DEDICAT 6G

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

Deliverable D3.1
First 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_D3.1
Deliverable Title:	First Release of Mechanisms for Dynamic Distribution of Intelligence
Editor(s):	Y. Carlinet (Orange)
Author(s):	A. Anttonen (VTT), Y. Carlinet (Orange), P. Demestichas (WINGS), F. Díaz (ATOS), M. Forsell (VTT), V. Lamprousi (WINGS), J. Moreno (ATOS), K. Mößner (TUC), S. Penjivrag (VLF), N. Perrot (Orange), P. Reiter (IMEC), D. Ribar (Airbus), V. Stavroulaki (WINGS), L. Valeyre (Orange)
Reviewer(s):	H. Cesar Carvalho de Resende (IMEC), R. Eyckerman (IMEC), D. Hadiwardoyo (IMEC), J. Mäkelä (VTT), V. Stavroulaki (WINGS), M. Uitto (VTT)
Contractual Date of Delivery:	31/12/2021
Submission Date:	22/12/2021
Dissemination Level:	PU
Status:	Final
Version:	v1.0
File Name:	DEDICAT6G_D3.1 Mechanisms for dynamic distribution of Intelligence_v1.0.doc

Disclaimer
The information and views set out in this deliverable are those of the author(s) and do not necessarily reflect the official opinion of the European Union. Neither the European Union institutions and bodies nor any person acting on their behalf may be held responsible for the use which may be made of the information contained therein.

Deliverable History

Version	Date	Modification
V1.0	*22/12/2021*	*Final version, submitted to EC through SyGMa*

Table of Content

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
MA	Mobile Assets
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
MIMO	Multiple-Input Multiple-Output
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
NSSAI	Network Slice Selection Assistance Information
NSSF	Network Slice Selection Function
OBU	On-Board Unit
OS	Operating System
OSM	Open Source MANO
PCF	Policy Control Function
PDCP	Packet Data Convergence Protocol
PDU	Protocol Data Unit
PFCP	Packet Forwarding Control Packet
PHY	PHYsical layer
PLMN	Public Land Mobile Network
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
RDF	Resource Description Format
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
RU	Radio Unit
SIMD	Single Instruction Multiple Data
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
UDR	Unified Data Repository
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 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 first instalment of a series of three, describing the achievements in WP3. These achievements follow three dimensions:

1. **Architecture overview (section 2).**

In order to fulfill the objectives of the project, the DEDICAT 6G platform must enable imperceptible end-to-end latency. To this end, it is necessary to move service and network intelligence closer to the end-users. In consequence, intelligence must be distributed towards the edge network, and in order to enable minimal resource consumption, it will be necessary to migrate services sometimes.

Security requirements are listed, and in order to meet them, the project will design and implement a blockchain technology (to build trust between devices and nodes), data flows with privacy, and data analytics techniques (for automated security audits).

The Key Performance Indicators (KPIs) measured in the project include Quality of Service (QoS) parameters, such as latency, throughput, and also other parameters like energy or service reliability.

Following up on D2.2 [13] (Initial System Architecture), the key components related to WP3 are identified, and described in detail.

2. **Algorithms for Distribution of Intelligence (section 4)**

The project will design and implement algorithms for the placement of intelligence (i.e. functional entities) while optimizing a given KPI, usable for all four use-cases.

We will also consider the particular case of Smart Warehousing, in which a given KPI is optimized, while considering a delay proportional to the load on the links. And in the Smart Highway use-case, in which the placement algorithm can be centralized or decentralized, the problem of Road Side Units (RSU) placement is modelled as a coverage problem.

3. **Architectural techniques for Distribution of Intelligence (section 5)**

Finally, we also study 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 are conducted at the processor and server level.

1 Introduction

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, practical aspects of them are integrated in the use case pilots in WP6.

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 preliminary as the final outcomes of the work will be detailed in the next upcoming deliverables This document is the first iteration out of the three planned over a period of 18 months.

1.2 Document Structure

This document presents, in Section 2, the architecture of the DEDICAT 6G system, with a particular focus on the elements related to WP3. This section also gives a definition of the term 'intelligence', in order to define better the scope of the work in this work package. There is also a description of security issues and the Key Performance Indicators that will be measured in order to assess the feasibility of the taken approaches. Finally, Section 2 also provides an introduction to the networked computing approach that will be studied in the project.

Section 3 gives an overview of the State-of-Art for the intelligence distribution algorithms. It also lists the most mature technologies for intelligence distribution, which could potentially be used as a building block for the development of the pilots.

Section 4 details the work achieved so far on the design of the algorithms for intelligence distribution, in the general case, and also in the particular cases of some use-cases defined in DEDICAT 6G.

Section 5 provides the work performed on architectural techniques for distribution of intelligence. In particular, the following techniques are studied: efficient context switching, patterns of computation and communication, e.g., multicasting, load balancing, movement of threads, support for keeping the state of the computation simple as well as efficient computation management.

2 Architecture Overview

2.1 Introduction

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. Indeed, in order to enable imperceptible end-to-end latency it is necessary to move service and network intelligence closer to the end-users. In consequence, intelligence must be distributed towards the edge network, and in order to enable minimal resource consumption, it must be migrated when necessary.

2.2 Definition of Intelligence

In the context of the DEDICAT 6G project, the term 'Intelligence' refers to two main concepts.

Firstly, Intelligence represents micro-services, which can be chained in order to provide a service to end-users. For instance, in UC4, the road user awareness is done through Service Level Agreement (SLA) Management and sensor post-processing. The intelligence of the system is Local Dynamic Map (caching) -- shared world view. Central LDM is on cloud, distributed to MECs edge RSUs. Intelligence distribution is a combination of task offloading and caching.

Secondly, Intelligence can refer to Virtual Network Functions (VNFs). VNFs are usually run in Virtual Machines (VMs), and they can be chained to provide network services, such as virtual Radio Access Network (vRAN) or virtual Evolved Packet Core (vEPC).

Finally, it should be noted that Intelligence also encompasses the set of Functional Components (FCs) of the DEDICAT 6G platform architecture itself. They are actually micro-services, which can be deployed or migrated in the same way as other micro-services.

2.3 Security Issues

The project will implement mechanisms and tools for improving security, privacy, and trust in the scope of IoT applications supported over B5G/6G networks, as illustrated in Figure 1.

Such mechanisms and tools will include:

- functionality for controlling data flow in terms of privacy and confidentiality
- cutting edge blockchain technology for building trust between devices and nodes with automated compliance tests, automated auditing and immutable record for key trust parameters
- AI and data analytics for the timely detection and forecasting of security, privacy and trust threats and for ensuring resilience. This will be based on federated learning mechanism for the highest level of privacy and data protection.

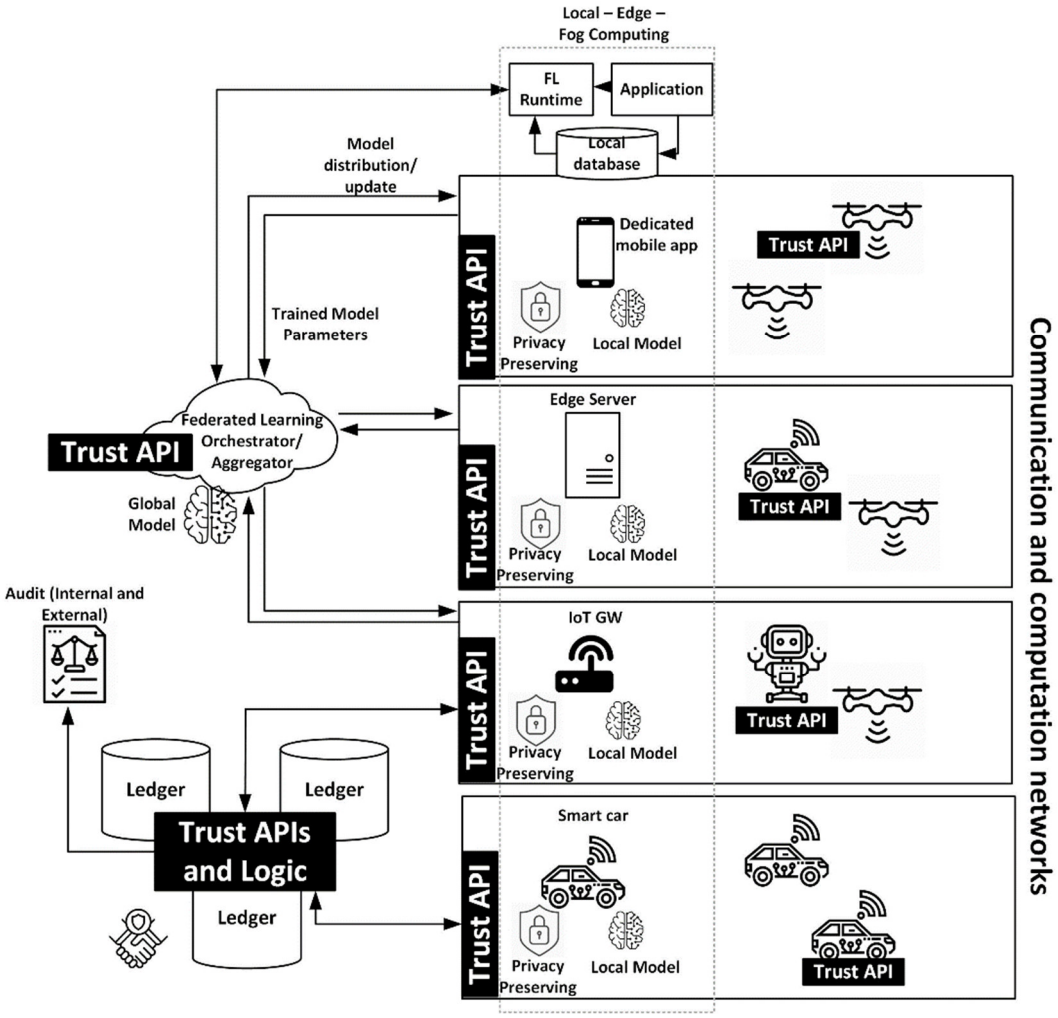


Figure 1: Security and trust logic

Security requirements will be associated with the security features. Requirements must be implemented early in the development phase and the appropriate security features are mounted on the devices themselves.

Table 1: List of general security requirements

Security Requirement	Description
Access, Authentication, and Authorization Management	Authenticate users through central authentication/authorization (AuthN/AuthZ) systems, grant the minimum, sufficient access, or privileges, employ role-based access controls, access sensitive data only as necessary for job duties, encrypt authentication and authorization mechanisms
Audit logging and analysis	Enable logging for endpoints, include essential events and elements in logs, restrict log access to authorized individuals, automate alerting on logging failures
Cryptography and key management	Protect digital assets and communications, implement GDPR, user/role access to the encryption keys
Network and data security	Implement default-deny, least-privilege policies on network devices, encrypt network traffic, securely configure network infrastructure devices

Code integrity	Validate the integrity of a component/driver or system file each time it is loaded into device memory, encrypt external transmission of data, Implement application logs with important event data
Data validation and sanitization	Validate on Input - ensuring that incoming data is uncompromised before it is allowed to be processed, sanitize device/storage media before transfer, ensure sanitization methods meet the standard's requirements

2.4 Key Performance Indicators

This sub-section outlines some Key Performance Indicators that are linked to the intelligence distribution mechanisms. Corresponding measurements will be collected during the pilots via tools such as: (i) ping, a software tool which measures the network layer Round Trip Time (RTT) for messages sent from one machine to another machine; (ii) iPerf an open source, multi-platform and freely available tool which can be used to measure network bandwidth, jitter and packet loss; (iii) custom measurements through software, etc.

Latency

The aim is to achieve decreased latency (incl. mean delay and delay jitter) via intelligence distribution mechanisms by up to a factor of 10 in congested and fault situations in order to improve quality of experience of multimedia applications. For 4G system, the target user plane latency (RTT) and control plane latencies (idle-to-active time) are, respectively, 20ms and 100ms, while 5G systems should lead to 1ms and 10ms for the corresponding latencies [14]. Often, these requirements are still not met in practical networks with high traffic density.

Validation methodology: Measurements will be collected at the application layer by adding timestamps to requests between functional entities/service components of an overall service. Then the difference in time will be calculated between the request from one entity (e.g., client) and the response from the other entity (e.g., server). Jitter measurements will also be collected e.g., with the use of iPerf.

Energy

The aim is to decrease energy consumption (incl. communication and computation) via intelligence distribution mechanisms by at least a factor of 10 in order to increase the operation lifetime of a mobile station or server. A typical measure of energy efficiency is the number of transmitted bits per Joule (bit/J), and it is subject to various systems assumptions. A theoretical physical upper limit is around 1 Pbit/J while typical 4G systems lead to a level of 10 Kbit/J and 5G systems to a level of 10 Mbit/J [15]. Other measures can also be found, depending on the system assumptions.

Validation methodology: The battery level of involved mobile nodes (e.g., phones, laptops, robots) will be measured with and without the use of intelligence distribution mechanisms for certain services/applications. The power consumption of involved servers may also be measured or at least estimated using various ways ranging from cheap watt hour meters for on

premises servers to estimation of FLOPs (Floating Point Operations)[1],[2].

Mission Critical Services QoS

The ETSI has released the Technical Specification 122 179 in 2020 [16] which defines indicators about quality and availability of the network used by 3GPP clients. The MCX mobile application (3GPP client) will use the network connectivity extended by the DEDICAT 6G platform in the context of loss of network of the use case UC3. The quality and availability of the Intelligent Distributed Coverage Extension brought by DEDICAT 6G shall not be regressed accordingly to the standard. The MC Audio Functional Component shall address the 4 main indicators defined in the standard:

- Provide MC-PTT (Mission Critical Push-To-Talk) access time less than 300 ms for 95% of all requests;
- The end-to-end MC-PTT access time less than 1000 ms for all MCX mobile application under the same network coverage;
- The mouth-to-ear latency less than 300 ms for 95% of all voice bursts;
- The maximum late call entry time shall be 150 ms for 95% of all late call request.

The indicators related to IP and data transmissions (MC Data or MC Video Functional Components shall be as following):

- The end-to-end Delay, which is the time, required for IP packets to be transmitted shall be less than 10 ms;
- The User Data Rate shall be 100 Mbps in downlink and 50 Mbps in uplink;
- The Reliability indicator of the DEDICAT 6G shall be at least 99.999% of success for the transmission of a packet of 32 bytes within 1ms.

Validation methodology: The indicators related to voice connectivity and transmission (MC-PTT) will be measured by using the MCX Mobile application using DEDICAT 6G platform. The indicators related to IP data will be measured by the use of a test application.

The results will be compared with the measures done with the use of MCS application in a nominal environment (without DEDICAT 6G platform) based on a 5G networks.

Throughput

Some services (e.g. augmented reality) require a high throughput in addition to low latency.

Validation methodology: Throughput can easily be obtained from the video renderer software.

Service reliability (application layer)

Service reliability can be defined as the success probability of transmitting a layer 2/3 packet within a maximum latency required by the targeted service (ITU-R M.2410, cf. [17]). With the use of intelligence distribution mechanisms this should either improve or at least remain the same as the baseline.

Validation methodology: The packet error rate at the application layer (packets that arrive delayed or erroneous are considered as lost packets) will be measured.

[1]https://devblogs.microsoft.com/sustainable-software/how-to-measure-the-power-consumption-of-your-backend-service/

[2] Schwartz, Roy, Jesse Dodge, Noah Smith and Oren Etzioni. "Green AI." *Communications of the ACM* 63 (2020): 54 - 63.

2.5 DEDICAT 6G Networked Computing Approach

The core in the distribution of intelligence (DoI) in WP3 is the ability to comprehend where the DoI-enabling computation should take place given the network state defined via a target system parameter set. Since there are a number of things to be considered, different optimization objectives can be formulated. In this section, the purpose is to collect the main affecting features and elaborate their different interrelationships as a whole. More detailed descriptions of specific design objectives are then provided in Section 5. In the following, we divide the required activities roughly into five different categories. These are further illustrated in Figure 2.

Incentives for DoI: Different DEDICAT 6G use cases, which provide a specific need for the DoI, have been analysed in D2.2 [13]. While a detailed stimulus may change depending on the application, the reasons for using the DoI across the use cases are typically three-fold, namely user device overloading, service remoteness, or server congestion. The avoidance of these events via sophisticated DoI evidently lead to decreased service delay, usability or system energy efficiency. In general, the triggering for the need of the DoI can be initiated either by a user or network operator.

Architectural designs: The architectural design aspects can be addressed at different abstraction levels. At the top level, the network topology architecture to ensure end-to-end connectivity is a dominating activity. At the middle level groups of edge servers form regions within which regional computational load balancing strategies can be applied. At the bottom level, server processing architecture (computer, processor, interconnect, memory, storage and input/output dimensions) is defined. The bottom level solutions should also make possible efficient application of upper levels. The fourth important element is the applied centralization degree of the decision-making control structure. The centralization degree may vary form fully centralized control to fully decentralized control with a number of hybrid solutions, e.g., regional control, in between. The actual implementation of the DoI architecture can be done, e.g., via specific container structures or virtual (distributed) machines.

System state monitoring: An important part of the DoI concept is to enable to monitor the system state that is specific with the target system parameter set. While the higher cardinality and estimation frequency of the parameter set lead to better accuracy, the complexity of subsequent decision-making tasks may increase exponentially. Therefore, some trade-offs must be made between the state monitoring accuracy and complexity. This is especially true for the large system scenarios that need to support high scalability regarding the users and interrelated servers. The system state may refer to network and server loads which change due to user mobility as well as service state which refers to the running state of an application that is important for service continuity if the service location is changed at runtime. The system state may also refer to the past, present, or future state of the system. These functionalities are found in the Context Awareness and Analytics Functional Groups, as described in the next Section (2.6).

Resource allocation: While the output of the resource allocation typically includes the user scheduling decisions, it involves a number of things affecting the allocation. The resource allocation should provide sufficient QoS levels for possibly heterogeneous users that may execute different applications within a target region of interest. The resources should be allocated so that the interference between different transmitting nodes is sufficiently mitigated. Finally, the resources should be allocated so that the load across the network is not too imbalanced while respecting given user fairness and prioritization targets. At lower levels, sources for interference include dynamic voltage frequency scaling (DVFS), dependencies between execution units, operating system scheduler, memory access/intercommunication

patterns, lack of bandwidth, out-of-order/speculative execution and processor pipeline hazards. These functionalities are found in the Service Orchestration Functional Group, as described in the next Section (2.6).

Placement decisions: In the concept of DoI there are a number of placement decisions to be made. One of the first tasks is to decide the physical server locations with network connectivity capability. While some servers are pre-deployed and static, certain servers may be mobile, introducing more flexibility for online. The next placement decision involves the runtime connected server selection for each user task entering the region of interest. Obviously, this decision is highly dependent on the requested service requirements, prevailing system state, and resource allocation situation to avoid overloading the servers and attached network access points. Instead of completing a certain task within a single server, in some cases cooperative computing involving multiple heterogeneous servers (e.g., work sharing, work stealing schemes) may be beneficial for more efficient computing task scheduling. Finally, adjusting service locations in servers (aka service migration) may help to address the consequences from dynamic network topology changes. Several things such as user-server distances and service location change costs must be taking into account. The underlying decisions call for an optimization procedure which may be based, e.g., on a rule-based approaches, model-based combinatorial optimization tools, or model-free machine learning methods. These functionalities are found in the Decision-Making Functional Group, as described in the next Section (2.6).

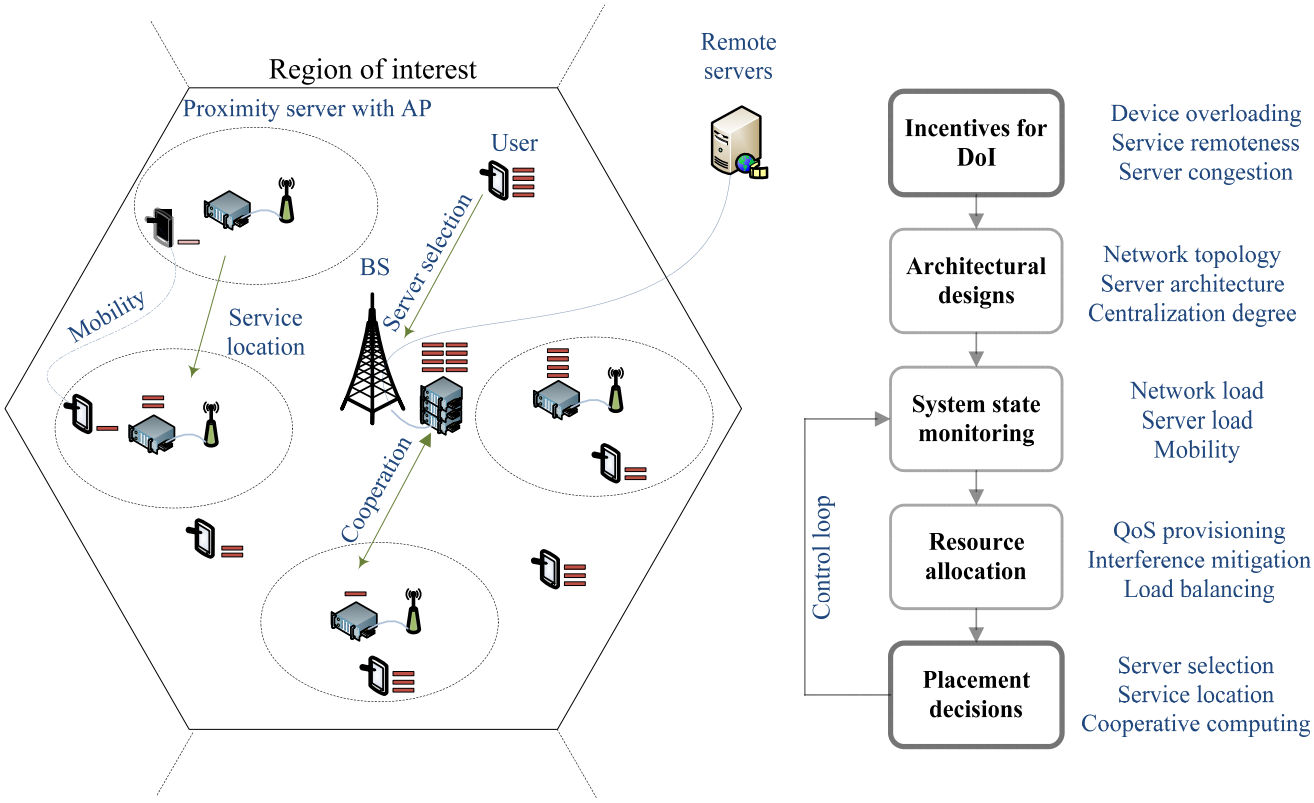


Figure 2: Networked computing framework in WP3

2.6 Functional components specific to WP3

The DEDICAT 6G functional model is composed by a group of core and transversal functional groups (FGs) that encompasses a set of functional components belonging to a common functional framework. Thus, according to D2.2 [13] the core DEDICAT 6G FGs are:

- **Context-Awareness FG:** provides a set of FCs that can provide the necessary information to build concrete contexts that can be potentially used by the Decision-Making FCs to take decisions according to such input in different levels.
- **Decision Making FG:** is responsible for making decisions on how to act on the corresponding context stimulus by hosting the algorithmic to solve the key problems addressed in this project, such as network operation, coverage extension and intelligence distribution. In this work, the focus is on the latter.
- **Service Operation FG:** oversees the implementation of the outcomes provided by the Decision-Making FG related to services in general. These functionalities include orchestration of network services and microservices, load balancing and resource management.
- **Intelligence Distribution FG:** is targeted to support the distribution of intelligence in the system by hosting the necessary registries and repositories as well as other FCs aimed at managing the MEC and SLA/policy procedures. WP3 is responsible for the definition and development of the FCs in this FG.
- **Coverage Extension FG:** comprises FCs that underpin dynamic coverage extension such as the operation of swarm-based device mobility or connected cars operations. This FG is entirely the responsibility of WP4.
- **Analytics FG:** monitors and analyses the DEDICAT 6G platform in general by using the inputs provided by the Decision-Making and Context-Awareness FGs in order to increase the overall performance of the system and some particular key FCs in other FGs. Namely, network optimization/prediction or ML-based techniques to enhance the performance of the platform.

Additionally, the three transversal FGs in DEDICAT 6G are able to interact with the rest of FGs to support their operations:

- **Management FG:** hosts the corresponding functionalities to transversally manage the DEDICAT 6G platform, like GUI interface, dashboard or deployment tracking.
- **Communication FG:** enables the FCs intercommunication in both the cloud and edge side and to establish communication with external entities to interact of command them when proceed.
- **Privacy/Security/Trust FG:** is focused to guarantee the security, privacy and trust requirements in the platform.

This section aims to provide a general vision of only the functional components (FCs) defined, in the first instance, in D2.2 [13], which are directly or indirectly related to dynamic intelligence distribution operations, putting the light on the most relevant ones for WP3. Such FCs will be upgraded and evolved during the execution of the project fed by a multi-directional interaction between WP2, WP3 and WP6.

Figure 3 represents a functional vision centered on Intelligence Distribution where vertical applications are on top of the DEDICAT 6G platform to assist and command their connectivity and edge computing resources through the use of the legacy 5G network.

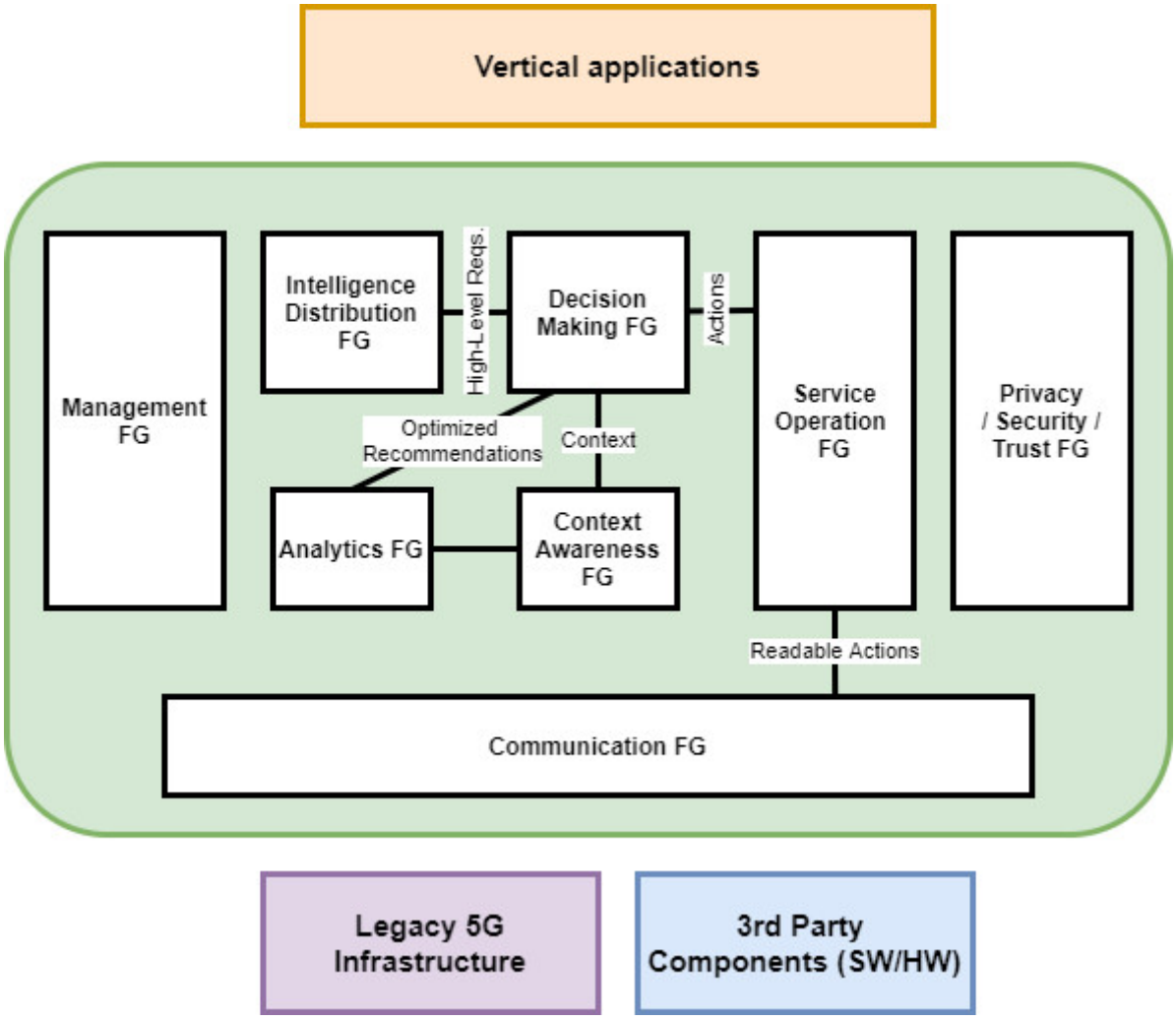


Figure 3: Functional Architecture of the DEDICAT 6G ecosystem applied to Intelligence Distribution

2.6.1 Intelligence Distribution FG Components

These components offer the placeholder to compile the all the information, artifacts and devices related to the services that are deployed by the verticals in the DEDICAT 6G platform. It acts as the backend of the Dashboard FC, from where all nodes, software components, service SLA and policies are managed. The Intelligent distribution functional group serves as reference point for the context aware and decision-making components in the sense of providing

1. A set of instructions negotiated with the verticals for an optimal deployment.
2. The registry of devices and nodes onboarded in the DEDICAT 6G platform that belongs to each vertical.
3. Registry for the software artifacts that can be potentially deployed in the platform.

Components are:

- **EC Policy Factory FC:**

It hosts and provides the set of pre-defined policies (per vertical) that support the proper deployment and configuration of the verticals' applications when using the Edge Computing capabilities of the DEDICAT 6G platform. The EC policies should provide the logic and timing

of the deployment of intelligence and dedicated resources to meet the QoS/QoE requirements set out in the FC Factory SLA.

- **SLA Factory FC:**

This FC contains a specific framework for the DEDICAT 6G platform and the vertical to negotiate the details related to QoS and QoE in the form of a contract. The outcome of such a contract will be a balance between the requirements of the vertical application and the technical constraints associated with the particular scenario in which the application will be deployed.

- **Edge Node Registry FC:**

It contains all the necessary information of the network and computational resources available in each edge node of the EC system to support the dynamic distribution of intelligence. Some of the metrics to be stored in the registry can be CPU processing power, number of cores available and consumed per node, RAM and storage availability, existing network interfaces, battery lifetime (for mobile edge nodes) and others.

- **Edge node discovery and lookup FC:**

This FC is in charge of registering new edge nodes to the system and notifying any potential issues related to the edge node availability.

- **µService Registry FC:**

In this FC, the instantiation information required by the microservices, such as number of CPU cores, RAM, storage size or bandwidth, are stored in a registry. This fine-grained information complements the EC policy factory FC data but is focused on the microservice level.

- **µService Repository FC:**

It registers and stores the available images and related metadata of the microservices.

- **µService Discovery and Lookup FC:**

The functionality provided by this FC is analogue to the edge node discovery and lookup FC but targeted to microservice instead of edge nodes.

2.6.2 Context-Awareness FG Components

The Context-Awareness FG encompasses all the functional components that retrieve information that contributes to build one or more than contexts, essential to assess the Decision-Making FG in its role. Additionally, the contexts built can be potentially used by the Analytics FG as inputs to enhance the performance or achieve optimal performance in the system.

Given its "information-donor" nature the Context-Awareness FG will be mainly related to the work performed in WP3 and WP4, and potentially, WP5. Given the scope of this document, here, we only provide descriptions of those FCs direct or indirectly related WP3.

- **Edge Node (EN) awareness FC:**

This FC comprises the necessary data from the Edge Nodes that can help to build a context of the Edge Nodes. Within the Decision-Making procedures, such context will be used as input by, at least, the Intelligence Distribution DM FC.

- **Edge Node (EN) status agents FC:**

This is an agent to be deployed in each Edge Node and it informs the Edge Node awareness FC about the current status of the node. It is under discussion if this FC can be an agent of the Edge Node awareness FC.

- **µService Awareness FC:**

This FC plays a similar role to the Edge Node awareness FC but applies to microservices. It provides information to some of the FCs of the Service FG, such as µService Orchestration FC or load balancing FC.

- **µService status agent FC:**

This agent will inform about the specific metric related to monitor the status of a µService (lifecycle, resource consumption in terms of number of CPU cores, storage or RAM).

- **Deployment Awareness FC:**

Assists the IDDM FC by providing a context of the microservices and FCs deployed in the system and their current impact on the edge node where they are hosted.

- **Deployment status agent FC:**

It is responsible for serving status on which FCs have been deployed in a given domain, and which Edge Nodes host them.

2.6.3 Analytics FG Components

Network Optimization FC, Network Prediction FC, Analytics Toolbox FC and Platform Performance Analysis FC are the FCs that make up the Analysis FG. As mentioned above, the Analysis FG is dedicated to improving system performance beyond the default mechanism on two main planes, the network and the platform, or a combination of them. By executing ML-based techniques, this FG will be able to complement the Decision-Making FG to perform specific tasks that may have an impact on the overall performance, such as network loading, KPI optimization or intelligence delivery optimization energy-wise.

The FCs directly related to networking are responsibility to WP4, in general, however, the platform performance analytics FC and analytics toolbox FC can be applied in the context of dynamic intelligence distribution.

- **Platform Performance analytics FC:**

This FC is constantly receiving data to monitor the performance in the DEDICAT 6G putting the light on specific KPIs, like system utilization rate, CPU utilization, cost or power consumption and to identify potential gaps to improve. Once, the gap is to identify this component is responsible to support other FCs to perform the same action under a different and more appropriate approach depending on the nature of the scenario and the KPI. For instance, it may command the Decision-Making FG to execute another algorithm more suitable than the original one, request the Context-Awareness FG to monitor new metrics or it can also apply ML-techniques available in the Analytics Toolbox FC.

- **Analytics toolbox FC:**

This component offers a set of general-purpose Machine Learning techniques to be used or consumed by other FCs in order to provide a better solution than the previous algorithm.

2.6.4 Decision-Making FG Components

The Decision-Making (DM) FG is the one responsible to execute algorithms that can solve certain problems related to coverage extension and intelligence distribution, in general, and facilitate the decision making in the DEDICAT 6G platform. This FG is key in the platform and can be considered as its "brain". The DM FG is fed of information and data by the Context-Awareness FG and it will command the Service Operation FG to apply the changes based on the DM FG recommendations.

As before, here we put the light on the part related to WP3.

- **Intelligence Distribution DM (IDDM) FC:**

The intelligence Distribution DM is one of the most relevant components in WP3. Its main role to play is to provide decision on how and where to consume computational resources and where to deploy the microservices. Additionally, the IDDM FC is not only responsible for making decisions on intelligence distribution or migration, but also to accommodate and execute the algorithms that can assist the decision making. The decision-making algorithms and mechanisms for distribution of intelligence presented in this deliverable will be part of this component. This set of algorithms are divided in three main groups:

  - *Service Placement algorithms*: allocate resources to services in the edge network, edge cloud or cloud domains.
  - *Intelligence Migration algorithms*: support for edge network node dynamicity and end-users or device mobility.
  - *Resource Allocation algorithms*: equity in the allocation of resources to services, in relation to their requirements.

In addition, this component will internally evaluate and manage the possible options for solving a problem and decide which algorithm to run in case several options can be applied.

2.6.5 Service Operation FG Components

This functional group oversees translating the commands given by the Decision Making FG to the external entities that can apply its recommendations in the system, such a NFV Orchestrator, load balancer or network/device controllers.

- **Orchestration FC:**

The Orchestration FC provides deployment and configuration details of the service or networking components in the selected node based on the recommendations of the DM FG and translates them to the corresponding entities that are able to apply them in the 5G/B5G system.

- **Load balancing FC:**

This FC is dedicated to managing the computational load balancing between the different FCs and microservices according to the results provided by the decision-making FG, precisely NODM and IDDM. This functionality will be further investigated and will oversee existing load balancing tools.

3 State-of-the-art

3.1 Intelligence Distribution Algorithms

The optimal computation, resource, and storage positioning for succeeding the best performance of challenging systems, populated by multiple mobile devices (IoT) with respect to mobile users and data protection, is of great interest in B5G/6G networks in academia as well as in industry sectors. Intelligence and resource allocation intended for B5G/6G networks, is anticipated to utilize novel optimization techniques and Artificial Intelligence-based approaches to effectively support this kind of systems.

Currently, cloud computing, edge computing and edge caching are three emerging and popular networking concepts to distribute computing. Cloud computing supports digital services that need to have high performance with unlimited operation lifetimes. Edge computing tends to encounter some cloud computing issues, like the high communication delay and network congestion due to long distance and bandwidth-intensive cloud applications. Edge caching aims at bringing frequently requested digital service content to the storages closer to the users. In parallel to these networking concepts, there are several scheduling methods that balance the workload of processor nodes for better performance.

Despite the development of the above concepts, there is still a need for further research and improvement to fulfil the emerging service and resource requirements of B5G/6G networks. User devices within these networks have increasing demand of power, are delay-sensitive and have immersive applications. Additionally, there are polarized communication needs in highly crowded areas and low-density but hard-to-reach areas. Also, the service needs are more heterogeneous and time varying, and resources become more limited and efficiency in all dimensions is vital.

In view of the above, the optimal allocation of computation and resources is, among others, related to the utilization of AI/ML functions and algorithms as micro-services that can be deployed as re-usable components for executing specific tasks across various applications. It is important though to study mechanisms that can dynamically determine which intelligence functions should be executed and where, based on the available resources across federated platforms and user mobility.

There are many studies in literature related to optimal resource allocation and functional placement in 5G, B5G and 6G [1], [2], [3]. The models usually found solving this problem are hybrid models, swarm intelligence, genetic modelling, rule-based systems, and case-based reasoning. Artificial intelligence techniques for resource allocation are used mainly for resolving issues such as a hybrid problem-solving approach. Some other popular techniques are game-theoretic approaches [4] and mixed integer programming [5]. In most cases resource allocation is an NP-hard problem, and algorithms of less possible complexity are preferable. The following two sub-sections provide an overview of some state-of-the-art on resource orchestration algorithms separated in two main categories, centralized and distributed algorithms.

3.1.1 Centralized Algorithms

Two centralized resource allocation algorithms were studied [6] and [7] papers. The first one addresses the energy-efficient resource allocation problem for device-to-multi-device communications and the second one addresses the energy-efficient resource allocation and power control in D2D-cluster scenarios. In paper [8] it is presented a centralized resource allocation method for addressing the problem of intercell interference for multiuser multiple-input multiple-output (MIMO) capable wireless LANs. Finally, a centralized dynamic resources

allocation method for LTE networks is proposed in [9] based on cloud radio access network (C-RAN) architecture.

3.1.2 Distributed Algorithms

A distributed resource scheduling as well as mode selection optimization algorithm based on coalition game is proposed in [10]. A novel distributed resource allocation algorithm based on Alternating Direction Method of Multipliers with Partial Variable Splitting for the utilization of both the spectrum resource offered by communication network and computational resources of a coexisting fog computing network is described in [11]. Another distributed resource allocation algorithm is proposed in [12], and it is based on game theory. It solves the problem of cross-tier interference between D2D communication and cellular users, as well as the co-tier interference between D2D communications.

3.1.3 Progress beyond the state-of-the-art

As it was mentioned above, there is an urgent need to further improve intelligence distribution mechanisms, to fulfil the increasing service requirements of B5G/6G networks. In this project and in WP3, we study and develop new mechanisms to dynamically allocate data, computation, and storage, based on network status information of both communication and computing loads. The dynamicity can come from different sources, namely share of elastic and inelastic on-demand services, user mobility, and mobility of data sources, network resource consumption, and computing resource consumption to meet the requirements of real-time and ultra-low-delay sensitive innovative IoT applications. Moreover, since many of the required decision-making tasks are coupled, the distribution mechanisms can be effective by jointly utilizing the underlying reactive and proactive approaches for multicasting, computing offloading, and edge caching methods. Within this project, it is planned to study, deploy and test initially centralized intelligence placement algorithms, as described in Section 4.

3.2 Technologies and frameworks

This section aims at giving an overview of the most advanced and mature technologies for distribution of intelligence in the edge network.

Dynamic deployment and migration of intelligence can only be achieved efficiently through virtualization, which in turn can be achieved with a container-based or a Virtual Machine-based technology. Both have pros and cons.

Virtual Machines (VM) provide better security boundaries with OS-level isolation. They also do not rely on a particular OS. However, VMs have higher resource management overhead and are not as easy to migrate as containers.

Containers are easier to manage in terms of portability and scalability. They require also less resources to execute.

The technologies described in this Section are potential candidates to be used in the DEDICAT 6G project, as building blocks to be built upon.

3.2.1 Kubernetes

Kubernetes[3] is a container-based platform. It allows service load-balancing, automatic scaling, and bin-packing, and it manages containers by restarting them when they fail or do not respond. Kubernetes is very mature because it was developed by Google and then open-sourced in 2014. Since then, the developers' community has been very active.

The Kubernetes orchestrator is in charge of selecting the physical nodes on which the tasks are to be deployed, among others. It does so with a care for balancing the load and allowing scalability of the services. However, it has two major limitations:

- It is not possible to explicitly provide isolation between 2 services (i.e. the tasks of one service does not share a physical node with another service). Isolation might be necessary, for instance, when a service has strict security requirements.
- There is no inter-cluster management. A service cannot span multiple tasks over different clusters and have them run with each other.

In the next Subsections, we describe potential technical solutions to overcome these limitations.

3.2.2 Crossplane.io

Crossplane.io[4] is a Kubernetes add-on for inter-cluster management. Its aim is to allow multiple Kubernetes clusters to run with each other with a unified API.

It is an Open-source project, released under the Apache 2.0 license. The community is huge and very active. It is a Cloud Native Computing Foundation project.

It proposes a CLI-based management system for hyper-scalars (Amazon Web Services, Google Cloud Platform, Elastic Kubernetes Service). Besides, a commercial solution with a graphical interface is also proposed.

This solution is not yet readily usable for bare metal platforms, which hinders its use within the DEDICAT 6G project.

3.2.3 Rancher

Rancher[5] is an open-source multi-cluster management platform that runs on top of Kubernetes. The company that distributes the solution also proposes a support, for a charge.

It has a user-friendly graphical interface to manage the clusters. It supports various Kubernetes distributions (including RKE, K3S, K8S, and GKE…). Rancher can replace the account management in Kubernetes, for easier user management among the different clusters.

It is very close to a production-ready platform, but the API is not rich and flexible enough for easy integration with the DEDICAT 6G platform.

3.2.4 Kubefed

Kubefed[6] allows to use a single Kubernetes cluster to coordinate multiple Kubernetes clusters. It can deploy multi-cluster applications.

[3] https://kubernetes.io

[4] https://crossplane.io/

[5] https://rancher.com/

[6] https://kubernetes.io/

Kubefed is open source. The community of developers is huge, and it was chosen by RedHat for their solution for managing all types of clusters. RedHat is also a main actor for the development of Kubefed.

Kubefed does not provide a graphical interface, but it allows a fine granularity for the clusters management, down to the namespace level or API group, for instance.

3.2.5 Anthos

Anthos[7] unifies the management of infrastructure and applications across on-premises, edge, and in multiple public clouds with a Google Cloud-backed control plane. It is also able to manage bare-metal, hyper-scalars, VM, server-less.

Since recently, Anthos can be installed on bare-metal in order to provide an edge computing solution. Anthos provides a new concept called 'environ' that allows to define a namespace across several clusters.

Anthos is a black-box proprietary solution; therefore it might not be suitable as a building block on top of which the DEDICAT 6G platform is built.

3.2.6 NFV Orchestration

Network Function Virtualization (NFV) is one of the key technologies that underpins the 5G emergence and it is considered as one of the main enablers for the future roll-out of B5G/6G networks. NFV is based on the virtualization of network functionalities, called as Virtual Network Functions (VNFs), originally implemented in bare-metal devices, but in this case deployed in virtual machines or containers in order to be easily accommodated along all network domains, for instance, remote cloud servers or devices with some computational availability placed at the edge. According to the NFV standards, the management and orchestration (MANO) procedures and tasks are in charge of the main actor in NFV, the NVF Orchestrator (NFV-O), more details below.

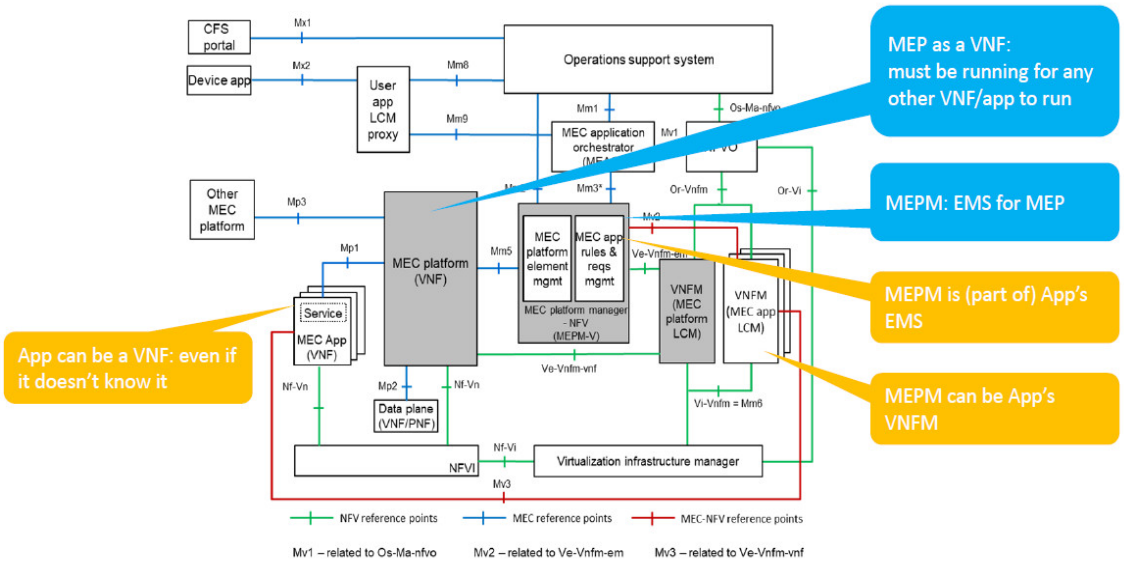


Figure 4: ETSI MEC and ETSI NFV standards interoperation

Given their software-based nature, VNFs will consume computational resources in the system

[7] https://cloud.google.com/anthos

to perform their specific functions or roles and therefore, in accordance with the above definitions of intelligence in this document, should be considered in the dynamic intelligence distribution mechanisms of the DEDICAT 6G scenario. Within the DEDICAT 6G context, NFV is a transversal technology that will be involved in two of the main pillars of this project, intelligence distribution (WP3) and coverage extension and connectivity (WP4). More details about NFV orchestration, in general, and how to be applied to coverage extension, in particular, can be consulted in D4.1 [18].

As key part of the 5G ecosystem, telco enterprises, network equipment vendors and standardization bodies have put the light on NFV and as consequence the technology is settled by specific standards. Thus, the European Telecommunications Standards Institute (ETSI) created a specific Industry Specification Group, the ETSI ISG NFV[8], to define the structure, challenges, and evolution of NFV. Moreover, the ETSI ISG is totally aligned to many 5G/B5G-related standards, like ETSI MEC, that leads the standardization activities around the Multi-Access Edge Computing (MEC)[9], or ETSI ENI, devoted to the definition of cognitive network management architecture with the assistance of Artificial Intelligence[10]. Figure 4 illustrates a basic architecture with MEC and NFV principles that are to be applied under the same context, where vertical applications are deployed by leveraging the MEC capabilities in a virtualized infrastructure (NFV-I) orchestrated and coordinated with the NFV-O assistance. The key here is that some MEC components can be deployed in the NFV-I as VNFs, this way enabling their management by the NFV-O.

Furthermore, one of the main results of the ETSI ISG effort is the creation of an Open-Source Platform to play the role of NFV-O, called as Open Source MANO (OSM)[11]. OSM is one of the most widely used tool for NFV-O in academia, research projects and even in industry. OSM not only serves a complete set of functionalities related to MANO operations, but also is providing two key offering for the DEDICAT 6G purposes:

- a standardized API, based on ETSI SOL005 [19] specifications, to be remotely controlled or executed,
- and an ETSI-based information model to create descriptors to define the deployment or instantiation details (among others) of VNFs, network services or even network slices following standardized rules (SOL006, cf. [20]).

For instance, in the SOL006 [20] information model we can specify some features totally aligned with the distribution of intelligence, such as CPU, storage or RAM to be consumed by the VMs associated to the VNFs, or the internal network settings in the NFV-I, or where to deploy the VNFs, thus indicating the distribution of the VMs/containers related to network functionalities. Further information about OSM can be found in D4.1 [18] and [6].

In Figure 5, we can see an exemplary diagram of how NFV (OSM as NFV-O) can be applied under the assistance of the DEDICAT 6G platform in intelligence distribution operations. In this

[8] ETSI, "ETSI ISG NFV", Online : https://www.etsi.org/technologies/nfv

[9] ETSI, "Multi-access Edge Computing (MEC)", Online: https://www.etsi.org/technologies/multi-access-edge-computing/mec

[10] ETSI, "Experiential Networked Intelligence (ENI)", Online: https://www.etsi.org/technologies/experiential-networked-intelligence

[11] https://osm.etsi.org/

case, after running a given intelligence distribution algorithm provided by the Decision-Making FG and with the assistance of the Intelligence Distribution and Context-Awareness FGs, the output of such execution is forwarded to the Orchestration FG. Here, it is in charge to convert the recommendations to a concrete template/blueprint based on the SOL006 [20] information model to be readable by OSM. Then, an OSM client available in the Communication FG would transfer the VNFs' placement recommendations, by exploiting the OSM API, to be translated to the network system (in edge or cloud nodes), defined in the SOL006-based template/blueprint. This way, the DEDICAT 6G platform could, for instance, command or assist OSM in the instantiation of network slices by indicating the distribution (where to place) of a concrete type of distribution, VNFs or the computational resources to be consumed by the VMs/containers associated to the VNFs.

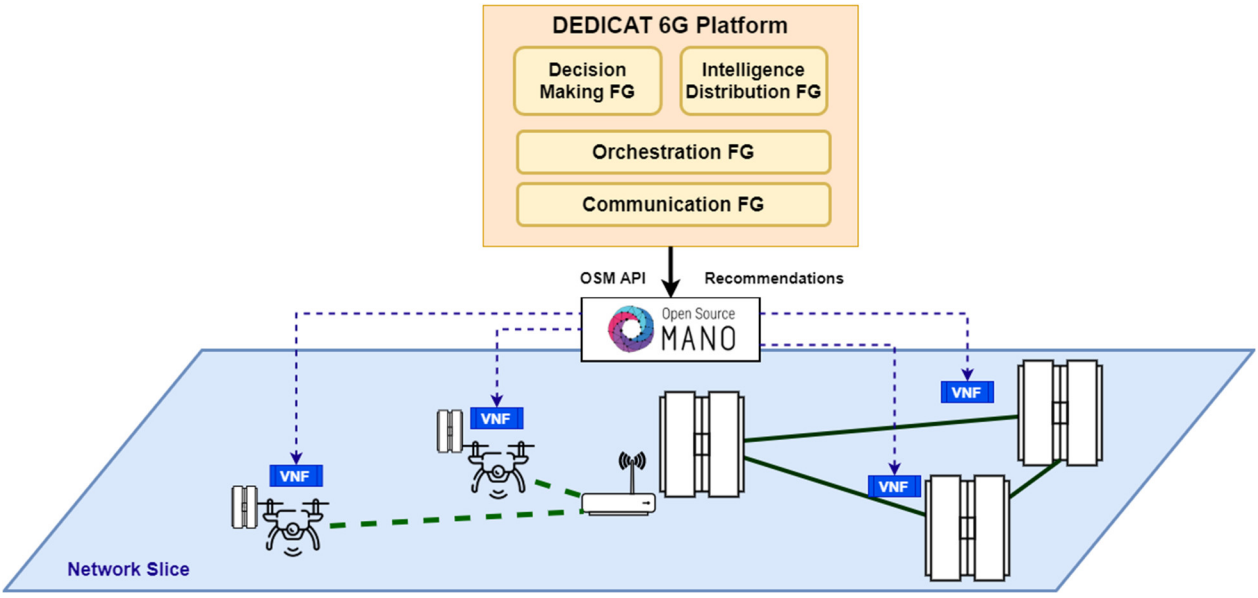


Figure 5: DEDICAT 6G Intelligence Distribution assistance with NFV capabilities

4 Algorithms for Distribution of Intelligence

In this section, we describe the preliminary work on algorithm design for intelligence distribution decision-making functional components.

4.1 Placement of Intelligence

As part of the specification and development of the mechanisms which will enable the dynamic distribution of intelligence (data, computation, storage), this sub-section presents an intelligence functions placement algorithm which ensures efficiency of the system, with respect to a chosen KPI, or a set of KPIs. This algorithm can be applied to all four use-cases. The problem statement is the following.

4.1.1 Problem Statement

We consider a set of Functional Entities (FEs) $F = \{F_1, F_2, \ldots, F_i, \ldots, F_n\}$ e.g., tasks, jobs, services and a set of Hosting Entities (HEs) $H = \{H_1, H_2, \ldots, H_j, \ldots, H_m\}$ e.g., edge nodes, core nodes, robotic units, end user devices, Virtual Machines, containers. Each FE has a processing time, a set of successors FEs and a computational load denoted by φ_i. The communications between FEs are represented by the functional graph (service graph), which is denoted by G_f

= (F, K). Each node F corresponds to a FE and each edge K connects interacting FEs and is weighted by, $k_{i,i'}$, according to the amount of data transferred between FEs F_i and $F_{i'}$.

Moreover, each HE H_j has some capabilities. These are the maximum computational load c_j which is related to the number of available CPU cores the available RAM and disk storage, the battery level if applicable and the functionality-wise E_j, which is a set that consists of FEs that the HE H_j can support ($E_j \subseteq F$). It is also considered a system layout graph G_H = (H, 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 communication cost, denoted as $r_{j,j'}$.

Our objective is the allocation of FEs to HEs. Let A_j, denote the set of FEs that will be assigned to HE H_j, $A_j \subseteq F$. We aim to optimize the objective function that satisfies a set of performance constraints. Specifically, we are looking for the minimum cost allocation that satisfies a set of performance constraints.

The objective function is associated with:

- the cost of utilizing a HE b_j, mostly related to the battery level of the HE if applicable. This cost takes higher values when battery is low and close to zero values when it is fully charged or when the HE is not battery-powered.
- the power consumption cost $e_{i,j}$ of running the FE F_i on the HE H_j.
- the computational cost $s_{i,j}$ of running the FE F_i on the HE H_j, which is related to the computational load φ_i of the FE and the maximum computational load c_j of the HE,
- the cost (latency), $r_{j,j'}$, imposed by the communication among HEs, which may be related to the availability of the HE to conduct the job (i.e., standby mode) etc.

The constraints of our problem address the following aspects:

- All FEs need to be assigned to the available HEs.
- All capabilities of HEs need to be respected.
- The capacity constraint of each communicational link should be respected.
- We assume that the communication cost between two FEs F_i, $F_{i'}$ executed by the same HE, is negligible ($k_{i,i'} \approx 0$)

4.1.2 Problem Formulation

We introduce the set of decision variables y_j:

$$y_j = \begin{cases} 1, & if\ H_j\ is\ utilized \\ 0, & if\ H_j\ is\ not\ utilized. \end{cases}$$

We introduce the set of decision variables $x_{i,j}$ to describe the allocation of FEs to HEs:

$$x_{i,j} = \begin{cases} 1, & if\ F_i\ is\ assigned\ to\ H_j \\ 0, & if\ F_i\ is\ not\ assigned\ to\ H_j. \end{cases}$$

Moreover, we define the set of decision variables $z_{j,j'}$, to describe the communication among HE:

$$z_{j,j'} = \begin{cases} 1, & if\ H_j\ and\ H_{j'}\ are\ communicating \\ 0, & if\ H_j\ and\ H_{j'}\ are\ not\ communicating. \end{cases}$$

Finally, we introduce the set of given binary variables $w_{i,j}$, to indicate the feasibility of assigning a FE to a HE in terms of functionality-wise, if, in general, the HE can support the FE:

$$w_{i,j} = \begin{cases} 1, & if\ F_i \subseteq E_j \\ 0, & otherwize \end{cases}$$

The problem of obtaining A_j may be reduced to the following problem:

$$min \sum_{j=1}^{m} y_j b_j + \sum_{i=1}^{n}\sum_{j=1}^{m}[x_{i,j} * e_{i,j} * w_{i,j}] + \sum_{i=1}^{n}\sum_{j=1}^{m}[x_{i,j} * s_{i,j} * w_{i,j}] + \sum_{j=1}^{m-1}\sum_{j'=j+1}^{m}[z_{j,j'} * r_{j,j'}]$$

Subject to:

- $\sum_{j=1}^{m} x_{i,j} = 1 , \forall\ 1 \le i \le n,\ i \in \mathbb{N}$, all FEs are allocated,
- $\sum_{i=1}^{n}[x_{i,j} * \varphi_i] \le c_j ,\ \forall\ 1 \le j \le m,\ j \in \mathbb{N}$, the maximum computational load of the HEs is respected.
- $\sum_{j=1}^{m} w_{i,j} \ge 1 , \forall\ 1 \le i \le n,\ i \in \mathbb{N}$, all FEs can be assigned to at least one HE in terms of functionality-wise.
- $\sum_{i=1}^{n}[x_{i,j} * y_j] \ge y_j , \forall\ 1 \le j \le m,\ j \in \mathbb{N}$, all HE that are utilized ($y_j = 1$) have at least one FE assigned on them.

The notation described above can be found in Table 2. The constraint of the maximum capacity of the links among HEs will be part of a future investigation.

Table 2: Intelligence Functions Placement notations

Notation	Definition
$F = \{F_1, F_2, \ldots, F_i, \ldots, F_n\}$	Set of Functional Entities (FEs)
$H = \{H_1, H_2, \ldots, H_j, \ldots, H_m\}$	Set of Hosting Entities (HEs)
i, i'	Indexes of FEs
j, j'	Indexes of HEs
n	Total number of FEs
m	Total number of HEs
φ_i	Computational load of FE F_i
$G_f = (F, K)$	Functional graph with nodes F and edges K
$k_{i,i'}$	Weights of interacting FEs F_i, $F_{i'}$
c_j	Maximum computational load of HE H_j
E_j	Functionality-wise of HE H_j
b_j	Cost of utilizing a HE (related to the battery level)
$s_{i,j}$	Computational cost of running FE F_i on HE H_j
$G_H = (H, L)$	System layout graph with nodes H and computational channels L

$r_{j,j'}$	Cost of communication between HEs H_j and $H_{j'}$
$e_{i,j}$	Power consumption cost of running FE F_i on HE H_j
A_j	Set of FEs that will be assigned to HE H_j
y_j	Takes 1(0) depending on whether HE H_j is (is not) utilized
$x_{i,j}$	Takes 1(0) depending on whether FE F_i is (is not) assigned to HE H_j
$z_{j,j'}$	Takes 1(0) depending on whether HE H_j and $H_{j'}$ are (are not) communicating
$w_{i,j}$	Takes 1(0) depending on whether FE F_i can (cannot) be assigned to HE H_j in terms of functionality-wise

4.1.3 Execution Environment / Deployment

The described intelligence functions placement algorithm is planned to be implemented in Python, will be containerized, and deployed using Docker. Also, an API documentation will be created for this algorithm by using the Swagger open-source tool.

In general, this algorithm is mapped to the Intelligence Distribution-Decision Making FC (IDDM FC) of the Decision Making FG described in §2.5.4. It is going to be closely connected with the Service Registry FC (§2.5.1), Orchestrator FC (§2.5.5), Platform Performance FC (§2.5.3) and Edge Node Awareness FC (§2.5.2). The Platform Performance FC will trigger the intelligence functions placement algorithm when detecting a performance anomaly or error. The Service Registry FC is going to inform the intelligence functions placement algorithm on services instantiation. The Edge Node Awareness FC is going to give resources (cloud, MEC, network, extreme edges) information to the intelligence functions placement algorithm. Finally, the intelligence functions placement algorithm will instantiate new service deployment etc. to the Service registry FC and the Service registry FC will send this information to the Orchestrator FC to execute the decision.

4.2 Placement of services in Smart Warehouse

4.2.1 Problem statement

One key aspect of the decision-making algorithm is to decide where to place the services and the data. Indeed, these placements have a major impact on the allocation of resources for services, and on the perceived quality of service by end-users.

In this subsection, we consider the problem of service placement in the smart warehouse. The aim is to decide where to locate the execution of the tasks needed by services requested by end-users.

An example of such services is Augmented Reality (AR): the end-user terminal (e.g. connected glasses) is live streaming to an application in the cloud, which in turn processes the images and encodes information in it. The resulting enriched video stream is sent back to the terminal which displays it. This service is characterized by a very strict latency requirement, because the enriched video stream must be almost on sync with the original one. It is also characterized by requiring a lot of computing resources for processing the original video stream, which prevents the processing to be performed by the terminal itself.

Another example of service is the same as augmented reality, except the enriched stream is sent to several terminals, and also to the cloud for storage. The problem is even more difficult in that case.

In order to design the placement algorithm, the first step is to model the services on one hand and the underlying infrastructure on the other hand.

Services are modelled as one task, and several flows connecting the task with different destinations.

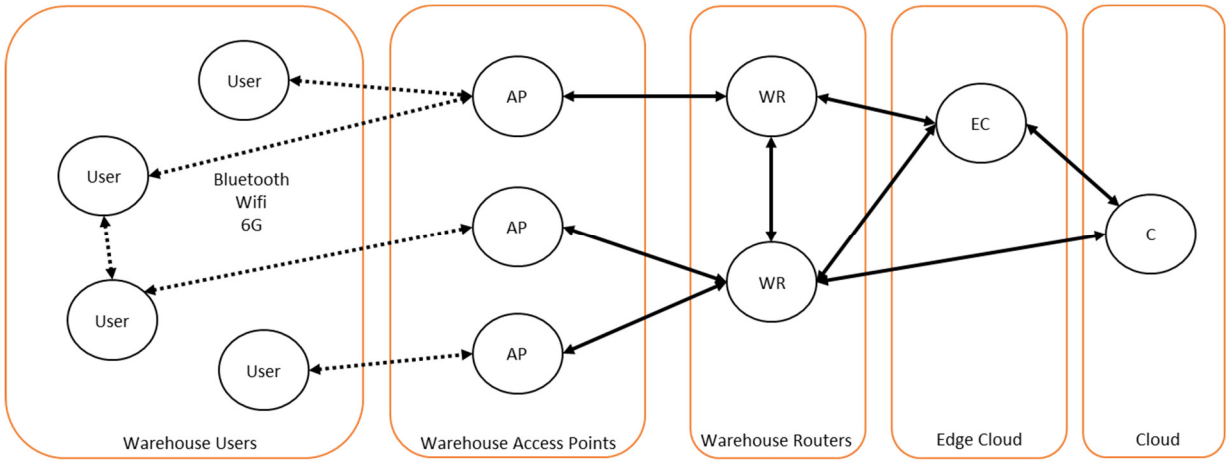


Figure 6: Illustration of users, edge cloud and cloud in the smart warehouse

Figure 6 gives an illustration of the network elements in the infrastructure, which have to be taken into account in a particular use-case, namely the Smart Warehousing use-case.

In this problem, we have to consider the capacity of the network links, for two reasons: firstly, most of the services have bandwidth requirements, so they have to share the link resources, secondly, experience has shown that the latency induced by a link is dependent on the load on this link. As a consequence, we need also to model the latency of a link as a function of its load. As a first approximation, we assume in the following that the latency is an affine function of the load, i.e. a function of the form: $\mathrm{L}(\mathrm{x}) = \propto x + \beta$, with L the latency and x the load.

Each service has a requirement in term of a maximum latency that must be met. The latency of the service is computed by adding the latency of each link crossed by the service traffic. Each flow of a given service may have different latency requirement. With the example given above, the flows of AR streams to the AR glasses have a stricter latency requirement than the flow going to the cloud for storage.

In addition, services have requirements in terms of minimum resources that must be available on the host on which they run. These resources requirements include computation (CPU or GPU), memory, storage, electrical power. These resources are additive, which means that the total resource of each type used on a particular node is the sum of the resources of all the services running on this node.

The flows from the service task may have variable throughput. Indeed the service may still be functional even with a degraded quality of service. In certain conditions, it might be necessary to lower the throughput of some flows in order to meet the latency requirement of the flow. Each service should define whether or not they accept a lower throughput, and in which range. Of course, when possible, the flows should be at the highest level of throughput,

so as to offer the best quality of service as possible.

4.2.2 Problem Formulation

We model the network and the end-user terminals as a digraph graph G = (E, A). E is the set of nodes and A is the set of arcs (i.e. the directed edges).

Each flow is given a profit value, i.e. a weight for its throughput. The objective function is to maximize the weighted average flow throughput.

In addition, Table 3 sums up the notations used in the following.

Table 3: Notations for the placement of services in Smart Warehouse

Sets	
E	Set of nodes (index u)
A	Set of arcs
S	Set services (index s)
F_s	Set of flows of service s (index f)
T	Set of possible throughputs for flows (cardinality N)
Constants	
R_s	Vector of resources required by the task of service s
P_s	Profit of service s
L_f	Maximum latency of flow f
d_f	Destination of flow f
C_u	Capacity of edge u (vector)
B_a	Bandwidth capacity of arc a
Other notations	
t_a	Total traffic of arc a
l_a	Latency of arc a, we have $l_a = \alpha_a T_a + \beta_a$
variables	
$x_f \in T$	Throughput of flow f
$y_{su} \in \{0, 1\}$	Indicator of task of service s placed on edge u
$z_{f(u,v)} \in \{0, 1\}$	Indicator of flow f using arc (u, v)

With the given notations, a compact formulation for the problem is the following:

$$\max \quad \sum_{s \in S} P_s \sum_{f \in F_s} x_f \tag{1}$$

subject to

$$\sum_{v \in \delta(u)} \left(z_{f(u,v)} - z_{f(v,u)} \right) = \mathbb{1}_{u=u_0} - y_{su} \qquad \forall u \in E, \forall s \in S, \forall f \in F_s \tag{2}$$

$$\sum_{s \in S} y_{su} R_s \leq C_u \qquad \forall u \in E \tag{3}$$

$$\sum_{s \in S} \sum_{f \in F_s} x_f z_{fa} \leq B_a \qquad \forall a \in A \tag{4}$$

$$\sum_{a \in A} z_{fa} \left[\alpha_a \left(\sum_{s' \in S} \sum_{f' \in F_{s'}} x_{f'} z_{f'a} \right) + \beta_a \right] \leq L_f \qquad \forall s \in S, \forall f \in F_s \tag{5}$$

$$x_f \in T = \{t_1, \ldots, t_N\} \qquad \forall s \in S, \forall f \in F_s \tag{6}$$

$$y_{su} \in \{0, 1\} \qquad \forall s \in S, \forall u \in E \tag{7}$$

$$z_{fa} \in \{0, 1\} \qquad \forall s \in S, \forall f \in F_s, \forall a \in A \tag{8}$$

The set of constraints (1) are the flow conservations constraints. The symbol δ(u) represents the neighbours of node u. The symbol u_0 represents the destination of flow f (recall its origin is the task of service s). Constraints (2) express the node capacity limit, and constraints (3) express the capacity of the arcs. Finally, constraints (4) enforce the latency requirements on the flows.

The resulting compact formulation is clearly not linear, because of constraints (4). In order to ease the resolution of this problem, we will propose a linearized version of this formulation. This is currently work in progress.

4.3 Placement Optimization in Smart Highway

4.3.1 Multi-Objective Optimization

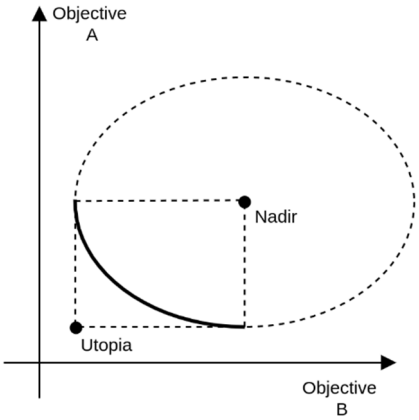
The field of Multi-Objective Optimization (MOO) considers the problem of optimizing several conflicting objectives. The generic shape of the problem definition in MOO is defined below, where F_k represents the k-th objective function over solution $\boldsymbol{x}$, g_j represents the inequality constraints and h_l represents the equality constraints.

$$\underset{\mathbf{x}}{\text{Minimize}}\ \mathbf{F}(\mathbf{x}) = [F_1(\mathbf{x}), F_2(\mathbf{x}), \ldots, F_k(\mathbf{x})]^T$$

$$\text{subject to } g_j(\mathbf{x}) \leq 0, \quad j = 1, 2, \ldots, m,$$

$$h_l(\mathbf{x}) = 0, \quad l = 1, 2, \ldots, e,$$

Working with multiple conflicting objectives will not provide a single optimal solution. Rather, it will provide a set of non-dominant solutions, where each solution improves on an objective but worse on another. This set of solutions form the Pareto Front (PF), depicted below.



This Pareto front contains all optimal solutions regarding a MOO problem. However, to optimize in a practical environment, a single solution from this PF is needed. This can be found by one of three approaches:

- A priori optimization: User preferences are modelled in the problem in advance, and the algorithm finds a single solution
- Interactive optimization: The Decision Maker (DM) interacts and fine tunes the preferences to find a single solution
- A posteriori optimization: The entire Pareto front is found, and a DM selects the solution

As the intelligence distribution service works autonomously, a priori optimization will be applied. Preference definition will be pushed towards the DM.

4.3.2 Multi-Objective Reinforcement Learning

The field of Multi-Objective Reinforcement Learning (MORL) is a promising field in which an RL agent optimizes multiple conflicting objectives concurrently. The environment will be modelled as a Multi-Objective Markov Decision Process (MOMDP), providing a general framework in which the agent can participate. In general RL, the agent will take actions in an environment which influence this environment. These actions cause the environment to change state, providing the agent a new observation of the environment, and a reward on how useful his action was. Using these rewards, the agent learns a policy on how to behave within a certain state of the environment to achieve the highest possible rewards.

In the field of MORL, this single reward becomes a reward vector, with each value in the vector representing a specific objective. This vector is then used in a variety of different ways to learn a policy which takes into account all objectives.

One of the common ways to do this is by building a utility function which takes into account all objectives and scalarizes them to a single value. This approach enables the usage of single-objective optimization techniques. One of the most common scalarization utility functions is the weighted sum approach. This approach multiplies each objective with a predefined weight and sums them together. These weights represent the user preferences, with each weight prioritizing a different objective. Although one of the simplest solutions, this approach has its drawbacks, such as having difficulties with finding pareto-optimal solutions.

As defined by Radulescu et al. [31], these utility functions can be integrated into the MORL agent in two distinct ways. The first integration is using the Expected Scalarized Returns (ESR)

description. This approach tries to optimize the objectives over a single run.

$$V_u^{\pi} = \mathbb{E}\left[u\left(\sum_{t=0}^{\infty}\gamma^t \mathbf{r}_t\right) \mid \pi, \mu_0\right]$$

Here, V_u^{π} represents the expected returns for policy pi and utility function u. γ^t represents the discount factor at timestep t, with $\boldsymbol{r}$ representing the vector of rewards/objectives on policy π and μ_0 being the initial distribution over the states.

The second integration is using the Scalarized Expected Returns (SER) description, which tries to optimize the objectives over all runs. This description fits the project problem definition for service allocation and will be the one more suited for the project.

$$V_u^{\pi} = u\left(\mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^t \mathbf{r}_t \mid \pi, \mu_0\right]\right)$$

These utility functions are used in several scenarios to support dynamic behaviour with regards to changing weights. The approach we propose to use within the project is the dynamic weights approach, first coined by Abels et al. [32]. This approach trains several networks for several sets of weights. These networks can then be used interchangeably were the weights of the problem case to change.

The agent works in a centralized environment with full observability. This enables a large degree of freedom and ensures that a global optimum can be achieved. In a first stage, the agent will be trained on a specific crossing, considering the possible cars connected to the network. In a next stage, this approach can be extended to work using Graph Neural Networks. This approach extends the input of the neural network to work with any size of graph.

4.3.3 Centralized Optimization

Centralized optimization techniques can range from Mixed Integer Programming to heuristic approaches and even Machine Learning methodologies such as Reinforcement Learning. These methodologies have the added benefit of utilizing the global state of the network, enabling global optimization. This does, however, have the increased cost of communicating the entire state to the node running the centralized optimization algorithm, which can induce network load and outdated views of the network. Global optimization helps ensuring that all applications are treated fairly and have an equal share of the resources. Centralized optimization techniques generally also better support migration optimization, since it can take into account the previous global state when finding an improved placement. The project considers multiple KPIs, such as latency and energy consumption minimization. These KPIs can be represented as multiple conflicting objectives, which will be simultaneously optimized by the selected algorithm. Multi-objective optimization suffers from normalization issues, since the metrics which are compared are usually not in the same magnitude, with for example the bandwidth metric being in the megabit/gigabit range and latency in the millisecond/microsecond range. This does not matter in pure multi-objective optimization, as all non-dominating solutions (solutions which are better on one objective but worse on another)

are kept, but the magnitude difference does matter for the project, as only a single solution will be applied based on the user preferences. These preferences are generally defined within the range [0, 1] and require the objectives to be normalized. The project architecture is not designed with a human in the loop, and as such requires a fully autonomous system. In order to select the best solution from a set of multiple competing solutions with regard to objectives, a preference is required. This preference can be achieved by using weighted sum scalarization techniques, which multiplies each objective with its specific weight, and then sums all objectives together. Another alternative is lexicographic ordering, where the objectives are ordered by importance and used to compare the quality of solutions.

Multiple state-of-the-art multi-objective optimization algorithms are already available. These include metaheuristic approaches such as the NSGA-II algorithm [39], a bio-inspired algorithm which evolves a solution set towards the optimum. Similarly, several Reinforcement Learning (RL) algorithms are available which learn the optimal behaviour for task allocation. Such approaches include universal value function approximation, generalizing over the available states and actions.

Machine learning methodologies can also be used in an offline fashion, where the agents are trained on a dataset or simulation of the use-case, learning the behaviour of the environment without physically interacting with it. This requires a large dataset of the specific environment, or an accurate simulator. Such offline approaches often suffer from the sim2real gap [48], where there is a difference between the offline learned policy and the actual optimal policy of the online environment.

4.3.4 Distributed Optimization

Distributed optimization techniques for placement optimization tries to tackle the problem from the decentralized perspective, where there is no single point of failure. Distributed algorithms tend to optimize more toward local optima, since they lack a global state of the environment. This lack of global state does reduce the global network overhead since no global state needs to be built, but this is compensated by the interactions between multiple agents of the distributed algorithm. Such techniques tend to be more able to immediately act on changes in the environment, since there is no overhead of communication with the centralized algorithm. This does depend on the selected algorithm, as this is not always the case: finding a consensus with the entire network can also take longer than in a centralized network.

Distributed optimization techniques are often built on game theory, with one example being the Contract Net Protocol [47], where multiple agents communicate and work together to find a consensus on where to offload the task

Machine learning methodologies also exist for distributed optimization, where multi-agent multi-objective reinforcement learning agents try to achieve a consensus on the optimal placement. This can be achieved using fully distributed methodologies, such as Gossiping RL, or using hierarchical methodologies, such as Feudal RL [35], where a manager tells multiple RL agents what they should be optimizing towards.

4.3.5 Placement Methodology

The placement of software components across the edge nodes will be optimized utilizing a multi-agent multi-objective approach. This approach will consist of an agent, the local network configurator, dynamically optimizing the network link connections between multiple services, and another agent, the global orchestrator, optimizing the placement of the requested services across the available resources. This orchestrator works at a lower temporal

resolution as the network configurator. This separation of concerns enables the network configurator to support URLLC scenarios, while the orchestrator ensures the global optimum of the placement across the dynamic resources without having significant impact on the QoS due to migration costs.

4.3.6 Orchestrator

A centralized orchestrator will ensure optimal placement of the software services across the network. The centralization of intelligence ensures that the algorithms will be able to define a global optimum, rather than multiple local optima which might reduce the efficiency of the global system in terms of latency and energy consumption. The centralized approach ensures that there is no resource competition from multiple independent service orchestrators, which could result in either resource starvation due to overloading of the edge node, or in unsafe scenarios where the target service is placed on a resource too far away from the source service for the services to function correctly. A third consideration for the centralization of intelligence is the support for services with a considerable start-up time. In a decentralized approach, these services would be started once a vehicle joins the network, but in this centralized approach, predictive placement could be utilized to anticipate on the network load and place certain services in advance. This orchestrator can also include the migration cost and QoS assurance: in the scenario where a service migrates between two nodes, the orchestrator can ensure that both stay available until all links have been moved to the new instance of the service. Additionally, the centralized algorithm can more easily support stateful migration. The methods utilized for this approach consist of state-of-the-art multi-objective reinforcement learning methodologies.

4.3.7 Link Optimization Methodology

To support low-latency QoS assurance, a local link optimization methodology is proposed. This multi-agent approach ensures that each connection from the VRU or the OBU to the RSU supports the requirements needed by the link. Each agent will optimize its own link based on the knowledge of the network and will continuously monitor and adapt to changes. The agent will be running on the side of the OBU and VRU and will be monitoring the available network resources in its local area. Due to its locality, the agent will be doing independent local optimization instead of global optimization, to increase response time and reduce network load.

The proposed methodology uses a Multi-Agent RL approach. In this approach, a reinforcement learning agent optimizes the selection of the used link based on the requirements and the availability. Chu et al. [37] compared both independent Advantage Actor Critic (A2C) and independent Q-Learning to a novel methodology in a cooperative setting. Although our setting is a cooperative setting as well, since the agents will get the most optimal policy if they work together, an independent approach might still be useful. This is due to the limited observability of the problem, requiring agents to know what the other agents are doing. To partially resolve the issue of the partial observability with decentralized agents, an approach is proposed by Zimmer et al. [49], who proposed a fair policy learning model for a MARL context. This approach uses a Centralized Training, Decentralized Execution (CTDE) for the agents. This allows the agents to cooperate and learn each other's behaviour when learning, which can in turn be used when they are executed decentralized over all respective computing nodes. This approach removes the partial observability, since the agents know what the other agents are doing. This approach will have difficulties with further online learning.

4.3.8 Intelligence Migration

One example for intelligence migration at the edge can be found in UC4, this also illustrates how intelligence and awareness are being handled and exchanged between different actors in the connected autonomous driving scenarios and sub-scenarios.

Vehicular Ad-hoc Networks (VANETs) mainly have two types of communication. Vehicle-to-Vehicle (V2V) communication and Vehicle-to-Infrastructure (V2I) communication. Communication systems must operate with low latency and ensure high reliability. Aiming to provide computing services close to the devices that need them, Mobile Edge Computing (MEC) is a potential solution that ensures reliability and low latency. MEC allows autonomous vehicles to offload resource-intensive tasks and run applications on multiple platforms. Mobile Edge Hosts are typically installed on roadside units (RSUs) or can be physically located close to the RSUs.

In V2I communication, the vehicle receives information about surrounding traffic information, pedestrians, cyclists, and vehicles on the road in real-time, greatly improving driving safety and comfort. RSU is an essential part of V2I, and the RSU distribution in the vehicle network has a significant impact on the performance of V2I communication.

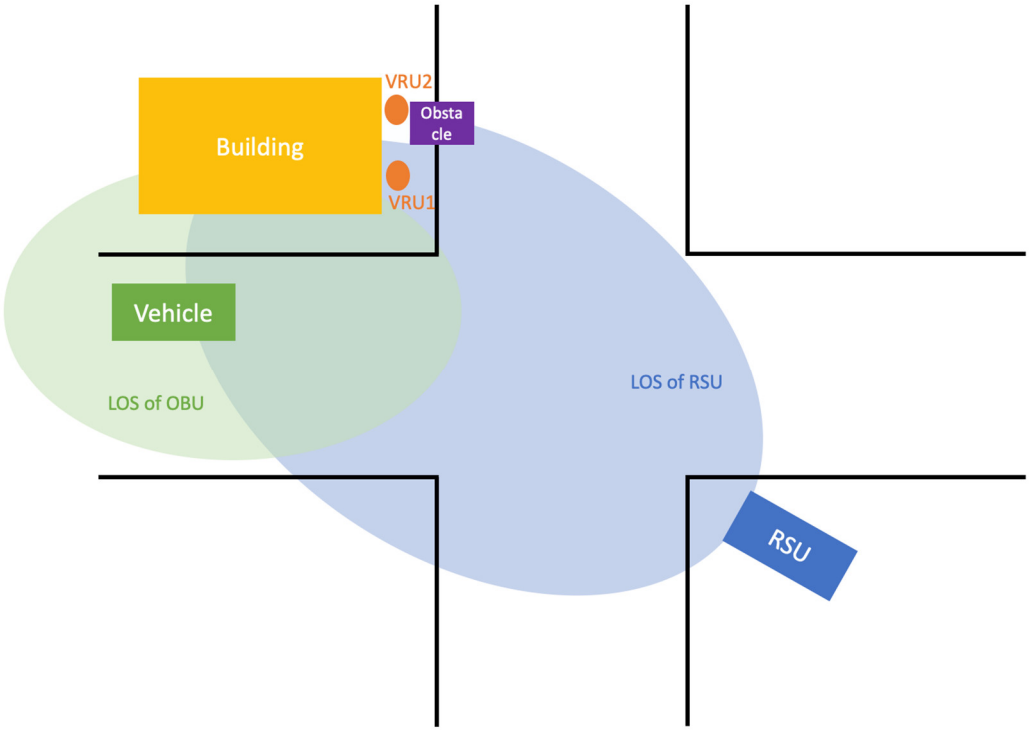


Figure 7: Example of VRU, RSU, and vehicle on the road

Buildings, fences, plants, and other obstacles can limit visibility at city traffic intersections, which can easily create dangerous situations. Thus, vehicle drivers approaching the intersection have limited reaction time when other obscured road users, such as pedestrians and cyclists, appear unexpectedly. The RUS is installed in the infrastructure to support real-time decision-making by drivers by transmitting safety-critical object data and hazard warnings to the vehicle in vehicle-to-road collaboration scenarios. For these tasks, the RSU is equipped with sensors, radio communication, and data processing units.

The RSU is used to collect intelligence vision data from objects outside the driver's field of view. The safety-critical object data information and its coordinates are transmitted from the RSU to the vehicle in real-time. This information is categorized and highlighted for the driver

on the screen to support real-time decision-making.

In Figure 7, since VRU1 and VRU2 are located in the vehicle's NLOS (Non-Line of Sight), it is impossible to recognize them in the OBU (On-Board Unit) mounted on the vehicle. On the other hand, VRU1 and VRU2 are located in the LOS (Line of Sight) of the RSU. The RSU can detect VRU1 and VRU2 provide their information to the vehicle. Although VRU2 is located in the LOS area of the RSU, it is difficult to accurately detect the VRU2 from the RSU due to obstacles on the road. In this case, since the location information of VRU2 is transmitted to the RSU through the VRU app, the VRU2 that is obscured by road obstacles can be integrated at the RSU side. In other words, the VRU app is responsible for estimating the current location of the VRU and sending that information to the RSU near the VRU location. The RSU then integrates the information sent by the VRU app with the VRU information detected by the RSU's vision sensor and forwards it to the vehicle.

Figure 8 shows the structure to be considered in a specific V2I use case. Information between RSU, vehicle, and VRU apps is transmitted through a cloud server. The cloud server classifies the vehicle's location received from the vehicle's OBU and the VRU location information received from the VRU app and delivers it to the RSU. In RSU, the location information transmitted from the cloud server and the VRU location information detected by the RSU's camera sensor are integrated and transmitted back to the cloud server in real time. The cloud server then analyses the integrated information and distributes it to the VRU app and vehicle. Finally, the vehicle and VRU apps determine dangerous situations and display appropriate warning messages to drivers and pedestrians.

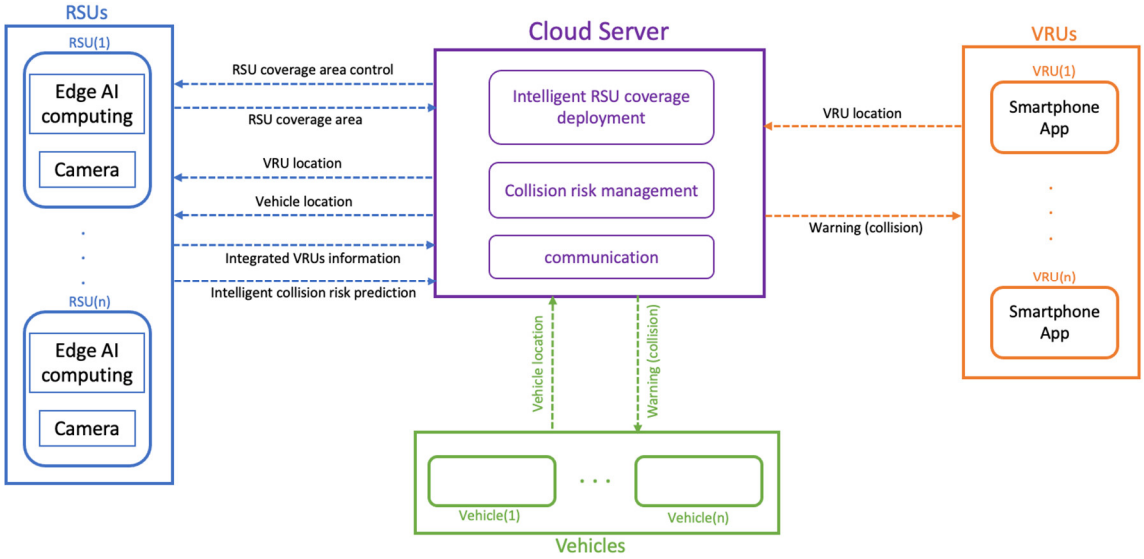


Figure 8: Structure for V2I communications consists of RSUs, Cloud Server, VRUs, and Vehicles

An example of the JSON data format transmitted between RSU, VRU, and Vehicle is shown in Figure 9, and the details of the JSON format are listed in Figure 9.

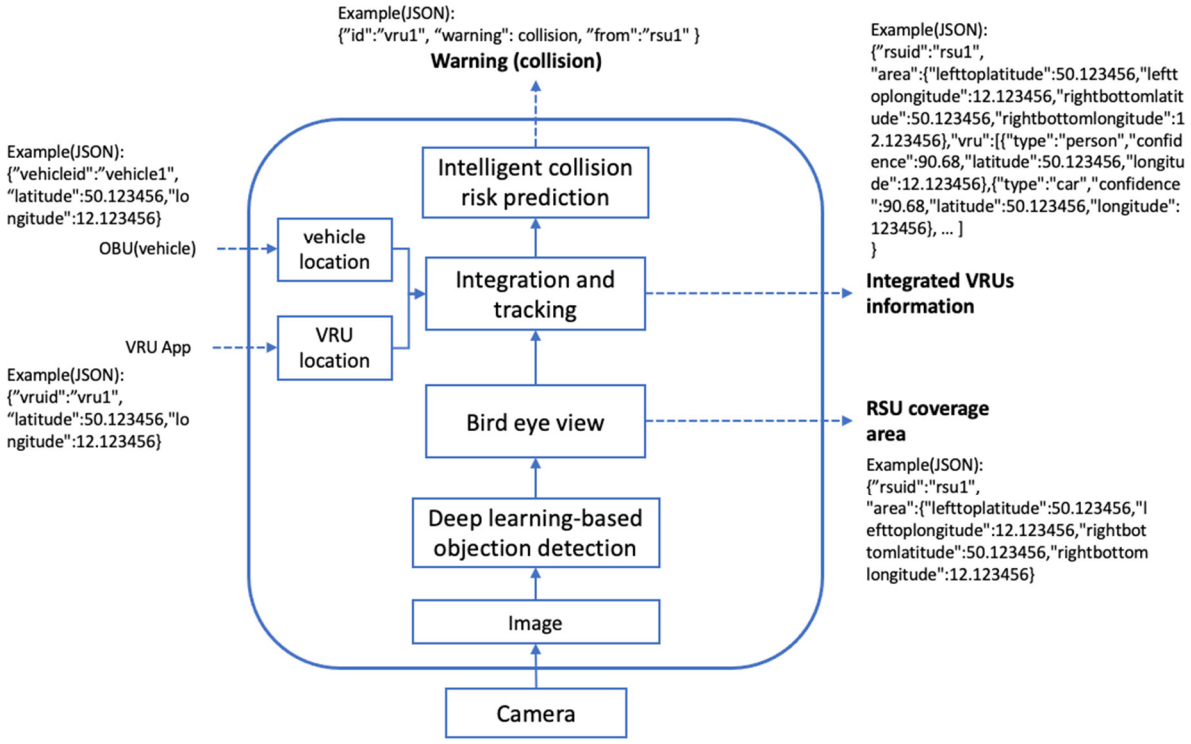


Figure 9: Example of data format transmitted in V2I

Table 4: Example of exchanged information for UC4 in JSON format

Direction	JSON	Type

Vehicle (OBU) → RSU	{
	"vehicleid": "vehicle1", "latitude":50.123456, "longitude":12.123456
	}
		▪ vehicleid: the id of vehicle
		▪ latitude: Latitude value of the vehicle's current location
		▪ longitude: Longitude value of the vehicle's current location

VRU app → RSU	{
	"vruid": "vru1",
	"latitude":50.123456,
	"longitude":12.123456
	}
		▪ vruid: the id of VRU
		▪ latitude: Latitude value of the VRU's current location
		▪ longitude: Longitude value of the VRU's current location

RSU → Cloud server	{
	"rsuid":"rsu1",
	"area":{
	"lefttoplatitude":50.123456,
	"lefttoplongitude":12.123456,
	"rightbottomlatitude":50.123456,
	"rightbottomlongitude":12.123456
	}
	}
		▪ rsuid: the id of RSU
		▪ area: Rectangular coordinate value of bird eye view of RSU detection area

RSU → Vehicle (OBU) & VRU app	{ "rsuid":"rsu1",	▪ rsuid: the id of RSU ▪ area: Rectangular coordinate value of bird eye view

	"area":{
	"lefttoplatitude":50.123456,
	"lefttoplongitude":12.123456,
	"rightbottomlatitude":50.123456,
	"rightbottomlongitude":12.123456
	},
	"vru":[{
	"type":"person",
	"confidence":90.68,
	"latitude":50.123456,
	"longitude":12.123456
	},
	{
	"type":"car",
	"confidence":90.68,
	"latitude":50.123456,
	"longitude":123456},
	…]
	}
		of RSU detection area
		▪ vru: Integrated VRU information from RSU (array form)
		▪ type: the type of VRU (e.g. person, cyclist, car)
		▪ confidence: VRU detection reliability (probability)
		▪ latitude: Latitude value of the VRU's current location
		▪ longitude: Longitude value of the VRU's current location

RSU → Vehicle (OBU) & VRU app	{
	"id":"vru1",
	"warning": collision,
	"from":"rsu1"
	}
		▪ id: the id of VRU
		▪ warning: the type of risk
		▪ from: The id value of the RSU that occurred risk prediction

4.3.8.1 Intelligence distribution and migration between RSUs

The core in the intelligence distribution of and migration between the RSUs is to determine the optimal installation locations to detect the VRU on the road. Because deployment of RSUs is expensive and requires pre-installed infrastructure for both power and telecommunication lines, the issue of intelligently and optimally determining where to deploy RSUs is one of the important tasks.

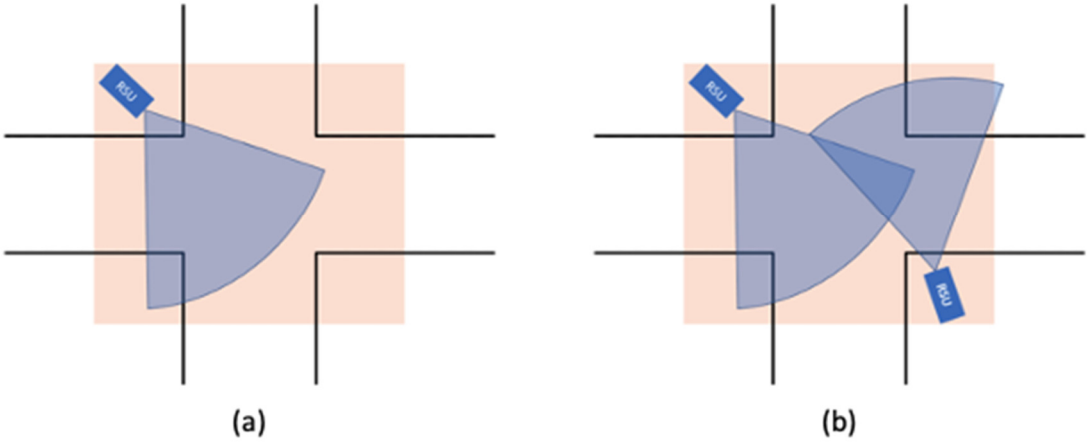


Figure 10: Placement problem of RSUs

Assuming that the blue range in Figure 10 is the area of the VRU detection sensor-equipped in the RSU, a single RSU cannot cover all intersections without blind spots. So, it needs to decide how many RSUs are needed and where to distribute them on the road.

4.3.8.2 Problem modelling

We formulate a coverage problem to solve the RSU placement problem. The field of view of a camera mounted on the RSU can be described by a triangle as shown in Figure 11 (a). The parameters of this triangle can be calculated from the camera parameters, which are given in advance. The field of view of the triangle enables us to express the area covered by RSUs camera at position (x_c, y_c). And the field of view can be translated to the origin of the coordinate system and then rotated to become parallel to the x-axis shown in Figure 11 (b).

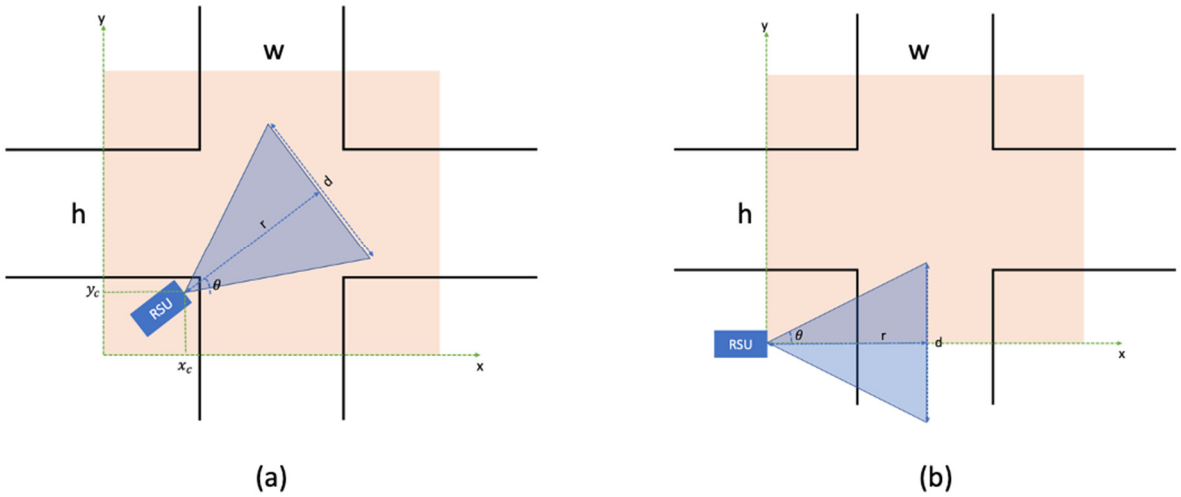


Figure 11: Model of RSU's field of view

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} cos\theta & sin\theta \\ -sin\theta & cos\theta \end{bmatrix} \cdot \begin{bmatrix} x - x_c \\ y - y_c \end{bmatrix}$$

The coordinate-transformed x', y' conditions are determined as follows:

$$x' \leq r$$

$$-\frac{d}{2r}x' \leq y' \leq \frac{d}{2r}x'$$

Therefore, the area occupied by the camera's field of view can be defined by the following three equations.

$$cos\theta \cdot (x - x_c) + sin\theta \cdot (y - y_c) \leq r$$

$$-sin\theta \cdot (x - x_c) + cos\theta \cdot (y - y_c) \leq \frac{d}{2r}\{cos\theta \cdot (x - x_c) + sin\theta \cdot (y - y_c)\}$$

$$-\frac{d}{2r}\{cos\theta \cdot (x - x_c) + sin\theta \cdot (y - y_c)\} \leq -sin\theta \cdot (x - x_c) + cos\theta \cdot (y - y_c)$$

We approximate the rectangular space with of width w and height h in a two-dimensional grid of the area that should be covered by the RSUs. RSUs can only be placed on these discrete grid points and coverage is guaranteed only for those grid points. And we assume that only one camera is equipped with an RSU and consider only one type of camera, i.e. one with the same angle of view. Given a set of grid points and a camera model, minimize

the total number of RSUs while ensuring that all grid points are covered by one or more RSUs. So, a RSU at position (x_c, y_c) with orientation θ can cover a grid point (x, y) if and only if the above three equations are satisfied.

Let a binary variable $c_{ij\theta}$ be define as:

$$c_{ij\theta} = \begin{cases} 1, & if\ a\ camera\ mounted\ on\ RSU\ is\ placed\ at\ grid\ point\ (i,j) \\ & with\ orientation\ \theta \\ 0, & otherwise \end{cases}$$

The total number of RSU is then given by

$$\sum_{i=0}^{w-1} \sum_{j=0}^{h-1} \sum_{\theta=1}^{2\pi} c_{ij\theta}$$

And a binary variable $p(i, j, \theta, x, y)$ is defined by:

$$p(i,j,\theta,x,y) = \begin{cases} 1, & if\ a\ RSU\ placed\ at\ grid\ point\ (i,j)\ with\ orientation\ \theta \\ & cover\ grid\ point\ (x,y) \\ 0, & otherwise \end{cases}$$

Thus, the RSU deployment problem can now be formulated as:

$$min \sum_{i=0}^{w-1} \sum_{j=0}^{h-1} \sum_{\theta=1}^{2\pi} c_{ij\theta}$$

Subject to

$$\sum_{i=0}^{w-1} \sum_{j=0}^{h-1} \sum_{\theta=1}^{2\pi} c_{ij\theta} \cdot p(i,j,\theta,x,y) \geq 1$$

$$\sum_{\theta=1}^{2\pi} c_{ij\theta} \leq 1$$

$$0 \leq i, x \leq w - 1$$

$$0 \leq j, y \leq h - 1$$

5 Architectural Techniques for Distribution of Intelligence

The key technique to improve the behaviour of the DEDICAT 6G networked system with respect to the Key Performance Indicators is to distribute intelligence dynamically and more optimally. To support this broad goal, we study algorithms, architecture and hardware supporting distributed intelligence. This section focuses on the two last perspectives in terms of architectural techniques and HW/SW solutions for distribution of intelligence.

The specific goals of WP3 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 subsection of this *First release of mechanisms for dynamic distribution of intelligence* deliverable describes early T3.1 work done in the field of architectural techniques and lists plans for the next period. As projected in the project plan, the considered individual techniques include efficient context switching, patterns of computation and communication, e.g., multicasting, load balancing, movement of threads, support for keeping the state of the computation simple as well as efficient computation management. These techniques are key to improving the performance of computation, i.e., reducing the latency of services provided by the network. Some techniques, such as work sharing and computation management, can be implemented with existing hardware while others, like fast context switching with multithreading and thick control flow execution, may require updated hardware.

Usage of intelligence is supported in the network by executing given functionalities in the computing nodes, such as data centers, edge servers and terminals, and inputting/outputting relevant data from/to users and exchanging it between the participating nodes. Since terminals often come with physical and logistic constraints related to computational capacity, size of the data storage, available energy, physical dimensions and footprint of the device, security, accessibility and maintenance, part of the computations is typically sent out/offloaded to be handled in the network either as computing services or user controlled remote computation. Besides utilizing networked computing hardware, movement of computation for this purpose also introduces a need to utilize communication links to guarantee that the right input, output and intermediate data is in the right place at the right time. As a result, many kinds of communication related aspects, such as latency, bandwidth, congestion, routing, protocols, and reliability, will also have effect on the usage of intelligence. While there are basic solutions for offloading computation into the network and balancing the workload among computing nodes at the edge and cloud levels, meeting the ambitious goals of DEDICAT 6G requires systematic research, development and experimentation on these techniques.

The network should ultimately provide sufficient computational capacity to serve the needs of the users. By summing up the potentially simultaneous need for computing power of a high number of individual users, it is clear that techniques of parallel and distributed computing need to be widely applied to be able to serve users with a sufficient quality of service. The main idea of parallel computation is to decompose or divide the computational problem at hand into subproblems that can be solved in parallel and to compose the solution of the original problem from the results of the subproblems. This may naturally happen hierarchically, recursively and/or in consecutive parts. Solving the subproblems in parallel introduces the need for communication between the parallel parts, which in turn may create dependencies that require synchronization between them. Finally, to get actual results, these parallel parts need to be executed in physical processors which raises a need to define the relationship of execution units and parts executed in parallel, known as mapping. A related concept,

partitioning, refers to the way how data are placed with respect to the execution units. Distributed computation is similar to parallel computation but now subproblems are executed in physically distributed machines. This introduces substantially higher latencies, lower bandwidth and heterogeneity of HW and SW affecting the performance and productivity of software development. As a result, the methodology of distributed computation is somewhat different than that for parallel computation.

The main methods to increase the performance of the networked systems, such as DEDICAT 6G, is the increase the computing and communication resources of the system and apply aforementioned parallel and distributed computing and communication to decrease the latencies, optimize the architecture of the system to better utilize the existing resource, improve the efficiency of the components, e.g., processors, memories, I/O systems, communication links and switches, improve the algorithms supporting execution of applications and finally, optimizing the user applications. The main approach for the work associated to this section is to optimize the architecture via selected architectural techniques.

In the following subsections, 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 are considered with the first results. The goal is to support optimization of the DEDICAT 6G system, i.e., placement of intelligence, with respect to DEDICAT 6G Key Performance Indicators. Evaluating the possibilities and advantages of selected architectural techniques at processor, server, cloudlet and region levels is planned as a part of DEDICAT 6G studies.

5.1 Hierarchical architectures for distributed computing

Before entering into architectural techniques, let us first take another short look at the architectures for distributed computing.

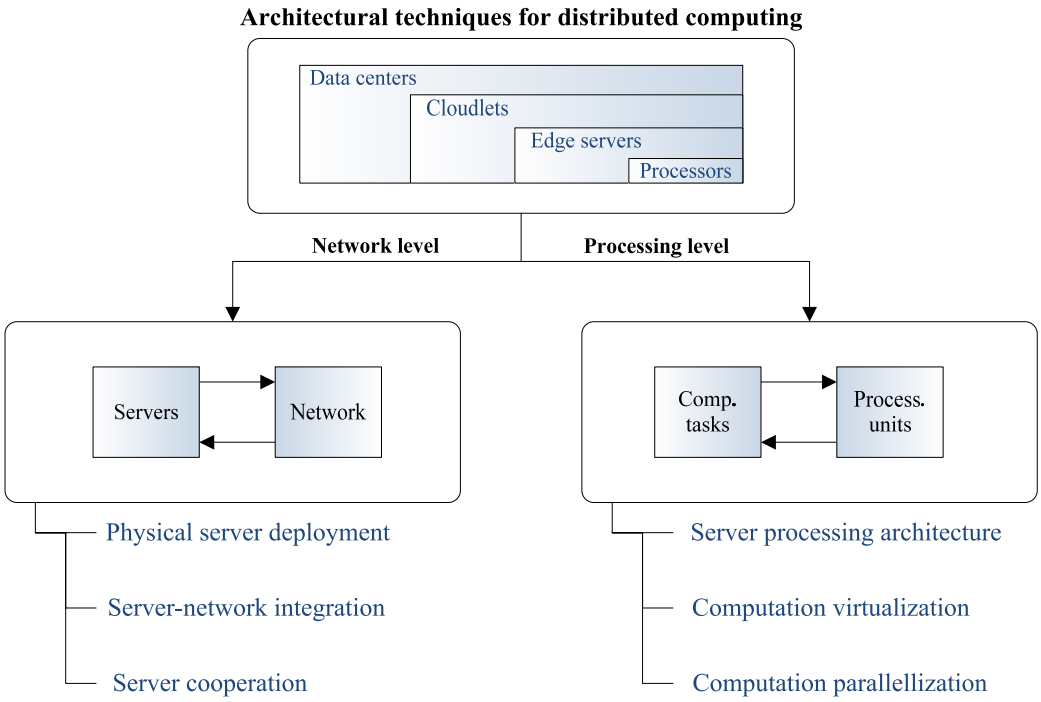


Figure 12: Architectural techniques for distributed computing at different levels

The main enabler for the intelligent distributed computing is ability to dynamically decide where the computing should take place for each user requesting service from the network, as described in the previous subsections. The desired features of good placement include

avoiding computational and communication imbalances of servers and access points from both overuse and underuse of resources. Underuse of resources, e.g., bandwidth, CPU time, of the network should be avoided in some cases including heavy network congestion or when maximizing the use of locally harvested energy. In addition, underuse can sometimes be seen an indication of wasting resources and therefore causing extra cost. Prior to this online computing placement decision, however, one has to make a number of architectural choices that then introduce constraints for the overall decision-making process for the computing placement on the physical server architecture and embedded processing units. We denote these methodologies as architectural techniques.

A number of hierarchical levels can be identified on which different architectural techniques can take place, cf. [22], [24], [25], [26], [27], [28], [29], [21], and [23]. Here we coarsely divide the techniques into network-level and processing-level architectural techniques, see Figure 12.

Regarding the network level of architectural techniques, the main objectives involve physical server deployment geographically, logical architecture with integration structures of servers into core network, and remote server cooperation.

The physical server architecture from a network deployment perspective is typically a hierarchical system where servers with different capabilities are placed in data centers, cloudlets (server clusters), and as single edge servers. The server deployment problem is typically more involved than traditional base station deployment problem. The number, type, cost, and location of servers in region of interest form a complicated architectural design problem involving several practical constraints. In general, edge servers can be co-located with gateways and access points or embedded in radio devices. The more hierarchical levels, the better is the scalability and responsiveness of the services. However, these benefits typically come with higher complexity and cost. The logical architecture then deals with the supported functionality on the top of the physical architecture. The main issues in the logical architecture include enabling physical/virtual resource orchestration via decision making and hiding the unnecessary system features via sophisticated abstraction methods. The number of hierarchical physical and logical layers may vary with different trade-offs (cf. [21]).

The integration of servers into the network is a critical enabler of geographically distributed computing. In essence, the network standards should seamlessly support the access to distributed servers. The logical network architecture can be divided into control and data planes where the control plane is in charge of distributing control strategy in centralized and/or decentralized manner. A number of different concepts have been proposed to integrate computing capabilities into wireless network (cf. [24]). The maximum distance between the parts of the distributed computing depends on the target use case. For latency critical applications, it is naturally expected to be more limited. The proposed frameworks differentiate according to the selected control centralization degree, controller position (e.g. base station), and server location (small cell, base station, gateway).

The server cooperation is an important characteristic of a sophisticated network-level architecture. The overall architecture should efficiently support the load balancing and dynamic service migration, as users move across different locations in the network, including the runtime application state for service continuity. In other words, the purpose of cooperation is to avoid both resource underutilization and overutilization while ensuring sufficient service quality. The competitive architecture must therefore support the message exchange towards such cooperation possibly deployed by multiple vendors.

Regarding the processing level, the main objectives include the definition of server processing architecture, computation virtualization structure, and computation parallelization structures.

The edge server typically involves a hosting platform for the hardware components and an application platform for virtualization activities. The processing hardware architecture of smaller edge servers is typically based on multicore CPU using X86/ARM architectures based on CISC/RISC approaches. Emerging specialized computing accelerations include graphical processing units (GPUs) and tensor processing units (TPUs) for graphical processing and machine learning, respectively. A modular edge server architecture then effectively combines different kind of core types, power distribution subsystems, cooling subsystems, memory, management, and I/O buses. A rough comparison of different processing units can be found from [30].

Computation virtualization refers to abstraction of above processor hardware architectures via software architectures such as virtual machines and containers. Virtualization frees an application to keep track on which physical processor it is running and enables easier rescheduling and parallelization. Edge computing orchestrators manage specific software platforms such as virtual machines or containers and related resources. E.g. FocusStack extends common OpenStack approach and coordinates edge resources for mobile nodes including cars and drones with location awareness. The container architectures can typically use resources more efficiently than virtual machines. Some examples include for edge scenarios KubeEdge, EdgeBench, DeFog, and Edgedroid and their benchmarking performance can be found from [23].

Computation parallelization refers to utilization of multiple and possibly heterogeneous processing unit architectures. Different processor unit architectures are dominated by the number of cores, clock frequencies, and cache memory sizes. A typical heterogeneous collaborative architecture involves GPUs and CPUs that can handle both throughput-critical and latency-critical applications for general purpose computation. The convergence properties of computation cores and storage elements are of high interest to use the resources efficiently. The expected performance gains from parallelization are strongly application dependent as the application task needs to be easily parallelized within one physical server. The multiprocessor architectures need specific attention to task scheduling, synchronization, load balancing.

In the following subsections, we look at some of these aspects in more detail.

5.2 Techniques

Practical improvements in placement of intelligence can be obtained by applying architectural techniques for improving the behaviour of the system with respect to KPIs in context switching, patterns of computation and communication, load balancing, movement of threads, reducing the state of computation, synchronization, programmability and placement of data and functionality:

Context switching—Traditional context switching in a CPU relies on interrupting execution of the current thread, waiting until all the operations under execution are completed, saving its registers to the thread table—a data structure keeping the state of the threads while they are not in execution—selecting a new thread for execution, loading the appropriate registers from the thread table and restarting the execution. If the context switching involves changing of the process too, writing to/reading from the process table, flushing resources keeping critical data and setting up memory spaces and privileges are needed.

While all CPUs capable of executing multitasking operating systems support switching of threads, the latency of thread switching is typically a few hundred clock cycles. This is not a problem when executing independent threads. However, if the threads of the functionality at hands require dense intercommunication, the performance can be catastrophically poor as will be illustrated by our experimental results in the next subsection.

The main mechanisms for accelerating context switching include multithreading and *thick control flow* (TCF) execution:

• **Multithreading** is an architectural technique that speeds up thread switching with a help of dedicated hardware and an on-chip thread storage. As long as the target thread is located in the on-chip thread storage, the thread switching time can be modest or even fully eliminated in certain CPU architectures.

• **TCF execution** is an abstraction of parallel computation that merges self-similar threads into a single computational entity that is independent of the number of threads. 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 as *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.

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 and spreading and permutation. These are used, e.g., in parallel processing and communication, collection of data, multicasting as well as in certain mapping tasks (see Figure 13).

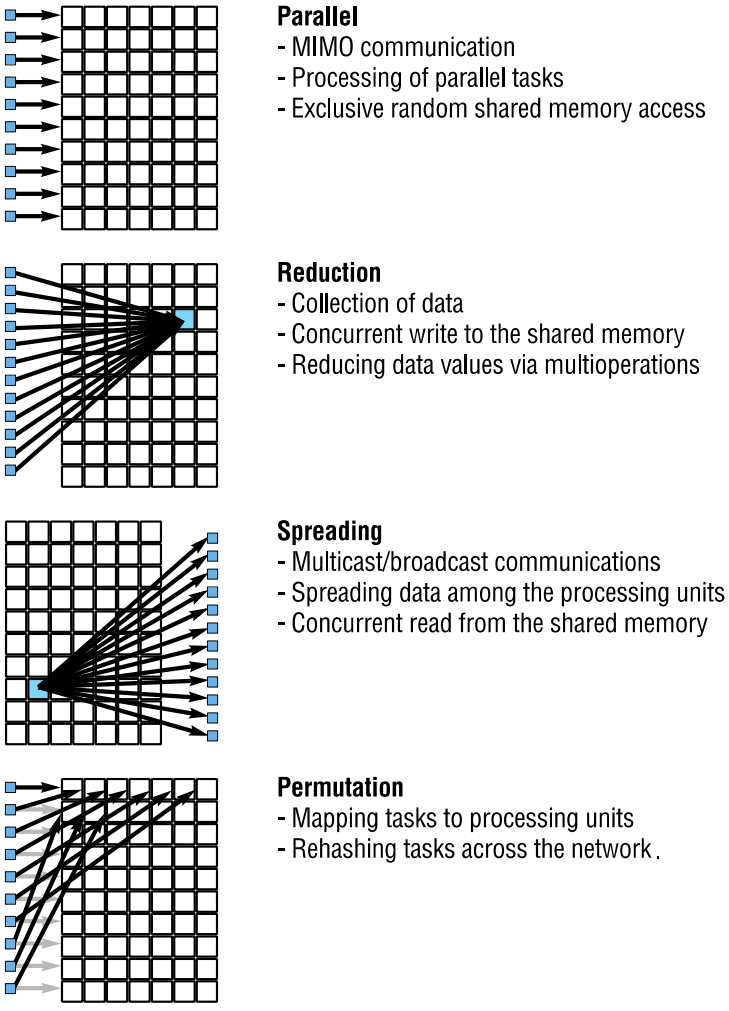


Figure 13: Patterns of computation and communication

There exist a few techniques to speed up these patterns:

• **Multioperations** are primitives of parallel computation by which threads perform reductions, e.g., additions, on values provided by multiple threads into a single value in a constant number of steps.

• **Broadcasting/multicasting** are primitives of communication by which a node sends a message/stream/transmission to multiple target nodes to reduce the load of the network.

• **Flexible mapping** is a technique to change the mapping of computational threads onto processing units. In certain situations, this can be used to implement permutation-style placement of data for free.

From the theoretical point of view, efficient hardware support of patterns can speed up execution by a logarithmic factor with respect to the baseline execution with a software solution. In our preliminary tests we have indication of this behaviour as we achieved the speedup with respect to a baseline system without these techniques.

Load balancing—Load imbalance is one of the most important reasons for poor utilization of the computational hardware. In the worst case, the execution time of a set of tasks in an *S* server region would be *S* times slower than perfectly balanced execution where all servers run with the full utilization.

The main means for load balancing include:

• **Work sharing** is a technique in which a computing node, e.g., an edge server, shares its workload with neighbouring edge servers or the servers within its edge server region to shorten the overall execution time or to reduce the load of the server. This may be applied to independent tasks or tasks with some dependencies.

• **Work stealing** is a technique in which a computing node, e.g., an edge server, looks at the queues of the neighbouring edge servers or servers within the same edge server region and "steals" their work items to reduce contention.

• **Moving threads** is an architectural technique in which threads are moved to a node where the data they are going to use is located rather than data is moved to threads making use of them. While this requires only one movement per thread whereas reading data from memory requires two movements per operation, the thread typically contains much more data than a memory reference.

Movement of threads—An essential part of computation offloading, functionality migration and load balancing is the movement of actual computational threads and processes. These include the state of the computation in processor (context) and memory area containing data and executable as well as needed libraries. The baseline technique is to move everything in the computational node to another. The overall latency associated to movement of a thread, set of threads or a process includes the amount of data that needs to be transported, time to move the functionality from one computer to another, downtime needed before a program can be restarted in the target node. These are highly dependent on the part of the network (nodes, routers/switches and communication links) involved in computing and transferring the functionality.

More advanced techniques for movement of threads include:

• **Containers** are a technique for moving functionality from a computer to another. A container consists of a runtime environment: an application, all its dependencies, libraries and other binaries, and configuration files needed to run it, bundled into one package.

• **Moving threads** is an architectural technique in which threads are moved to a node where the data they are going to use is located rather than data is moved to threads making use of them. While this requires only one movement per thread whereas reading data from

memory requires two movements per operation, the thread typically contains much more data than a memory reference.

Reducing the state of computation—The state of computation at processor level is directly proportional to the latency of moving/migrating computation in the network. The smaller the state is the faster the movement gets.

The most popular model of Flynn's Taxonomy of parallel execution is the *Multiple Instruction Multiple Data* (MIMD) model. In this model, multiple threads are executed in multiple processor cores in parallel. The main problems of the MIMD execution are that the state of computation is fully replicated for each thread of execution and that providing a (unique) program for each thread can be tedious if the number of threads is high. There exist, however, alternatives to the MIMD model, but they also come with limitations. The most interesting ones include Single Instruction Multiple Data (SIMD) and Thick Control Flow (TCF) execution:

• **SIMD execution** is an architectural technique in which parallel data is processed in the control of single instruction. This can potentially reduce the state of computation since there are only single instruction active at the time. The cost of SIMD execution is, however, that control parallel patterns must be executed sequentially. This becomes a problem with functionalities containing control parallelism. In addition, functionalities featuring heterogeneity need to be executed partially one after another.

• **TCF execution** is an abstraction of parallel computation that merges self-similar threads into a single computational entity that is independent of the number of threads. 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.

Synchronization—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 supposed to be able to execute fine-grained parallel algorithms efficiently.

The low-level mechanisms to support synchronization include:

• **Barrier synchronization** is a technique to prevent any thread from proceeding beyond a certain point of the program as long as not all threads have reached this point. After all the threads have reached the barrier, the threads are released for execution.

• **Wave synchronization** is a technique in which special synchronization messages are sent by the processors to the memory modules and vice versa. The idea is that when a processor has sent all its messages belonging to a single step on their way, it sends a synchronization message. Synchronization messages from various sources push on the actual messages, and spread to all possible paths, where the actual messages could go. When a switch receives a synchronization message from one of its inputs, it waits, until it has received a synchronization message from all inputs, then it forwards the synchronization wave to all outputs. The synchronization wave may not bypass any actual messages and vice versa. When a synchronization wave sweeps over a network, all switches, modules and processors receive exactly one synchronization message via each input link and send exactly one via each output link.

Programmability—A processor can be said to have good programmability if the functionalities can be expressed compactly and naturally without unnecessary architecture-dependent constructs. An important factor of programmability is also portability and ability to retain speedup with respect to the number of execution units among a group of processors using the same paradigm/approach but having different hardware implementation parameters. 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 additional challenges to programmability since the latencies are much higher, throughputs much lower than those in clusters or parallel machines. Programmability is important performance indicator since it is directly proportional to productivity of software development, and thus cost of the software.

A known method to address this challenge is to use Emulated Shared Memory (ESM) architecture.

• **ESM architecture** can be used to implement synchronous machines that are not sensitive to used computational patterns. These will make programming substantially simpler than that in current multicore processors and distributed systems.

Placement of data and functionality— 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. The main software techniques to reduce the performance penalties are matching the software parallelism with the hardware one and the blocking technique:

• **Matching parallelism** is a programming technique in which the number of active threads T_{sw} is limited to the number of hardware threads T_{hw} and cases with more parallelism than T_{hw} are handled by processing at most T_{hw} data element at the time with T_{hw} threads and repeating this until all the data elements have been processed. The challenges of this technique include the loop overhead, need for temporary variables in the case of self-modifying dataset and increased number of code lines.

• **Blocking** is a programming technique in which the functionality is divided into blocks that are executed in parallel. Since the current multicore CPUs and more distributed machines suffer from weak performance in the case of functionalities containing frequent exchange of data and/or non-trivial computational patterns. The challenges of this technique include the loop overhead, existence of suitable locality maximization technique, need for temporary variables in the case of self-modifying dataset and increased number of code lines.

5.3 Experimental results

In order to evaluate the goodness of the techniques listed in the previous subsection, we have made a number of early experimentations at processor/server level.

Context switching—We measured the execution time of the blocked version of the *memcopy* kernel program as a function of the number of threads in systems featuring 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.

According to our measurements, the execution time decreases almost linearly with respect to the number of threads as far as the number of threads does not exceed the number of processor cores (see Figure 14).

The measurements confirm that as far as the number of threads does not exceed the number of hardware threads, the performance stays roughly the same as with only one thread per core. However, as the number of threads exceeds the number of hardware threads, the execution time explodes by two to three orders of magnitude and keeps growing when the number of threads is further increased. The main reason for this is the interference of inefficient thread switching and synchronization amplified by operating system scheduler that tries to minimize the cost of thread switches by allocating long time slices for them.

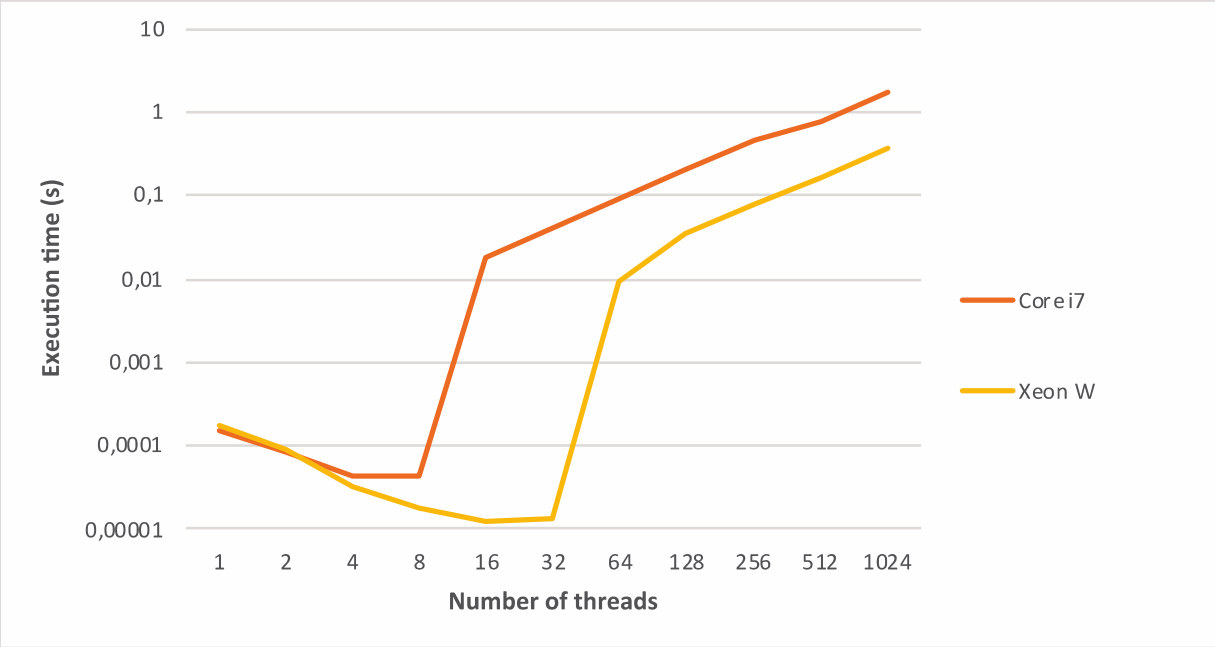


Figure 14: Execution time of blocked version of the memcopy benchmark (log scale)

Patterns of computation and communication—We measured the execution time of memory-to-memory reduction patterns in a machine supporting five variants of multioperations and compared the results to sequential algorithm without any of these operations. The variants include 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).

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 15).

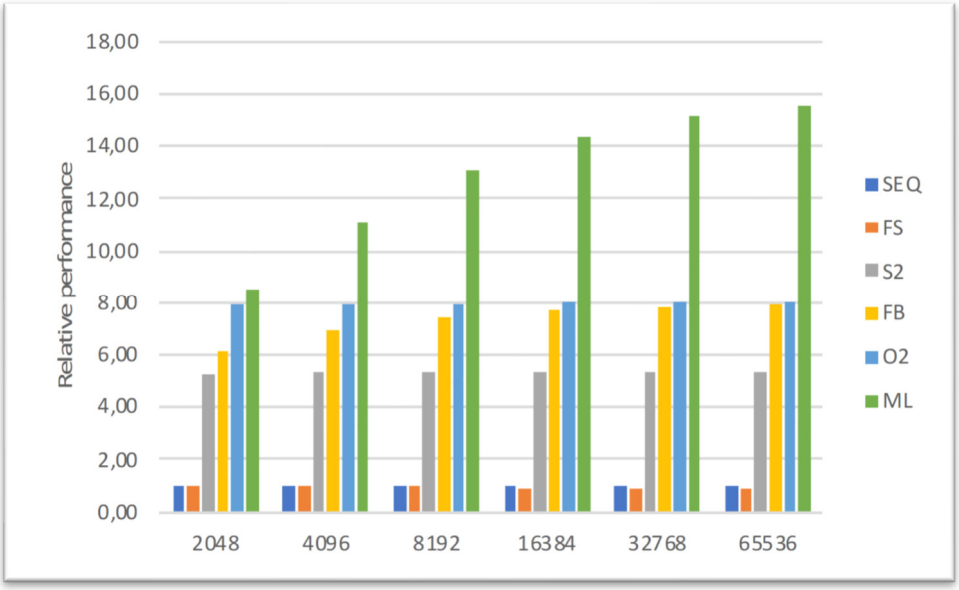


Figure 15: Relative performance of a multioperation reduction as a funct. of the input data array size

Programmability—In order to measure the complexity of programming, we implemented three versions of matrix addition algorithm A:=A+B for a typical Intel multicore CPU system with C/pthreads and a single version for a system utilizing shared memory emulation (ESM) 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 millions of times slower execution time than the 16-core ESM machine even if 18-core Xeon W system is used. The matched parallel pthreads version is also performance limited giving almost 100 times slower performance than ESM. Finally, the blocked version is almost able to match the performance of ESM, with its 44% (62% per processor core) slower execution speed.

Figure 16 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.

```
// madd Intel straight-forward pthreads
1 A[id]+=B[id];
2 Synchronize;
```

```
// madd Intel matched parallel pthreads
1 for (id=tid; id<N; id+=NUM_THREADS)
2   A[id]+=B[id];
3 Synchronize;
```

```
// madd Intel blocked pthreads
1 blocksize=SIZE/NUM_THREADS;
2 start = tid*blocksize;
3 stop = start + blocksize;
4 for (id=start; id<stop; id+=gap)
5   A[id]+=B[id];
6 Synchronize;
```

```
// madd ESM version
1 A_[$]+=B_[$];
```

Figure 16: Parallel matrix addition algorithm for Intel CPUs and an ESM machine

Placement of functionality—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.

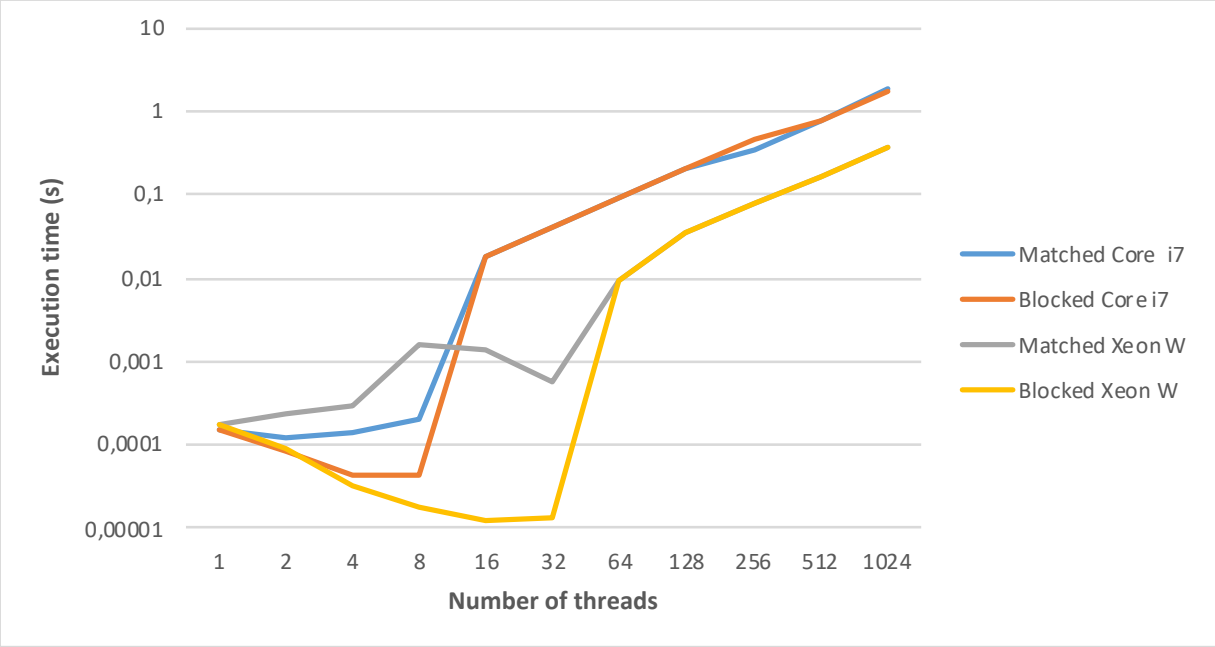


Figure 17: Execution time of matched parallel and blocked versions of the memcopy benchmark (log scale)

Figure 17 shows the results of the measurements. The matched parallel version of the benchmark utilizes interleaved access pattern whereas the blocked version relies on blocked access. The effect of switching from blocked to matched parallel version is dramatic in both processors: There is hardly any speedup as the number of threads is increased and the overall performance is much weaker.

6 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. In order to achieve these objectives, the work in WP3 is divided into three tasks: task T3.1 deals with the architectural techniques for supporting distribution of intelligence, task T3.2 deals with the algorithms and optimization of intelligence placement, and task T3.3 is about building a prototype.

This document has presented the elements in the DEDICAT 6G architecture that are relevant to WP3, namely the Context-Awareness, Analytics, Service Orchestration and Decision-Making Functional Groups. It has also given a short overview of the State-of-Art about the distribution of intelligence and an overview of the most mature technologies. Finally, it has described the preliminary work achieved in tasks 3.1 and 3.2.

In the next work period, the algorithms described in Section 4 will be refined and implemented in a Proof-of-Concept prototype. In addition, the work on architectural techniques described in section 5 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] Kamal MA, Raza HW, Alam MM, Su'ud MM, Sajak AbAB. "Resource Allocation Schemes for 5G Network: A Systematic Review". Sensors. 2021; 21(19):6588. https://doi.org/10.3390/s21196588.

[2] B. Dudin, N. A. Ali, A. Radwan and A. -E. M. Taha, "Resource Allocation with Automated QoE Assessment in 5G/B5G Wireless Systems," in *IEEE Network*, vol. 33, no. 4, pp. 76-81, July/August 2019, doi: 10.1109/MNET.2019.1800463.

[3] K. Lin, Y. Li, Q. Zhang and G. Fortino, "AI-Driven Collaborative Resource Allocation for Task Execution in 6G-Enabled Massive IoT," in *IEEE Internet of Things Journal*, vol. 8, no. 7, pp. 5264-5273, 1 April1, 2021, doi: 10.1109/JIOT.2021.3051031.

[4] S. Abidrabbu, H. Arslan, Energy-efficient resource allocation for 5G cognitive radio NOMA using game theory, 2021, arXiv preprint arXiv: 2101.00225.

[5] Nasim Kazemifard, Vahid Shah-Mansouri, Minimum delay function placement and resource allocation for Open RAN (O-RAN) 5G networks, Computer Networks, Volume 188, 2021, 107809, ISSN 1389-1286, https://doi.org/10.1016/j.comnet.2021.107809.

[6] A. Mukherjee and A. Hottinen, "Energy-efficient device-to-device MIMO underlay network with interference constraints," in *Proc. IEEE WSA'12*, Dresden, Germany, Mar. 2012, pp. 105–109.

[7] S. Wen, X. Zhu, Z. Lin, X. Zhang, and D. Yang, "Energy efficient power allocation schemes for device-to-device (D2D) communication," in *Proc. IEEE VTC Fall'13*, Las Vegas, USA, Sep. 2013, pp. 1–5.

[8] S. YAMASHITA, H. SUGANUMA, T. MURAKAMI, Y. TAKATORI and F. MAEHARA, "Centralized Resource Allocation Method for Multiuser MIMO Capable Wireless LANs," *2019 22nd International Symposium on Wireless Personal Multimedia Communications (WPMC)*, 2019, pp. 1-5, doi: 10.1109/WPMC48795.2019.9096141.

[9] H. Beshley, M. Beshley, T. Maksymyuk and I. Strykhalyuk, "Method of centralized resource allocation in virtualized small cells network with IoT overlay," *2018 14th International Conference on Advanced Trends in Radioelecrtronics, Telecommunications and Computer Engineering (TCSET)*, 2018, pp. 1147-1151, doi: 10.1109/TCSET.2018.8336397.

[10] D. Wu, J. Wang, R. Q. Hu, Y. Cai *et al.*, "Energy-efficient resource sharing for mobile device-to-device multimedia communications," *IEEE Trans. Veh. Tech.*, vol. 63, no. 5, pp. 2093–2103, Mar. 2014.

[11] A. Huang, Y. Li, Y. Xiao, X. Ge, S. Sun and H. Chao, "Distributed Resource Allocation for Network Slicing of Bandwidth and Computational Resource," *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, 2020, pp. 1-6, doi: 10.1109/ICC40277.2020.9149296.

[12] H. Dun, F. Ye, S. Jiao, Y. Li and T. Jiang, "The Distributed Resource Allocation for D2D Communication with Game Theory," *2019 IEEE-APS Topical Conference on Antennas and Propagation in Wireless Communications (APWC)*, 2019, pp. 104-108, doi: 10.1109/APWC.2019.8870437.

[13] DEDICAT 6G Deliverable D2.2 "Initial System Architecture".

[14] M. Lauridsen et al., "From LTE to 5G for connected mobility," IEEE Communication Magazine, 2017.

[15] E. Björnson et al., "How energy-efficient can a wireless communication system become ?" in Proc. 52nd Asilomar Conference on Sig. Sys, and Comp., 2018.

[16] ETSI, "Mission Critical Push to Talk (MCPTT)", Stage 1 (3GPP TS 22.179 version 16.5.0 Release 16)

[17] ITU, "Minimum requirements related to technical performance for IMT-2020 radio interface(s)", report M.2410-0 (11/2017).

[18] DEDICAT 6G Deliverable D4.1, "First release of mechanisms for dynamic coverage and

connectivity extension"
[19] ETSI, "Network Functions Virtualisation (NFV) Release 3; Protocols and Data Models; RESTful protocols specification for the Os-Ma-nfvo Reference Point", 09/2020.
[20] ETSI, "Network Functions Virtualisation (NFV) Release 3; Protocols and Data Models; NFV descriptors based on YANG Specification", 08/2020.
[21] C. Li et al., "Edge-oriented computing paradigms: a survey on architecture design and system management," 2017.
[22] Y. Mao et al., "A survey on mobile edge computing: The communication perspective," 2017.
[23] B. Varghese et al., "A survey on edge performance benchmarking," ACM Computing Surveys, 2021.
[24] P. Mach et al., "Mobile edge computing: A survey on architecture and computation offloading," IEEE Commun. Surveys & Tut., 2017.
[25] H. Liu et al., "Mobile edge cloud system: Architectures, challenges, and approaches," IEEE Systems Journal, 2018.
[26] P. Habibi et al., "Fog computing: A comprehensive architectural survey," IEEE Access, 2020.
[27] M Habibi et al., "A comprehensive survey of RAN architectures toward 5G mobile communication system," IEEE Access, 2019.
[28] G. Blake et al., "A survey of multicore processors," IEEE Signal Processing Magazine, 2009.
[29] S. Mittal et al., "A survey of CPU-GPU heterogeneous computing techniques," ACM Computing Surveys, 2015.
[30] C. Byers, "Heterogeneous computing in the Edge, " IIC Journal of Innovation, 2021.
[31] Radulescu, R. e. (2020). "Multi-objective multi-agent decision making: a utility-based analysis and survey. Autonomous Agents and Multi-Agent Systems", (pp. 1-52).
[32] Abels, A. e. (2019). "Dynamic weights in multi-objective deep reinforcement learning. International conference on machine learning".
[33] 5G-Ensure. (2017). Deliverable D3.6 5G-PPP security enablers open specification.
[34] 5GPP. (2017). White Paper Phase 1 Security Landscape.
[35] Ahilan, S. e. (2019). Feudal multi-agent hierarchies for cooperative reinforcement learning. ICLR.
[36] B. Sas, K. S. (2014). Classifying Users based on their Mobility Behavior in LTE networks. 10th Int. Conf. on Wireless and Mob. Comms. (ICWMC).
[37] Chu, T. e. (2019). Multi-agent deep reinforcement learning for large-scale traffic signal control. IEEE Transactions on Intelligent Transportation Systems.
[38] Cisco. (2014). Quality of Service Design Overview. Dans Cisco Press book End-to-End QoS Network Design: Quality of Service for Rich-Media & Cloud Networks, 2nd Edition .
[39] Deb, K. e. (37750). A Fast and Elitist Multi-Objective Genetic Algorithm: NSGA-II.
[40] Hsu, M. L.-J. (2013). Mining GPS data for mobility patterns: A survey. Pervasive and Mobile Computing.
[41] Marjalaakso, M. (s.d.). Security Requirements and Constraints of VoIP. Helsinki University of Technology.
[42] METIS. (s.d.). FP7-ICT-317669-METIS/D1.1 .
[43] NGMN. (s.d.). 5G White Paper.
[44] Qualcomm. (2016). Making Immersive Virtual Reality Possible in Mobile.
[45] Qualcomm Technologies, I. (2017). VR and AR pushing connectivity limits.
[46] S. Rangan, T. S. (2014). Millimeter-Wave Cellular Wireless Networks: Potentials and Challenges. IEEE, 102(3), 366-385.
[47] Smith, R. (1980). The Contract Net Protocol: High level communication and control in a distributed environment. IEEE Transactions on computers.

[48] Troch, A. e. (2021). Transfer Learning in Automomous Driving using Real-World Samples. Advances in P2P, Parallel, Grid, Cloud and Internet Computing.

[49] Zimmer, M. e. (2021). Learning Fair POlicies in Decentralized Cooperative Multi-Agent Reinforcement Learning. Proceedings of the 38th international conference on machine learning.