-nss: 'false'
        description: NetSlice Subnet (service) drone_ns_nsd
        nsd-ref: drone_ns_nsd
    -   id: car_ns_nsd
        is-shared-nss: 'false'
        description: NetSlice Subnet (service) car_ns_nsd
        nsd-ref: car_ns_nsd
```

(a)

```
nsd:
  nsd:
  - id: car_ns_nsd
    name: car_ns_nsd
    designer: None
    description: Generated by DEDICAT6G's orchestration engine
    version: '1.0'
    vnfd-id:
    - car_cirros_1_vnfd
    - car_cirros_2_vnfd
    df:
    - id: default-df
      vnf-profile:
      - id: car_cirros_1
        vnfd-id: car_cirros_1_vnfd
        virtual-link-connectivity:
        - virtual-link-profile-id: car_ns_nsd_vld0
          constituent-cpd-id:
          - constituent-base-element-id: "1"
            constituent-cpd-id: vnf-cp0-ext
      - id: car_cirros_2
        vnfd-id: car_cirros_2_vnfd
        virtual-link-connectivity:
        - virtual-link-profile-id: car_ns_nsd_vld0
          constituent-cpd-id:
          - constituent-base-element-id: "1"
            constituent-cpd-id: vnf-cp0-ext
```

(b)

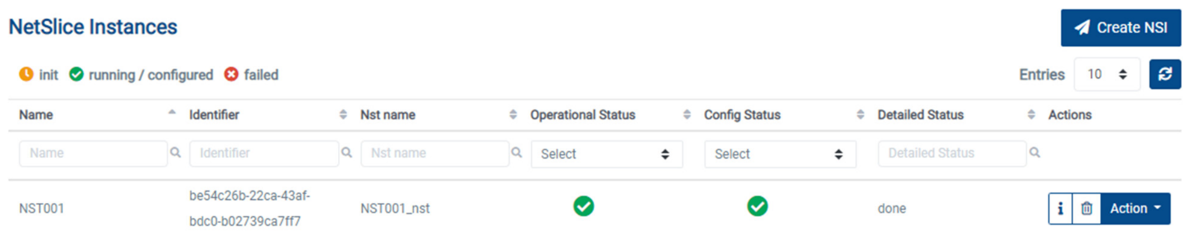
```
vnfd:
  id: car_cirros_1_vnfd
  product-name: car_cirros_1_vnfd
  description: Generated by DEDICAT6G's orchestration engine
  provider: ATOS
  version: '1.0'
  mgmt-cp: vnf-cp0-ext
  virtual-storage-desc:
  - id: car_cirros_1_vnfd-VM-storage
    size-of-storage: 20
  virtual-compute-desc:
  - id: car_cirros_1_vnfd-VM-compute
    virtual-cpu:
      num-virtual-cpu: 2
    virtual-memory:
      size: 2
  sw-image-desc:
  - id: "cirros-0.3.5-x86_64-disk.img"
    name: "cirros-0.3.5-x86_64-disk.img"
    image: "cirros-0.3.5-x86_64-disk.img"
  df:
  - id: default-df
    instantiation-level:
    - id: default-instantiation-level
      vdu-level:
      - vdu-id: car_cirros_1_vnfd-VM
        number-of-instances: 1
```

(c)

Figure 42. Examples of some of the generated descriptors of: a) network slice (NST), b) network service (NSD) and c) VNF (VNFD).

An example of each type of descriptors can be seen in Figure 42, showing the hierarchy of each one, from the NST relating the NS, to the car VNF specifying the needed image and computational resources. Focusing on the VNF descriptor, we can see that the resources requested in the VNF table in Figure 41 the first row are mapped to the corresponding fields in the descriptor. There we can appreciate corresponding metrics to model the RAM, VCPU, storage and also the VNF image name is also available, which in this particular case is a very lightweight version of Linux called cirrOS with some basic functionalities. Furthermore, a configuration file is created to map the multi-site information received by the FCs, making it possible to instantiate different NS that belongs to the same network slice in different VIMs.

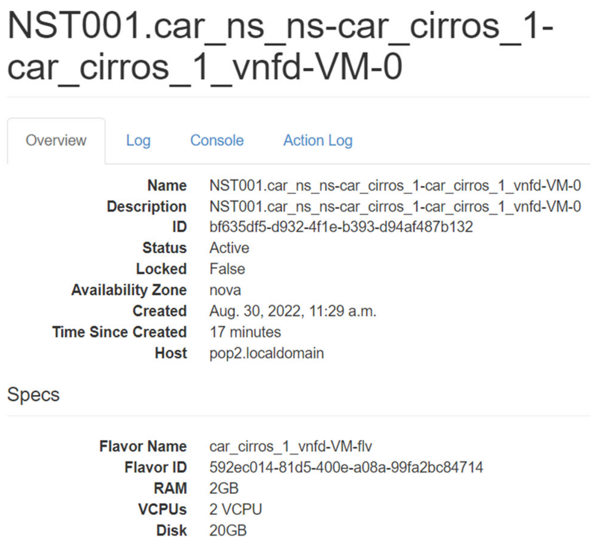
Once the descriptors are generated, they are onboarded into OSM by the OE, after which it commands OSM to instantiate the required slice. These actions are executed via the OSM Client implemented in the OE. Figure 43 shows the NST001 network slice successfully running in OSM. More information about the network slice instance is presented in D4.2.



Name	Identifier	Nst name	Operational Status	Config Status	Detailed Status	Actions
NST001	be54c26b-22ca-43af-bdc0-b02739ca7ff7	NST001_nst	✓	✓	done	Action

Figure 43. Snapshot of OSM with the network slice instantiated commanded by the Orchestrator Engine

As for the computational resources consumed associated with this network slice instance (NST001), the key elements are the VNFs, in turn 3 different VNFs, two instantiated in the car VIM and one in the drone VIM. In Figure 44, the different requirements that were previously established in Figure 42c, have been successfully captured in the VNF instance.



NST001.car_ns_ns-car_cirros_1-car_cirros_1_vnfd-VM-0

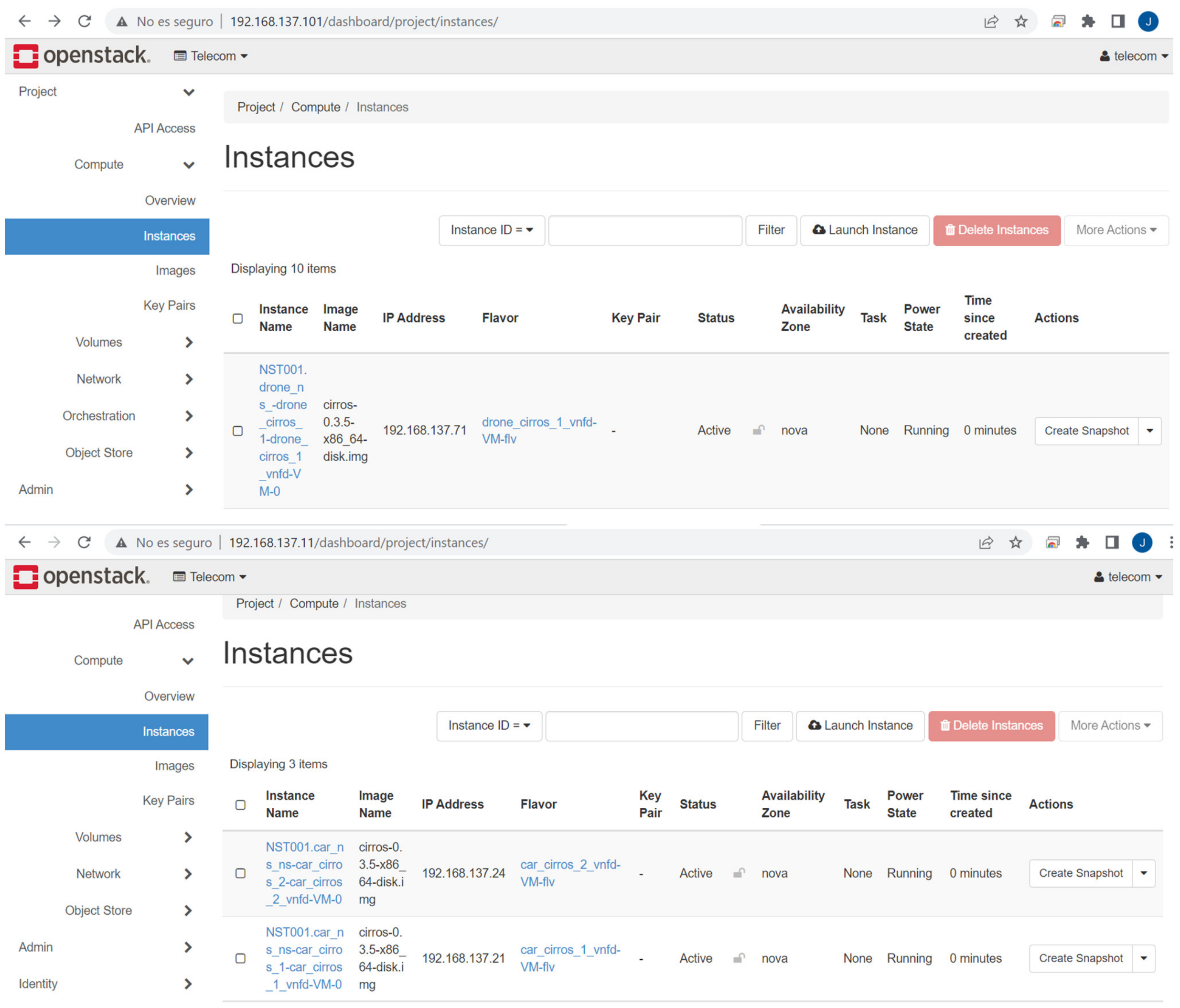
Name	NST001.car_ns_ns-car_cirros_1-car_cirros_1_vnfd-VM-0
Description	NST001.car_ns_ns-car_cirros_1-car_cirros_1_vnfd-VM-0
ID	bf635df5-d932-4f1e-b393-d94af487b132
Status	Active
Locked	False
Availability Zone	nova
Created	Aug. 30, 2022, 11:29 a.m.
Time Since Created	17 minutes
Host	pop2.localdomain

Specs

Flavor Name	car_cirros_1_vnfd-VM-flv
Flavor ID	592ec014-81d5-400e-a08a-99fa2bc84714
RAM	2GB
VCPUs	2 VCPU
Disk	20GB

Figure 44. VNF instance in car VIM (OpenStack) indicating the requirements (RAM, VCPU and Disk).

As can be seen in Figure 45, one VNF instance of cirrOS is running in the (emulated) drone VIM with IP 192.168.137.101, which represents the drone VIM. For the car NS, two VNFs with cirrOS are deployed in the car node, with IP 192.168.137.11, accomplishing the target of a multi-site network deployment commanded by the DEDICAT6G platform.



Instances

Displaying 10 items

	Instance Name	Image Name	IP Address	Flavor	Key Pair	Status		Availability Zone	Task	Power State	Time since created	Actions
☐	NST001.drone_ns_-drone_cirros_1-drone_cirros_1_vnfd-VM-0	cirros-0.3.5-x86_64-disk.img	192.168.137.71	drone_cirros_1_vnfd-VM-flv	-	Active		nova	None	Running	0 minutes	Create Snapshot

Instances

Displaying 3 items

	Instance Name	Image Name	IP Address	Flavor	Key Pair	Status		Availability Zone	Task	Power State	Time since created	Actions
☐	NST001.car_ns_ns-car_cirros_2-car_cirros_2_vnfd-VM-0	cirros-0.3.5-x86_64-disk.img	192.168.137.24	car_cirros_2_vnfd-VM-flv	-	Active		nova	None	Running	0 minutes	Create Snapshot
☐	NST001.car_ns_ns-car_cirros_1-car_cirros_1_vnfd-VM-0	cirros-0.3.5-x86_64-disk.img	192.168.137.21	car_cirros_1_vnfd-VM-flv	-	Active		nova	None	Running	0 minutes	Create Snapshot

Figure 45. Snapshot of the nodes (OpenStack) with the VNFs' instances up and running

3.3.4 Conclusions

As we have shown in this work, the provision of network slices has an impact on the management of the computational resources available in a 5G/B5G system, especially critical in the edge domain, and in turn, on the distribution of intelligence. Considering this fact, the DEDICAT6G project is working to cover it and provide optimization when needed, precisely the orchestration engine is the one in charge of such role.

One of the main insights of this orchestration engine is that its cloud-based design and microservices-based implementation permits its deployment in multiple B5G/6G scenarios thanks to its potential high availability, scalability and distributed manner. In the future, we will be able to test these capabilities by extending our work to take on new proof-of-concepts where the orchestration engine can be scaled in/out according to scenario requirements, deployed on multiple nodes, and creating multiple replicas to improve its availability and resilience.

Additionally, it is expected to extend the functionalities offered by the OE to assist the deployment of vertical apps and other FCs instance on edge nodes using the Kubernetes interface and command it, as Edge Orchestrator.

4 Security and Trust

In order to monitor security threats and establish secure and privacy-preserving AI/ML training and inference, a framework for secure data exchange must be set up. The DEDICAT 6G security and privacy protection framework is based on a decentralized, blockchain powered data marketplace for secure, automated monetization, processing and exchange of IoT sensors and digital assets data with technical and policy-based data verification.

The framework's unique features for monetization and exchange of data between arbitrary interested parties are:

- Private, permissioned Blockchain technology which provides network security, data integrity, smart contract for fast automated transactions and micropayments with a token economy.
- Data verification, technical and policy-based through blockchain Smart Contracts and data hashing (anchoring).

Data and algorithm providers, as well as data buyers and consumers, can use web or mobile app clients to communicate with the framework using specified API, while framework components communicate internally using internal APIs and/or message brokers. Framework components include two domains:

- Privacy preserving domain that relies on blockchain
- AI domain that relies on tools for AI workflow management and workflow management in general.

All the internal components from both domains – services and its dependencies – are running in the private network and public entry points to them are strictly controlled by ingress controllers and reverse-proxies

4.1 Access, Authentication, and Authorization Management

Access control, authentication, and authorization is achieved through Security Framework and Trust Management Platform. Architecture of the solution is explained in D2.4 and, for the sake of better understanding the complete flow, this document will share some parts with D2.4. Also, the entire WP5 (all the tasks in it) explains the Platform and provides detailed overview of its architecture and implementation, so some content will be shared with D5.1 and D5.2. We will focus on architecture and deployment and address scalability and integration with the orchestration engine. Authentication, authorization, and access control will be addressed in that respective order.

The access control flow is designed in such a way as to provide a scalable solution and quick and easy integration of new services. There are multiple ways to include access control calls to the new service. The least intrusive approach that does not require code changes is at the deployment level. A sidecar container can be deployed alongside the new service that will intercept all the requests and authorize them. The other option is direct integration with the Framework. While authentication, authorization, and access control represent different FCs, implementation-wise only one remote procedure call (RPC) request is necessary for the integration. Detailed explanation of the flow and architecture can be found in D2.4. The figure below shows the integration of the new service into the Platform.

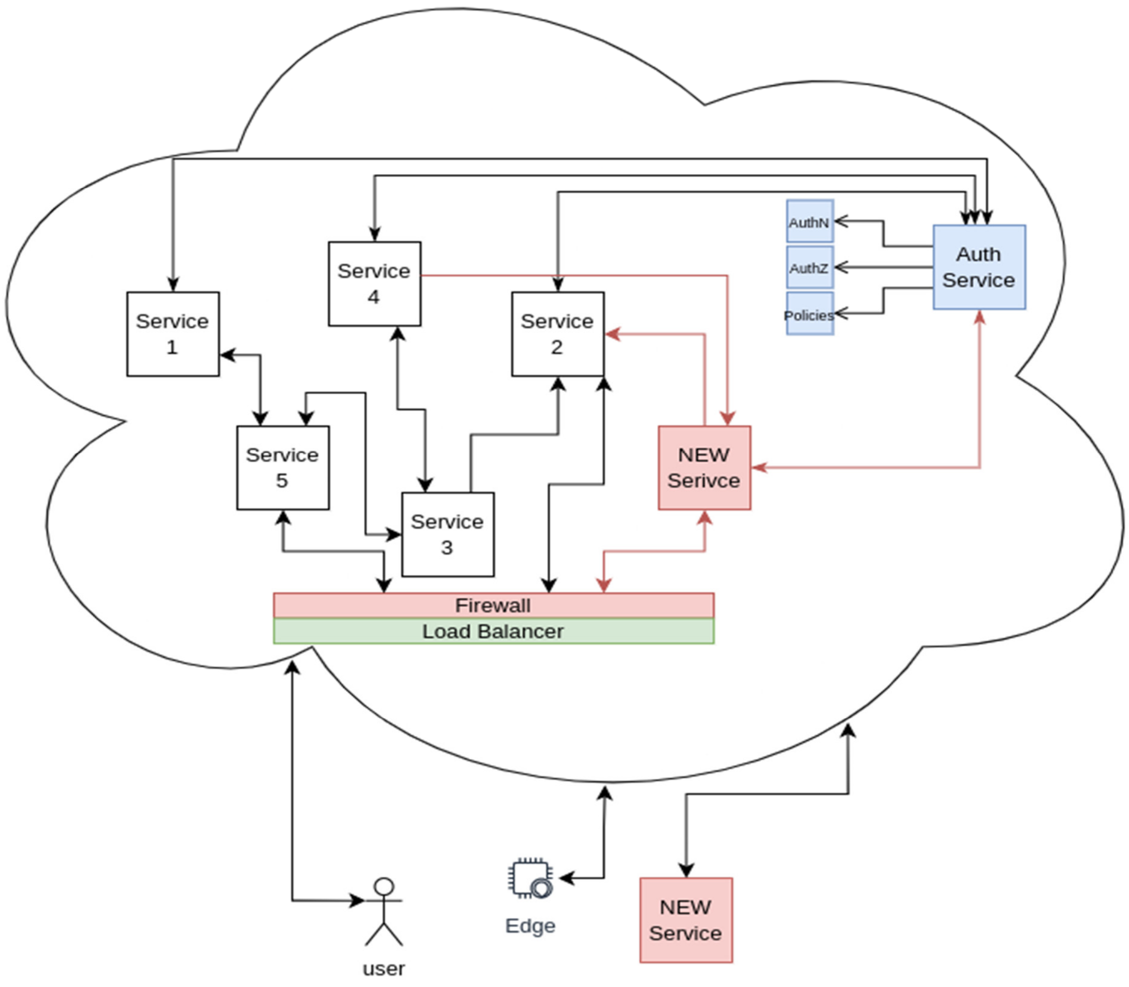


Figure 46. Integration of the new service with the existing access control flow

4.1.1 Authentication

Authentication is the process of determining a participant's identity in the system. The Framework relies on token-based authentication - which means that the client's identity data is encoded in the self-contained token that is exchanged with each request. The system's authentication is by its nature stateless - anyone with the correct token can be authenticated successfully and all the data necessary for the process is encoded in the token. Implementation-wise, the default token is JSON Web Token (JWT) with a short expiration time. Once JWT is expired, the refresh token needs to be used to issue a new JWT. Refresh tokens, unlike access tokens, are stored in the can be revoked. Refresh tokens can be used only once. For HTTP requests, the token is embedded in the HTTP request headers.

Authentication solution is, however, developed with constrained devices in mind. In some cases, IoT devices are rather low power and low energy which means that exchanging a large JWT with each request is simply too heavy for them. Also, in some cases, those devices do not even use the request-response model of communication, but rather publish-subscribe or even fire and forget models. The Framework is structured in such a way that it supports those constrained devices by design. This means that the token type can easily be replaced with a proprietary, simpler, and less secure token if needed. Currently, there is ongoing work in the WP5 on the support for the MQTT protocol which will be using a publish-subscribe model of communication.

4.1.2 Authorization

Authorization connects access control policies to the identity that is resolved in the authentication process and evaluates those rules against it. There are a couple of simple general rules in the approach to authentication, authorization, and access control:

- Each interaction with external entities needs to be authorized.
- When assigning access control rules, use the Principle of least privilege (PoLP).
- All the networking, internal and external, needs to be secured via TLS.
- Access control evaluation must be quick.

Since authentication, authorization, and access control are three stages of the single flow and are by nature inseparable, some of these steps are applicable across all of them.

The first rule imposes that all the interactions, even those that essentially do not require authorization (such as health check endpoints) need to go through the authorization flow. The reason for this is the simplification of the process and providing necessary logging and future-proof support. The second rule is enforced by design, but it is really the administrator's responsibility to respect it. It means that all the clients have only the minimum privileges necessary to perform the given action. By default, almost all the actions are forbidden, except for logging in and fetching its own data. The third rule imposes using the active best practices for network protection of the data in transit - TLS by default and mutual TLS as an even more secure solution. Mutual TLS is not set by default because the certificate and key distribution for all the clients is simply not viable for large-scale deployment with general-purpose clients. The fourth rule is loosely defined on purpose. The exact acceptance criteria have not been yet developed to strictly define the limits and would be a useful contribution to WP7. The problem with strict criteria is the versatility of use cases and client types that all have different tolerance to latency. Experience from Nokia Data Marketplace (NDM) that is used as a reference when developing the Framework witnesses it should be in the 200ms range. However, since the Framework includes more machine-to-machine (M2M) communication, some parts of it can tolerate even greater latency. Others, such as real-time sensor data monitoring - cannot.

4.1.3 Access control

Access control is based on combined Role-based access control (RBAC) on the higher level and Attribute-based access control (ABAC) for fine-grained permission tuning. The policy design and implementation are rather simple, as well as policy evaluation against the given client for the provided action. Each client has a role assigned to it. The roles define rights for the client to set up policies for other clients over the resources in the system. Resources are called digital assets. Assets can be anything from dataset and algorithm to pictures, video, or NFT – the only requirement is that it has the URI.

The policy consists of rules in the form of Subject-Action-Condition. The subject is the client attached to the policy, action is one from the list of predefined actions and condition is the condition under which the policy applies. The simplest condition is represented by simple key-value pairs where the key is the name of the subject attribute, and the value is the expected value for that attribute - thus attribute-based access control. The conditions are generic by design and can incorporate more complex scenarios such as temporal or spatial constraints.

4.1.4 Audit logging and analysis

Logging enables collection and secure storage of all types of logs across the DEDICAT 6G system (with main focus on security) and implements blockchain technology to ensure logs consistency and trustworthiness.

Its main functions are:

- Collection of logs (transactions, audit of sessions, reports)
- Storage of trusted logs in a secure manner
- Providing access to logs for authorized clients

Trustworthiness metrics are calculated for edge nodes, processes, users and data streams. Trust metric value indicates if a node can join a local network, if process output can be further used, if a user can execute specific rule. Trust metrics are implemented as ML models whose outputs are written on private permissioned blockchain through dedicated smart contracts. This way all stakeholders in DEDICAT 6G instance have access to immutable record of trust metrics calculated for all actors, resources and processes.

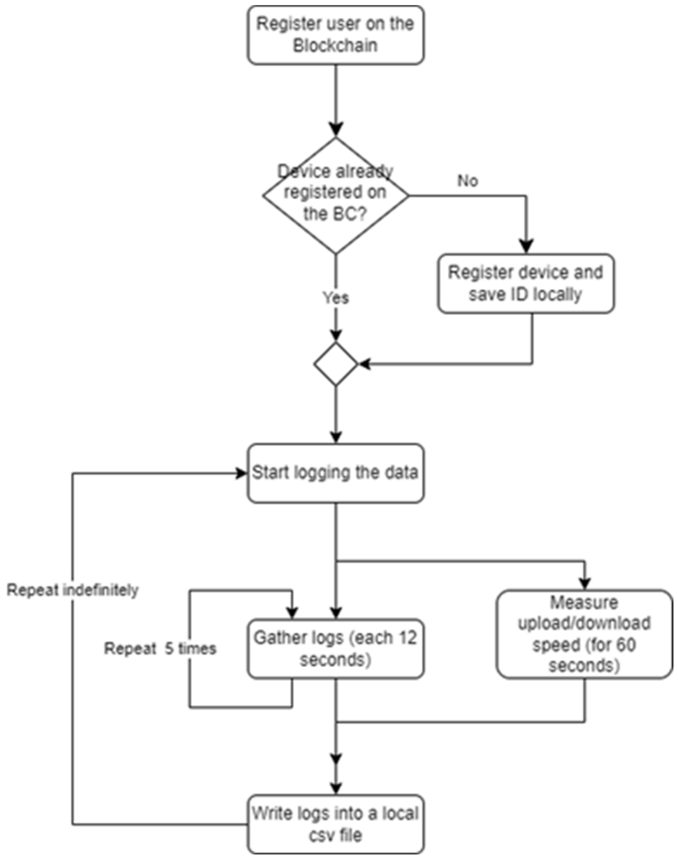

Register user on the Blockchain
Device already registered on the BC?
No
Register device and save ID locally
Yes
Start logging the data
Repeat indefinitely
Repeat 5 times
Gather logs (each 12 seconds)
Measure upload/download speed (for 60 seconds)
Write logs into a local csv file

Figure 47. Logging data process on edge device

The algorithms used for logged data analysis in our test lab environment are Decision tree and Random forest. Each of these algorithms is tried for two types of problems: for regression and for classification. In the case of classification, there are two classes: “Trust” and “Don’t trust”. Trust values above 0.5 are considered to belong to the “Trust” class and are assigned a value of 1, and values below or equal to 0.5 are considered as members of the "Don't trust" class and are assigned a value of 0.

4.2 Network and data security, Cryptography, and key management

While the privacy and protection framework that is under construction in WP5 is mostly focused on innovative approaches of a combination of trust management using blockchain technologies and extensible fine-grained access control, it is also using the best practices when it comes to securing data in transit. As has already been mentioned, all communication between all the services is secured using TLS. Also, the Platform is extensible for supporting mutual TLS. Problems with certificate distribution, key rotation, and device secure bootstrapping are not addressed yet. The other preferred practices for securing sensitive data are implemented some of them being hash and salt for users' passwords, account lock on subsequent unsuccessful login attempts, short-lived access tokens signed with cryptographically strong algorithm and key, and load balancer set up to prevent distributed denial-of-service (DDoS) attacks… Fine-tuning of the load balancer and REST API is done by security experts to improve safety and lower the risks of exploiting any sensitive information. Also, logging and collecting sensitive user data is prevented by using only internal unique user identifiers in logs, rather than sensitive information such as the user's email address.

All these measures, alongside the aforementioned approach to access control and integration with blockchain for trust support provide a very robust, yet scalable and reliable security framework that is easy to integrate, monitor, and use.

5 Conclusions

The DEDICAT 6G project has defined a set of global objectives, among which:

- To provide imperceptible end-to-end latency and response time, with a minimal energy and resource consumption in B5G networks for the support of innovative applications
- Reinforce security, privacy and trust in B5G systems in support of advanced IoT applications
- Develop human-centric applications and showcase novel interaction between humans and digital systems

These three objectives are at the core of the work performed in WP3. These objectives can only be achieved by combining both computation and communications capabilities of the whole network: core, edge, access, and even terminals. However, in 5G, it is lacking a unified platform that can leverage infrastructure programmability and AI techniques to meet the most stringent requirements of services.

This document has presented the work achieved so far in WP3. First, a description of several algorithms for Distribution of Intelligence was given, along with experimental results that were collected. Next, a detailed description of low-level architectural techniques was given, as well as the design of the orchestration engine. Finally, an overview of the security issues and how they are tackled is given.

In the next work period, the work will be refined and implemented in Proof-of-Concept prototypes. In addition, the work on architectural techniques will be continued, expanded to machine, cloudlet and edge region levels, and implemented in a prototype. These prototypes will then later on be integrated in the pilots defined in WP6.

References

[1] DEDICAT 6G Deliverable D2.2. Initial System Architecture, Editor F. Carrez, 2021.
[2] Alharbe, N.; Aljohani, A.; Rakrouki, M.A. A Fuzzy Grouping Genetic Algorithm for Solving a Real-World Virtual Machine Placement Problem in a Healthcare-Cloud. *Algorithms* 2022, *15*, 128. https://doi.org/10.3390/a15040128
[3] Canali, Claudia & Lancellotti, Riccardo. (2019). GASP: Genetic Algorithms for Service Placement in Fog Computing Systems. Algorithms. 12. 201. 10.3390/a12100201.
[4] Lu, J.; Zhao, W.; Zhu, H.; Li, J.; Cheng, Z.; Xiao, G. Optimal machine placement based on improved genetic algorithm in cloud computing. J. Supercomput. 2022, 78, 3448–3476.
[5] Abdoun O, Jaafar A, Chakir T (2012) Analyzing the performance of mutation operators to solve the travelling salesman problem. Neural Evolut Comput Int J Emerg Sci 2(1):61–77
[6] DEDICAT 6G D3.1.First Release of Mechanisms for Dynamic Distribution of Intelligence, Deliverable D3.1, 2021
[7] Mitchell S, O'Sullivan M, Dunning (2011) Pulp: A linear programming toolkit for python. Accessed May 1, 2013, https://code.google.com/p/pulp-or/
[8] Kaidi Yang, S.Ilgin Guler, and Monica Menendes, Isolated intersection control for various levels of vehicle technology: Conventional, connected, and automated vehicles, Transportation Research Part C: Emerging Technologies, Vol. 72, November 2016, Pages109-129.
[9] Xiangdong Xu, Hong K. Lo, Anthony Chen, and Enrique Castillo, Robust network sensor location for complete link flow observability under uncertainty, Transportation Research Part B: Methodological, Vol. 88, June 2016, Pages1-20.
[10] Yunyi Liang, Zhizhou Wu, and Jia Hu, Road side unit location optimization for optimum link flow determination, Computer-Aided Civil and Infrastructure Engineering, Volume 35, Issue 1, January 2020, Pages 61-79
[11] http://www.pyomo.org
[12] https://coin-or.github.io/pulp/
[13] Z. Ma, "The Generalized Quadratic Assignment Problem," 2004.
[14] R. Kasimbeyli, "Comparison of Some Scalarization Methods in Multiobjective Optimization," in Bulletin of the Malaysian Mathematical Sciences Society, 2019.
[15] Z. Tang, "Migration Modeling and Learning Algorithms for Containers in Fog Computing," in IEEE Transactions on Services Computing, 2019.
[16] H. Mossalam, "Mult-Objective Deep Reinforcement Learning," 2016.
[17] A. Abels, "Dynamic Weights in Multi-Objective Reinforcement Learning," 2017.
[18] R. Eyckerman, "Evaluation of Objective Function Descriptions and Optimization Methodologies for Task Allocation in a Dynamic Fog Environment," in IOTSMS, 2020.
[19] D. Balemans, "Resource Efficient Sensor Fusion by Knowledge-Based Network Pruning," in Internet of Things (Netherlands, 2020.
[20] S. Rangan, "Millimeter-Wave Cellular Wireless Networks: Potentials and Challenges," IEEE, vol. 102, no. 3, pp. 366-385, 2014.
[21] NGMN, "5G White Paper".
[22] I. Qualcomm Technologies, "VR and AR pushing connectivity limits," 2017.
[23] Qualcomm, "Making Immersive Virtual Reality Possible in Mobile," 2016.
[24] METIS, "FP7-ICT-317669-METIS/D1.1".
[25] M. Marjalaakso, "Security Requirements and Constraints of VoIP," Helsinki University of Technology.
[26] Cisco, "Quality of Service Design Overview," in Cisco Press book End-to-End QoS Network Design: Quality of Service for Rich-Media & Cloud Networks, 2nd Edition , 2014.
[27] 5GPP, "White Paper Phase 1 Security Landscape," 2017.

[28] 5G-Ensure, "Deliverable D3.6 5G-PPP security enablers open specification," 2017.

[29] B. Sas, "Classifying Users based on their Mobility Behavior in LTE networks," in 10th Int. Conf. on Wireless and Mob. Comms. (ICWMC), 2014.

[30] J. Hsu, "Mining GPS data for mobility patterns: A survey," in Pervasive and Mobile Computing, 2013.

[31] Boyd et al, Convex Optimization, Cambridge University Press, 2004.

[32] Q. Ye et al., "User association for load balancing in heterogeneous cellular networks," IEEE Trans. Wireless Communication, 2013.

[33] S. Aaronson, The Limits of Quantum Computers, Scientific American 298, 3 (March 2008), 62-69.

[34] M. Forsell, Minimal pipeline architecture—an alternative to superscalar architecture, Microprocess Microsystems 20, 5 (1996), 277–284.

[35] M. Forsell, A Scalable High-Performance Computing Solution for Network on Chips, IEEE Micro 22, 5 (September- October 2002), 46-55.

[36] M. Forsell, Architectural differences of efficient sequential and parallel computers, Journal of Systems Architecture 47, 13 (July 2002), 1017-1041.

[37] M. Forsell and V. Leppänen, An Extended PRAM-NUMA Model of Computation for TCF Programming, International Journal of Networking and Computing 3, 1 (2013), 98-115.

[38] M. Forsell, V. Leppänen and M. Penttonen, Cost of Bandwidth-Optimized Sparse Mesh Layouts, In the Proceedings of 13th International Conference on Parallel Computing Technologies (PaCT'15), Lecture Notes in Computer Science (LNCS) 9251, Au-gust 31 - September 4, 2015, Petrozavodsk, Russia, 375-389.

[39] M. Forsell, J. Roivainen, V. Leppänen, Outline of a thick control flow architecture. In: Proc. MPP 2016, SBAC-PAD 2016, October 26–28, 2016. Marina del Rey Marriott, Los Angeles, USA.

[40] M. Forsell, J. Roivainen and V. Leppänen, REPLICA MBTAC - Multithreaded Dual Mode-Processor, Journal of Supercomputing 74, 5 (2018), 1911-1933.

[41] M. Forsell, REPLICA Multiprocessor Framework, White Paper, VTT, April 2020.

[42] International Technology Roadmap for Semiconductors, Semiconductor Industry Association, year 2015; http://www.itrs2.net.

[43] J. Keller, C. Keßler, and J. Träff, Practical PRAM Programming, Wiley, New York, 2001.

[44] V. Leppänen, M. Forsell and J-M. Mäkelä, Thick Control Flows: Introduction and Prospects, Proc. 2011 Int. Conf. on Parallel and Distributed Processing Techniques and Applications, July 18-21, 2011, Las Vegas, USA, 540-546.

[45] A. Mämmelä and A. Anttonen, Why Will Computing Power Need Particular Attention in Future Wireless Devices?, IEEE Circuits and Systems 17, 1 (2017).

[46] A. Ranade. How to Emulate Shared Memory. Journal of Computer and System Sciences 42, (1991) 307–326.

[47] ETSI GS NFV-SOL 006, "NFV descriptors based on YANG Specification". Available Online: https://www.etsi.org/deliver/etsi_gs/NFV-SOL/001_099/006/03.06.01_60/gs_nfv-sol006v030601p.pdf

[48] Docker - https://www.docker.com

[49] Kubernetes - https://kubernetes.io

[50] GRPC - https://grpc.io

[51] Protobuf - https://developers.google.com/protocol-buffers

[52] SQLIte - https://www.sqlite.org

[53] OSM - https://osm.etsi.org/

[54] OpenStack - https://www.openstack.org/