Programmable edge-to-cloud virtualisation fabric for the 5G Media industry

D3.4: 5G-MEDIA Operations and Configuration Platform

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Lead partner:</td>
<td>UCL</td>
</tr>
<tr>
<td>Delivery date (DoA):</td>
<td>30/11/19</td>
</tr>
<tr>
<td>Actual delivery date:</td>
<td>13/12/19</td>
</tr>
<tr>
<td>Dissemination level:</td>
<td>Public</td>
</tr>
<tr>
<td>Version number:</td>
<td>1.0 Status: Final</td>
</tr>
<tr>
<td>Grant Agreement N°:</td>
<td>761699</td>
</tr>
<tr>
<td>Project Acronym:</td>
<td>5G-MEDIA</td>
</tr>
<tr>
<td>Project Title:</td>
<td>Programmable edge-to-cloud virtualisation fabric for the 5G Media industry</td>
</tr>
<tr>
<td>Instrument:</td>
<td>IA</td>
</tr>
<tr>
<td>Call identifier:</td>
<td>H2020-ICT-2016-2</td>
</tr>
<tr>
<td>Topic:</td>
<td>ICT-08-2017, 5G PPP Convergent Technologies, Strand 2: Flexible network applications</td>
</tr>
<tr>
<td>Start date of the project:</td>
<td>1 June, 2017</td>
</tr>
<tr>
<td>Duration:</td>
<td>33 months</td>
</tr>
</tbody>
</table>

Project co-funded by the European Commission under the Horizon 2020 Programme.
### Revision History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Who</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interim v0.1</td>
<td>13/12/18</td>
<td>UCL/all</td>
<td>Outline of CNO/MAPE algorithms</td>
</tr>
<tr>
<td>Interim v0.2</td>
<td>15/01/19</td>
<td>UCL/all</td>
<td>Revised algorithm specification</td>
</tr>
<tr>
<td>Interim v1</td>
<td>04/03/19</td>
<td>UCL/all</td>
<td>Frozen specification of algorithms</td>
</tr>
<tr>
<td>ToC</td>
<td>13/06/19</td>
<td>UCL/all</td>
<td>ToC of full D3.4</td>
</tr>
<tr>
<td>V0.0</td>
<td>04/10/19</td>
<td>UCL/all</td>
<td>Initial version of sections 2 and 3</td>
</tr>
<tr>
<td>V0.1</td>
<td>20/11/19</td>
<td>UCL/all</td>
<td>First integrated version</td>
</tr>
<tr>
<td>V0.2</td>
<td>25/11/19</td>
<td>UCL/all</td>
<td>New sub-sections added</td>
</tr>
<tr>
<td>V0.3</td>
<td>26/11/19</td>
<td>UCL/all</td>
<td>1st complete version</td>
</tr>
<tr>
<td>V0.4</td>
<td>28/11/19</td>
<td>UCL/all</td>
<td>Full integrated version for internal review</td>
</tr>
<tr>
<td>V0.5</td>
<td>06/11/19</td>
<td>UCL/all</td>
<td>Updates following internal review</td>
</tr>
<tr>
<td>V1.0</td>
<td>13/12/19</td>
<td>UCL/ENG/all</td>
<td>Final version</td>
</tr>
</tbody>
</table>

### Quality Control

<table>
<thead>
<tr>
<th>Role</th>
<th>Date</th>
<th>Who</th>
<th>Approved/Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reviewer</td>
<td>02/12/19</td>
<td>Igor Fritsch [BIT]</td>
<td>Comments incorporated</td>
</tr>
<tr>
<td>Reviewer</td>
<td>03/12/19</td>
<td>Francesca Moscatelli [NXW]</td>
<td>Comments incorporated</td>
</tr>
<tr>
<td>Reviewer</td>
<td>03/12/19</td>
<td>Stamatia Rizou [SiLO]</td>
<td>Comments incorporated</td>
</tr>
<tr>
<td>Reviewer</td>
<td>13/12/19</td>
<td>Pasquale Andriani [ENG]</td>
<td>Approved</td>
</tr>
</tbody>
</table>
Disclaimer

This document may contain material that is copyright of certain 5G-MEDIA project beneficiaries, and may not be reproduced or copied without permission. The commercial use of any information contained in this document may require a license from the proprietor of that information. The 5G-MEDIA project is part of the European Community's Horizon 2020 Program for research and development and is as such funded by the European Commission. All information in this document is provided "as is" and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability. For the avoidance of all doubts, the European Commission has no liability with respect to this document, which is merely representing the authors’ view.

The 5G-MEDIA Consortium consists of the following organisations:

<table>
<thead>
<tr>
<th>Participant number</th>
<th>Participant organisation name</th>
<th>Short name</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>ENGINEERING – INGEGNERIA INFORMATICA SPA</td>
<td>ENG</td>
<td>Italy</td>
</tr>
<tr>
<td>02</td>
<td>IBM ISRAEL - SCIENCE AND TECHNOLOGY LTD</td>
<td>IBM</td>
<td>Israel</td>
</tr>
<tr>
<td>03</td>
<td>SINGULARLOGIC ANONYMI ETAIREIA PLIROFORIAKON SYSTIMATON KAI EFARMOGON PLIROFORIKIS</td>
<td>SILO</td>
<td>Greece</td>
</tr>
<tr>
<td>04</td>
<td>HELLENIC TELECOMMUNICATIONS ORGANIZATION S.A. - OTE AE (ORGANISMOS TLEPIKIOINONION TIS ELLADOS OTE AE)</td>
<td>OTE</td>
<td>Greece</td>
</tr>
<tr>
<td>05</td>
<td>CORPORACION DE RADIO Y TELEVISION ESPANOLA SA</td>
<td>RTVE</td>
<td>Spain</td>
</tr>
<tr>
<td>06</td>
<td>UNIVERSITY COLLEGE LONDON</td>
<td>UCL</td>
<td>United Kingdom</td>
</tr>
<tr>
<td>07</td>
<td>TELEFONICA INVESTIGACION Y DESARROLLO SA</td>
<td>TID</td>
<td>Spain</td>
</tr>
<tr>
<td>08</td>
<td>UNIVERSIDAD POLITECNICA DE MADRID</td>
<td>UPM</td>
<td>Spain</td>
</tr>
<tr>
<td>09</td>
<td>INSTITUT FUER RUNDFUNKTECHNIK GMB</td>
<td>IRT</td>
<td>Germany</td>
</tr>
<tr>
<td>10</td>
<td>NEXTWORKS</td>
<td>NXW</td>
<td>Italy</td>
</tr>
<tr>
<td>11</td>
<td>ETHNIKO KENTRO EREVNAS KAI TECHNOLOGIKIS ANAPTYXIS</td>
<td>CERTH</td>
<td>Greece</td>
</tr>
<tr>
<td>12</td>
<td>NETAS TELEKOMUNIKASYON ANONIM SIRKETI</td>
<td>NET</td>
<td>Turkey</td>
</tr>
<tr>
<td>13</td>
<td>INTERINNOV SAS</td>
<td>IINV</td>
<td>France</td>
</tr>
<tr>
<td>14</td>
<td>BITTUBES GMB</td>
<td>BIT</td>
<td>Germany</td>
</tr>
<tr>
<td>15</td>
<td>NATIONAL CENTER FOR SCIENTIFIC RESEARCH - DEMOKRITOS</td>
<td>NCSRD</td>
<td>Greece</td>
</tr>
</tbody>
</table>
Table of Contents

EXECUTIVE SUMMARY

1. INTRODUCTION AND SCOPE OF DELIVERABLE

2. MAPE ARCHITECTURE
   2.1. NFVI MONITORING
       2.1.1. OpenStack monitoring metrics
       2.1.2. Kubernetes (FaaS) Monitoring Metrics
       2.1.3. OpenNebula Monitoring Metrics
   2.2. VNFS MONITORING
   2.3. PUBLISH/SUBSCRIBE MECHANISM DASHBOARD
   2.4. MONITORING DASHBOARDS

3. COGNITIVE NETWORK OPTIMISER
   3.1. ROLE OF CNO OPTIMISATION ALGORITHMS IN 5G-MEDIA
       3.1.1. CNO for 5G-MEDIA platform (CNO\textsubscript{platform})
       3.1.2. CNO for individual use cases
   3.2. THE OVERALL OPTIMIZATION OF CNO\textsubscript{platform}
       3.2.1. VNF forwarding graph (VNFFG) structures
       3.2.2. Evaluation results
       3.2.3. Conclusion
   3.3. MEDIA SERVICE LEVEL OPTIMISATION
       3.3.1. Training methodology
       3.3.2. Offline vs online learning
   3.4. REINFORCEMENT LEARNING IN UC1: IMMERSIVE MEDIA
       3.4.1. State-space modelling and reward function
       3.4.2. Employed monitoring parameters
       3.4.3. Supported optimization actions
       3.4.4. Quality-of-Experience
       3.4.5. Initial training
       3.4.6. Algorithmic flow
       3.4.7. Experimenting with reward weights
   3.5. REINFORCEMENT LEARNING IN UC2: MOBILE CONTRIBUTION, REMOTE AND SMART PRODUCTION IN BROADCASTING
       3.5.1. Scenario 1: Optimise vCompression levels
       3.5.2. Scenario 2: Mobile contribution
       3.5.3. Scenario 3: Optimise computational resources allocated to cognitive services
3.6. **Optimisation in Use Case 3: Ultra-High Definition over Content Distribution Networks** ................. 73

3.6.1. **Scenario 1: Triggering of scale-out and scale-in of vCaches** ........................................... 73

3.6.2. **Scenario 2: Dynamic instantiation of serverless vTranscoders** ........................................... 82

3.6.3. **Scenario 3: Joint optimisation of load balancing and video encodings at vCaches** ................. 84

4. **Serverless Framework within the 5G-Media SVP** ................................................................. 86

4.1. **Serverless-Based VNFs and ETSI MANO** ........................................................................... 86

4.2. **Generalization of the Problem** ......................................................................................... 89

4.3. **From Serverless VIM to Serverless VNFM and SO** .............................................................. 90

4.3.1. **Argo Project** ................................................................................................................. 90

4.3.2. **Serverless VNFM and SO** ............................................................................................ 93

5. **Conclusions** .......................................................................................................................... 95

6. **References** ............................................................................................................................ 96

List of Figures

Figure 1 - MAPE software architecture ................................................................. 15
Figure 2 - Indicative list of topics in the Kafka Manager UI ........................................ 24
Figure 3 - List of consumer groups in Kafka Manager UI ........................................... 25
Figure 4 - Visualization of a running VNF (single VDU) in Kubernetes ...................... 26
Figure 5 - Visualization of a running vTranscoder VNF, part of the UC1 service, deployed in Kubernetes NFVI ........................................................................... 26
Figure 6 - Visualization of a running vCompressionEngine VNF, part of the UC2 service, deployed in OpenNebula NFVI ........................................................................... 26
Figure 7 - Visualization of a running vCache VNF, part of the UC3 service, deployed in OpenStack NFVI ........................................................................... 27
Figure 8 - Role of CNO in 5G-MEDIA ........................................................................ 28
Figure 9 - Pareto front of cost and QoE ....................................................................... 30
Figure 10 - QoE/cost list ......................................................................................... 31
Figure 11 – VNFFG Linear structure ......................................................................... 32
Figure 12 – VNFFG Tree structure ............................................................................... 33
Figure 13 – VNFFG Loop structure ............................................................................... 33
Figure 14 - Optimisation of routing and placement for VNFFGs ................................. 33
Figure 15 - VNFFG placement with computational resource constraint ...................... 34
Figure 16 - Example of VNFFG representation: (a) linear (b) tree and (c) loop structures .................................................................................................................. 36
Figure 17 - Penalty function p used to normalise the latency and cost metrics ............. 37
Figure 18 - Evolutionary algorithm flow chart ............................................................ 40
Figure 19 - Geant network topology ......................................................................... 42
Figure 20 - Convergence of SPO and CNO_platform: (a) linear (b) tree and (c) loop structures .................................................................................................................. 43
Figure 21 - CNO_platform vs. SPO .............................................................................. 44
Figure 22 - CNO_platform with different delay configuration L1 - L4 ......................... 45
Figure 23 - CNO_platform algorithm with different weight configurations ............... 45
Figure 24 - Parallel training using A3C algorithm ...................................................... 47
Figure 25 - Asynchronous Advantage Actor-Critic (A3C) .......................................... 48
Figure 26 - Mean Opinion Score with respect to packet loss rate (ITU-T Recommendation P.1070) .................................................................................................................. 57
Figure 27 - Box plot of Mean Opinion Score for combinations of reward weights
Figure 28 - Box plot of Transcoding Cost for combinations of reward weights
Figure 29 - Box plot of QoE-to-Cost Rate for combinations of reward weights
Figure 30 - CNO dynamically changes the compression level of the vCompression VNF by switching between different vCompression VNFs. Each VNF has FFmpeg running with specific compression level. We consider four different compression levels: low, medium, high and premium.
Figure 31 - CNO dynamically changes the compression level of the vCompression VNF by choosing an appropriate FFmpeg instance via an internal load-balancer.
Figure 32 - CNO/MAPE instantiates a new vCompression VNF on the fly (see the box with dotted line) either via a FaaS technique or a scale-out operation supported by OSM. In case of the latter, a new vCompression VNF equipped with high compression scheme is instantiated on the fly. CNO/MAPE executes this operation via OSM. Once a new VNF becomes operational, CNO reroutes the MPE traffic to it via a load-balancer VNF (see the dotted arrows).
Figure 33 - CNO workflow
Figure 34 - CNO instructs the LB to move the journalist traffic to another already instantiated face/image detection VNF that may be located anywhere in the TID network. The digits on the arrows depicts the order of which the actions should be taken. Note that we can think of this scenario in a Multi-NFVI use case.
Figure 35 - First CNO instructs a FaaS controller to instantiate a new temporary face/image detection VNF. It then instructs the LB to reroute the journalist traffic to the new VNF. The digits on the arrows depicts the order of which the actions should be taken.
Figure 36 - CNO should switch to a more powerful/resourceful Speech-To-Text VNF via a load-balancer VNF.
Figure 37 - CNO attempts to dynamically change the maximum capacity of the computational resources (such CPU, memory, and storage) allocated to the Speech-To-Text VNF.
Figure 38 - vCDN NS topology
Figure 39 - A simple autoencoder neural network
Figure 40 - Loss per iteration
Figure 41 - Load balancer with ML
Figure 42 - vCDN service with integrated probing function
Figure 43 - Degradation of the users' link caused by traffic anomalies
Figure 44 - Spawning of the vTranscoder
Figure 45 - FaaS vTranscoder providing a new transcoded content at the edge
Figure 46 - Configuration of load balancer and vCaches ......................................................... 84
Figure 47 - Integer linear program (ILP) model ........................................................................... 85
Figure 48 - Use Case 1 Schematic Example .................................................................................. 87
Figure 49 - Extended Serverless NFV Stack ................................................................................ 89
Figure 50 - High Level Architecture of the Serverless NFVO ......................................................... 93
Figure 51 - Component Level Architecture of Serverless VNFM and SO Extensions in 5G-MEDIA Platform ................................................................................................................. 94
List of Tables

Table 1 - List of monitoring metrics collected from OpenStack VIM ................................................. 17
Table 2 - List of monitoring metrics collected from Kubernetes cluster ................................................. 19
Table 3 - List of monitoring metrics in OpenNebula VIM .................................................................... 20
Table 4 - List of provided metrics from the vTranscoder VNF (multiple qualities) ............................... 21
Table 5 - List of provided metrics from the vCompression Engine VNF ............................................... 21
Table 6 - List of provided metrics from the vCache VNF (per network interface) ............................... 22
Table 7 - List of provided metrics from the vCache VNF (statistics) ................................................... 22
Table 8 - List of metrics provided from the spectators in UC1 ............................................................. 23
Table 9 - List of metrics provided from the PNF media server (Plex) in UC3 ..................................... 23
Table 10 - Summary of key notations (alphabetical order) .................................................................. 35
Table 11 - The monitoring parameters employed by the CNO in UC1 ............................................... 54
Table 12 - Supported Optimization Actions in UC1 ............................................................................. 55
Table 13 – Mapping of QoE requirements to bandwidth allocation schemes ...................................... 69
Table 14 – Mapping of billing type to bandwidth allocation schemes ............................................... 69
Table 15 - CNO-utilized metrics and their origins .............................................................................. 78
Table 16 - Available CNO actions ....................................................................................................... 79
Table 17 - A simple example of an Argo flow ...................................................................................... 92
Definitions and acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANN</td>
<td>Artificial Neural Network</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>CAIDA</td>
<td>Center for Applied Internet Data Analysis</td>
</tr>
<tr>
<td>CDN</td>
<td>Content Delivery Network</td>
</tr>
<tr>
<td>CNO</td>
<td>Cognitive Network Optimiser</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>DC</td>
<td>Data Centre</td>
</tr>
<tr>
<td>ELK</td>
<td>Elastic Search, Logstash, Kibana</td>
</tr>
<tr>
<td>FaaS</td>
<td>Function as a Service</td>
</tr>
<tr>
<td>GA</td>
<td>Genetic Algorithm</td>
</tr>
<tr>
<td>GPU</td>
<td>Graphics Processing Unit</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hyper Text Transfer Protocol</td>
</tr>
<tr>
<td>ICMP</td>
<td>Internet Control Message Protocol</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>JDK</td>
<td>Java Development Kit</td>
</tr>
<tr>
<td>JMX</td>
<td>Java Management Extensions</td>
</tr>
<tr>
<td>JSON</td>
<td>Java Script Object Notation</td>
</tr>
<tr>
<td>ISP</td>
<td>Internet Service Provider</td>
</tr>
<tr>
<td>K8s</td>
<td>Kubernetes</td>
</tr>
<tr>
<td>LSTM</td>
<td>Long Short-Term Memory</td>
</tr>
<tr>
<td>LXC</td>
<td>Linux Containers</td>
</tr>
<tr>
<td>MANO</td>
<td>Management and Orchestration</td>
</tr>
<tr>
<td>MAPE</td>
<td>Monitor, Analyse, Plan, Execute</td>
</tr>
<tr>
<td>MILP</td>
<td>Mixed Integer Linear Program</td>
</tr>
<tr>
<td>ML</td>
<td>Machine Learning</td>
</tr>
<tr>
<td>NFVI</td>
<td>Network Functions Virtualisation Infrastructure</td>
</tr>
<tr>
<td>NN</td>
<td>Neural Network</td>
</tr>
<tr>
<td>NS</td>
<td>Network Service</td>
</tr>
<tr>
<td>NSD</td>
<td>Network Service Descriptor</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>OSM</td>
<td>Open Source MANO</td>
</tr>
<tr>
<td>QoE</td>
<td>Quality of Experience</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality of Service</td>
</tr>
<tr>
<td>ReLU</td>
<td>Rectified Linear Unit</td>
</tr>
<tr>
<td>REST</td>
<td>Representational State Transfer</td>
</tr>
<tr>
<td>RRN</td>
<td>Recurrent Neural Network</td>
</tr>
<tr>
<td>RRT</td>
<td>Round Trip Time</td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kit</td>
</tr>
<tr>
<td>SDN</td>
<td>Software Defined Network</td>
</tr>
<tr>
<td>SLA</td>
<td>Service Level Agreement</td>
</tr>
<tr>
<td>SVP</td>
<td>Service Virtualisation Platform</td>
</tr>
<tr>
<td>UC</td>
<td>Use Case</td>
</tr>
<tr>
<td>UHD</td>
<td>Ultra High Definition</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
<tr>
<td>UUID</td>
<td>Universal Unique Identifier</td>
</tr>
<tr>
<td>vCache</td>
<td>Virtual Cache</td>
</tr>
<tr>
<td>vCPU</td>
<td>Virtual CPU</td>
</tr>
<tr>
<td>VDU</td>
<td>Virtual Deployment Unit</td>
</tr>
<tr>
<td>VIM</td>
<td>Virtualised Infrastructure Manager</td>
</tr>
<tr>
<td>VM</td>
<td>Virtual Machine</td>
</tr>
<tr>
<td>VNF</td>
<td>Virtual Network Function</td>
</tr>
<tr>
<td>VNFFG</td>
<td>Virtual Network Function Forwarding Graph</td>
</tr>
</tbody>
</table>
Executive summary

This document accompanies the software prototype release of the 5G-MEDIA Operations and Configuration Platform. It focuses on the design of the platform and the associated algorithms for Quality of Service (QoS) Control and Management tools in the 5G-MEDIA project.

The basis of these tools is the Monitoring, Analysis, Planning and Execution (MAPE) component of the project’s Service Virtualisation Platform.

The main objectives of the 5G-MEDIA Media Service MAPE are:

- To collect and store metrics about the status of infrastructure resources and the performance and behaviour of media applications.
- To organise and harmonise the collected metrics under a common data model.
- To integrate machine learning and resource planning algorithms to optimise the media applications and the Network Functions Virtualisation Infrastructure resources.
- To implement deployment and scaling directives to Management and Orchestration (MANO) components to optimise resource management, network performance and enforce Quality of Service guarantees.

Monitoring collects data from the various infrastructure technologies and application domains, and assesses critical KPIs about their performance. It has been designed to collect data from the following environments in the 5G-MEDIA project testbeds:

- OpenStack-based cloud computing environments,
- Kubernetes management platforms,
- OpenNebula cloud computing platforms,
- OpenWhisk FaaS framework,
- Virtual execution environments based on unikernels,
- The application domain (e.g. by integrating Telegraf and Apache Traffic Server).

Pre-process & analysis services are concerned with converting raw monitored data into useful statistics on past performance and future predictions. The planning service executes algorithms to configure and adapt the deployment of the Network Services (NS) and the infrastructures upon which they run. The outputs of the optimisation algorithms are implemented by the execution service to configure the MANO components and the Virtual Network Functions (VNF) forming the NS.

The software prototypes implementing the 5G-MEDIA Operations and Configuration Platform are listed in Appendix A together with links to the repositories containing the source code, API specifications and user guides.
1. **Introduction and scope of deliverable**

This document accompanies the software prototype release of the 5G-MEDIA Operations and Configuration Platform. It focusses on the design of the platform and the associated algorithms for Quality of Service (QoS) Control and Management tools in the 5G-MEDIA project.

The final 5G-MEDIA software architecture is specified in detail in deliverable D2.4 and the full evaluation results of the optimisation algorithms are contained in deliverables D6.2, D6.3 and D6.4.

Chapter 2 presents the high-level design of the Monitoring, Analysis, Planning and Execution (MAPE) components of the project’s Service Virtualisation Platform, with a focus on the monitoring system, including the metrics being collected from the underlying infrastructure. Monitoring collects data from the various infrastructure technologies and application domains, and assesses critical KPIs about their performance. It has been designed to collect data from the following environments in the 5G-MEDIA project testbeds:

- OpenStack\(^1\)-based cloud computing environments,
- Kubernetes\(^2\) management platforms,
- OpenNebula\(^3\) cloud computing platforms,
- OpenWhisk\(^4\) FaaS framework,
- Virtual execution environments based on unikernels\(^5\),
- The application domain (e.g. by integrating Telegraf\(^6\) and Apache Traffic Server\(^7\)).

Chapter 3 presents the main intelligence behind the MAPE system: the Cognitive Network Optimiser (CNO). The CNO aims to manage the platform resources in an efficient and cost-effective manner with the aim of delivering the performance levels required to meet the Quality of Experience (QoE) objectives of the media applications being supported by the 5G-MEDIA platform. The CNO is a multi-level system where the highest level has the oversight of the entire platform resources, managing the allocation of resources to the different applications being considered in each of the use cases. It is at this level where competing demands between application types are handled and where appropriate trade-offs can be made. The lower-level CNO algorithms take the scope of an individual media service and work within the constraints set by the higher platform level CNO. The lower level algorithms manage the potential competition between the individual sessions of the application it is optimising.

---

1 [https://www.openstack.org/](https://www.openstack.org/)
2 [https://kubernetes.io/](https://kubernetes.io/)
3 [https://opennebula.org/](https://opennebula.org/)
4 [https://openwhisk.apache.org/](https://openwhisk.apache.org/)
5 [http://unikernel.org/](http://unikernel.org/)
6 [https://www.influxdata.com/time-series-platform/telegraf/](https://www.influxdata.com/time-series-platform/telegraf/)
7 [https://trafficserver.apache.org/](https://trafficserver.apache.org/)
Chapter 3 presents the design of the algorithms at the overall platform level and for each of the use cases, together with ideas for future expansion beyond the scope of the project lifetime.

Chapter 4 documents the serverless framework within the 5G-MEDIA SVP. It builds upon the design of the serverless VIM presented in deliverable D3.2 and presents the advances made when considering the practical application in the project’s use case scenarios. The chapter documents the project’s proposed enhancements to the ETSI NFV architecture to support the dynamic instantiation of serverless VNFs.

Finally, the software components forming the 5G-MEDIA SVP are available through the project’s public repositories as linked in Appendix A.
2. **MAPE architecture**

Figure 1 depicts the final version of the MAPE architecture and the integration among the involved services. The description of the architecture as a whole as well as the interfaces of the services is described in D2.4. In this section, the list of the collected monitoring metrics is depicted. Moreover, a brief description in the MAPE dashboards follows; it includes the publish/subscribe mechanism dashboard (Kafka Manager UI) and the Monitoring dashboard.

![MAPE software architecture](image)

**Figure 1 - MAPE software architecture**

### 2.1. **NFVI monitoring**

#### 2.1.1. *OpenStack monitoring metrics*

Through the Ceilometer publisher, VM-level metrics are collected and sent to the MAPE. By default, a large set of metrics are provided from the Ceilometer service. To reduce the traffic burden and the load of exchanged messages, the Ceilometer service in the 5G-MEDIA OpenStack VIMs has been configured to provide to the 5G-MEDIA Kafka bus the metrics that are listed in Table 1.

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Type</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>memory</td>
<td>Gauge</td>
<td>megabytes</td>
<td>Volume of RAM allocated to the instance</td>
</tr>
<tr>
<td>memory.usage</td>
<td>Gauge</td>
<td>megabytes</td>
<td>Volume of RAM used by the instance from the amount of its allocated memory</td>
</tr>
</tbody>
</table>

---

8 [https://kafka.apache.org/](https://kafka.apache.org/)
<table>
<thead>
<tr>
<th>Metric</th>
<th>Type</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>memory.resident</td>
<td>Gauge</td>
<td>megabytes</td>
<td>Volume of RAM used by the instance on the physical machine</td>
</tr>
<tr>
<td>cpu</td>
<td>Cumulative</td>
<td>nanoseconds</td>
<td>CPU time used</td>
</tr>
<tr>
<td>cpu_util</td>
<td>Gauge</td>
<td>%</td>
<td>Average CPU utilization</td>
</tr>
<tr>
<td>vcpus</td>
<td>Gauge</td>
<td>vCPU</td>
<td>Number of virtual CPUs allocated to the instance</td>
</tr>
<tr>
<td>disk.read.requests.rate</td>
<td>Gauge</td>
<td>request/second</td>
<td>Average rate of read requests</td>
</tr>
<tr>
<td>disk.write.requests.rate</td>
<td>Gauge</td>
<td>request/second</td>
<td>Average rate of write requests</td>
</tr>
<tr>
<td>disk.read.bytes.rate</td>
<td>Gauge</td>
<td>bytes/second</td>
<td>Average rate of reads</td>
</tr>
<tr>
<td>disk.write.bytes.rate</td>
<td>Gauge</td>
<td>bytes/second</td>
<td>Average rate of writes</td>
</tr>
<tr>
<td>disk.latency</td>
<td>Gauge</td>
<td>milliseconds</td>
<td>Average disk latency</td>
</tr>
<tr>
<td>disk.capacity</td>
<td>Gauge</td>
<td>bytes</td>
<td>The amount of disk that the instance can see</td>
</tr>
<tr>
<td>disk.allocation</td>
<td>Gauge</td>
<td>bytes</td>
<td>The amount of disk occupied by the instance on the host machine</td>
</tr>
<tr>
<td>disk.usage</td>
<td>Gauge</td>
<td>bytes</td>
<td>The physical size in bytes of the image container on the host</td>
</tr>
<tr>
<td>network.incoming.bytes.rate</td>
<td>Gauge</td>
<td>bytes/seconds</td>
<td>Average rate of incoming bytes</td>
</tr>
<tr>
<td>network.outgoing.bytes.rate</td>
<td>Gauge</td>
<td>bytes/seconds</td>
<td>Average rate of outgoing bytes</td>
</tr>
<tr>
<td>network.incoming.packets.rate</td>
<td>Gauge</td>
<td>packets/second</td>
<td>Average rate of incoming packets</td>
</tr>
<tr>
<td>network.outgoing.packets.rate</td>
<td>Gauge</td>
<td>packets/second</td>
<td>Average rate of outgoing packets</td>
</tr>
<tr>
<td>perf.cache.references</td>
<td>Gauge</td>
<td>-</td>
<td>the count of cache hits</td>
</tr>
<tr>
<td>perf.cache.misses</td>
<td>Gauge</td>
<td>-</td>
<td>the count of cache misses</td>
</tr>
<tr>
<td>network.incoming.packets.drop</td>
<td>Cumulative</td>
<td>packets</td>
<td>Number of incoming dropped packets</td>
</tr>
</tbody>
</table>
Table 1 - List of monitoring metrics collected from OpenStack VIM

2.1.2. Kubernetes (FaaS) Monitoring Metrics

The Kubernetes cluster includes one cAdvisor\(^9\) worker per node and one Prometheus\(^10\) instance for the whole cluster. Each cAdvisor worker collects metrics, including both Infrastructure-level and pod-/container-level metrics, from the proper kubernetes node and pushes them in the Prometheus. The Kubernetes publisher that has been implemented in the frame of the 5G-MEDIA project acts as a mediator among the kubernetes cluster and the MAPE. The publisher exports the metrics (e.g. container-level metrics) from the Prometheus API and feeds the monitoring part of the MAPE, actually the 5G-MEDIA Kafka bus.

For each type of metric, a specific transformation (e.g. rate, avg_over_time) is applied to obtain the rate or its average value given the start time, end time and the step. The list of collected metrics and the corresponding applied transformation functions are depicted in Table 2.

---

\(^9\) [https://github.com/google/cadvisor](https://github.com/google/cadvisor)

\(^10\) [https://prometheus.io/](https://prometheus.io/)
<table>
<thead>
<tr>
<th>Metric</th>
<th>Type</th>
<th>Unit</th>
<th>Description</th>
<th>Aggregation</th>
</tr>
</thead>
<tbody>
<tr>
<td>container_fs_inodes_free</td>
<td>Gauge</td>
<td></td>
<td>Number of available Inodes</td>
<td>avg_over_time</td>
</tr>
<tr>
<td>container_fs_io_current</td>
<td>Gauge</td>
<td></td>
<td>Number of I/Os currently in progress</td>
<td>avg_over_time</td>
</tr>
<tr>
<td>container_fs_reads_bytes_total</td>
<td>Counter</td>
<td>bytes</td>
<td>Cumulative count of bytes read</td>
<td></td>
</tr>
<tr>
<td>container_fs_usage_bytes</td>
<td>Gauge</td>
<td>bytes</td>
<td>Number of bytes that are consumed by the container on this filesystem</td>
<td>avg_over_time</td>
</tr>
<tr>
<td>container_fs_writes_bytes_total</td>
<td>Counter</td>
<td>bytes</td>
<td>Cumulative count of bytes written</td>
<td></td>
</tr>
<tr>
<td>container_memory_max_usage_bytes</td>
<td>Gauge</td>
<td>bytes</td>
<td>Maximum memory usage recorded</td>
<td>avg_over_time</td>
</tr>
<tr>
<td>container_memory_swap</td>
<td>Gauge</td>
<td>bytes</td>
<td>Container swap usage</td>
<td>avg_over_time</td>
</tr>
<tr>
<td>container_memory_usage_bytes</td>
<td>Gauge</td>
<td>bytes</td>
<td>Current memory usage, including all memory regardless of when it was accessed</td>
<td>avg_over_time</td>
</tr>
<tr>
<td>container_memory_working_set_bytes</td>
<td>Gauge</td>
<td>bytes</td>
<td>Current working set</td>
<td>avg_over_time</td>
</tr>
<tr>
<td>container_network_receive_bytes_total</td>
<td>Counter</td>
<td>bytes</td>
<td>Cumulative count of bytes received</td>
<td>rate</td>
</tr>
<tr>
<td>container_network_receive_packets_dropped_total</td>
<td>Counter</td>
<td></td>
<td>Cumulative count of packets dropped while receiving</td>
<td>rate</td>
</tr>
<tr>
<td>container_network_receive_packets_total</td>
<td>Counter</td>
<td></td>
<td>Cumulative count of packets received</td>
<td>rate</td>
</tr>
<tr>
<td>container_network_receive_errors_total</td>
<td>Counter</td>
<td></td>
<td>Cumulative count of errors encountered while receiving</td>
<td>rate</td>
</tr>
<tr>
<td>container_network_transmit_bytes_total</td>
<td>Counter</td>
<td>bytes</td>
<td>Cumulative count of bytes transmitted</td>
<td>rate</td>
</tr>
<tr>
<td>container_network_transmit_packets_total</td>
<td>Counter</td>
<td></td>
<td>Cumulative count of packets transmitted</td>
<td>rate</td>
</tr>
</tbody>
</table>
Table 2 - List of monitoring metrics collected from Kubernetes cluster

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Type</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>container_network_transmit_packets_dropped_total</td>
<td>Counter</td>
<td>-</td>
<td>Cumulative count of packets dropped while transmitting</td>
</tr>
<tr>
<td>container_network_transmit_errors_total</td>
<td>Counter</td>
<td>-</td>
<td>Cumulative count of errors encountered while transmitting</td>
</tr>
<tr>
<td>container_network_tcp_usage_total</td>
<td>Gauge</td>
<td>-</td>
<td>tcp connection usage statistics for container</td>
</tr>
<tr>
<td>container_network_udp_usage_total</td>
<td>Gauge</td>
<td>-</td>
<td>udp connection usage statistics for container</td>
</tr>
<tr>
<td>container_spec_memory_limit_bytes</td>
<td>Gauge</td>
<td>bytes</td>
<td>Memory limit for the container</td>
</tr>
<tr>
<td>container_spec_memory_reservation_limit_bytes</td>
<td>Gauge</td>
<td>bytes</td>
<td>Memory reservation limit for the container</td>
</tr>
<tr>
<td>container_network_transmit_packet_loss_percentage</td>
<td>Gauge</td>
<td>packets/sec</td>
<td>Packet loss per pod (derived metric)</td>
</tr>
<tr>
<td>container_network_receive_packet_loss_percentage</td>
<td>Gauge</td>
<td>packets/sec</td>
<td>Packet loss per pod (derived metric)</td>
</tr>
</tbody>
</table>

2.1.3. OpenNebula Monitoring Metrics

OpenNebula VIM provides an inherent monitoring subsystem that gathers information from the hypervisor relevant of the hosts and the Virtual Machines (VMs), such as the host status, basic performance indicators, as well as VM status and resource consumption. The OpenNebula publisher that has been implemented in the frame of the 5G-MEDIA project acts as a mediator among the OpenNebula infrastructure and the MAPE.

This publisher retrieves VM-level monitoring metrics from the OpenNebula XML-RPC API\(^1\) and sends them in the 5G-MEDIA Kafka bus. The metrics that are collected are listed in Table 3.

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Type</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>memory</td>
<td>gauge</td>
<td>kilobytes</td>
<td>The usage of RAM</td>
</tr>
<tr>
<td>vcpu</td>
<td>gauge</td>
<td>vCPU</td>
<td>The number of virtual CPU</td>
</tr>
</tbody>
</table>
2.2. VNFs monitoring

MAPE also supports the monitoring of the VNFs that are part of a running network service. Each VNF is able to provide periodically application specific metrics in the 5G-MEDIA Kafka bus under a specific topic. The structure of these messages is mentioned in D2.4. The following tables show the provided metrics per VNF such as vTranscoder (UC1), mid/edge vCache (UC3), vCompression Engine (UC2).

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Type</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>frame_count</td>
<td>gauge</td>
<td>-</td>
<td>The working fps of the transcoder</td>
</tr>
<tr>
<td>working_fps</td>
<td>gauge</td>
<td>fps</td>
<td>The working fps of the transcoder</td>
</tr>
<tr>
<td>no_of_profile_produced</td>
<td>gauge</td>
<td>-</td>
<td>The number of profiles (qualities) that are exposed</td>
</tr>
<tr>
<td>total_transcoding_time_ms</td>
<td>gauge</td>
<td>ms</td>
<td>The total transcoding time</td>
</tr>
<tr>
<td>theoretic_load_percentage</td>
<td>gauge</td>
<td>%</td>
<td></td>
</tr>
<tr>
<td>input_data_bytes</td>
<td>gauge</td>
<td>bytes</td>
<td>The volume of incoming data</td>
</tr>
<tr>
<td>output_data_bytes</td>
<td>gauge</td>
<td>bytes</td>
<td>The volume of outgoing data</td>
</tr>
<tr>
<td>input_mesh_codec</td>
<td>-</td>
<td>-</td>
<td>The codec of the input mesh, e.g. DRACO</td>
</tr>
<tr>
<td>no_of_input_textures</td>
<td>gauge</td>
<td>-</td>
<td>The number of input textures</td>
</tr>
</tbody>
</table>
transcoder_engine | - | - | The version of the transcoder engine, e.g. v2.8.4-GPU

Table 4 - List of provided metrics from the vTranscoder VNF (multiple qualities)

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Type</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_fps</td>
<td>gauge</td>
<td>fps</td>
<td>The current fps</td>
</tr>
<tr>
<td>num_frame</td>
<td>gauge</td>
<td>frames</td>
<td>The actual frame</td>
</tr>
<tr>
<td>enc_quality</td>
<td>gauge</td>
<td>-</td>
<td>The encoding quality</td>
</tr>
<tr>
<td>enc_dbl_time</td>
<td>gauge</td>
<td>seconds</td>
<td>The encoding time</td>
</tr>
<tr>
<td>enc_speed</td>
<td>gauge</td>
<td>kbps</td>
<td>The encoding speed</td>
</tr>
<tr>
<td>act_bitrate</td>
<td>gauge</td>
<td>kbps</td>
<td>The actual bitrate</td>
</tr>
<tr>
<td>avg_bitrate</td>
<td>gauge</td>
<td>kbps</td>
<td>The average bitrate</td>
</tr>
<tr>
<td>max_bitrate</td>
<td>gauge</td>
<td>kbps</td>
<td>The maximum bitrate</td>
</tr>
<tr>
<td>gop_size</td>
<td>gauge</td>
<td>-</td>
<td>Distance between two consecutive keyframes</td>
</tr>
</tbody>
</table>

Table 5 - List of provided metrics from the vCompression Engine VNF

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Type</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bytes_recv</td>
<td>gauge</td>
<td>bytes</td>
<td>The volume of incoming bytes (incremental counter)</td>
</tr>
<tr>
<td>bytes_sent</td>
<td>gauge</td>
<td>bytes</td>
<td>The volume of outgoing bytes (incremental counter)</td>
</tr>
<tr>
<td>drop_in</td>
<td>gauge</td>
<td>-</td>
<td>The number of incoming dropped packets</td>
</tr>
<tr>
<td>drop_out</td>
<td>gauge</td>
<td>-</td>
<td>The number of outgoing dropped packets</td>
</tr>
<tr>
<td>err_in</td>
<td>gauge</td>
<td>-</td>
<td>The number of errors in the incoming data</td>
</tr>
</tbody>
</table>
Table 6 - List of provided metrics from the vCache VNF (per network interface)

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Type</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>err_out</td>
<td>gauge</td>
<td>-</td>
<td>The number of errors in the outgoing data</td>
</tr>
<tr>
<td>packets_recv</td>
<td>gauge</td>
<td>packets</td>
<td>The number of incoming packets</td>
</tr>
<tr>
<td>packets_sent</td>
<td>gauge</td>
<td>packets</td>
<td>The number of outgoing packets</td>
</tr>
</tbody>
</table>

Table 7 - List of provided metrics from the vCache VNF (statistics)

In addition to the monitoring of the VNFs, MAPE takes into account monitoring metrics from involved services such as media servers (PNFs) that streams content and participants the running network services (e.g. spectators in UC1). The structure of each message is described in D2.4 while the following tables list the metrics that are collected.

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Type</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bitrate</td>
<td>gauge</td>
<td>Mbps</td>
<td>The current bitrate</td>
</tr>
<tr>
<td>bitrate</td>
<td>aggregated</td>
<td>Mbps</td>
<td>The aggregated bitrate</td>
</tr>
<tr>
<td>framerate</td>
<td>gauge</td>
<td>fps</td>
<td>The current frame rate</td>
</tr>
<tr>
<td>framerate</td>
<td>aggregated</td>
<td>fps</td>
<td>The aggregated frame rate</td>
</tr>
</tbody>
</table>
### Table 8 - List of metrics provided from the spectators in UC1

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Type</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stall_events</td>
<td>aggregated</td>
<td>counts/minute</td>
<td>The aggregated number of stall events</td>
</tr>
<tr>
<td>stall_time</td>
<td>aggregated</td>
<td>%</td>
<td>The percentage of aggregated stall time</td>
</tr>
<tr>
<td>Latency</td>
<td>gauge</td>
<td>milliseconds</td>
<td>The current latency</td>
</tr>
<tr>
<td>Latency</td>
<td>aggregated</td>
<td>milliseconds</td>
<td>The aggregated latency</td>
</tr>
<tr>
<td>Mesh_dec_time</td>
<td>gauge</td>
<td>milliseconds</td>
<td></td>
</tr>
<tr>
<td>quality_shifts</td>
<td>gauge</td>
<td>counts/minute</td>
<td></td>
</tr>
</tbody>
</table>

### Table 9 - List of metrics provided from the PNF media server (Plex) in UC3

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Type</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>response_data_total_bandwidth</td>
<td>gauge</td>
<td>Kbps</td>
<td>Total bandwidth consumed by active streaming sessions</td>
</tr>
<tr>
<td>response_data_wan_bandwidth</td>
<td>gauge</td>
<td>Kbps</td>
<td>WiFi bandwidth consumed by active streaming sessions</td>
</tr>
<tr>
<td>response_data_stream_count_transcode</td>
<td>gauge</td>
<td>Kbps</td>
<td>Number of active transcoding sessions</td>
</tr>
<tr>
<td>response_data_stream_count_direct_play</td>
<td>gauge</td>
<td></td>
<td>Number of streamed videos in direct mode, i.e. not live transcoded</td>
</tr>
<tr>
<td>response_data_lan_bandwidth</td>
<td>gauge</td>
<td>Kbps</td>
<td>LAN bandwidth consumed by active streaming sessions</td>
</tr>
<tr>
<td>response_data_stream_count</td>
<td>gauge</td>
<td></td>
<td>Total active streaming sessions</td>
</tr>
</tbody>
</table>

### 2.3. Publish/subscribe mechanism dashboard

An open source tool, Kafka Manager UI\(^{12}\), has been integrated with the Kafka bus, as depicted in Figure 1, offering management capabilities over the running 5G-MEDIA Kafka bus. It is a web-based application that lists the registered topics in the Kafka bus including the number of partitions per topic, the number or replicas per topic, the incoming messages rate per topic and the total messages per topic. Moreover, the Kafka Manager UI lists the consumer groups per topic and the lag per consumer group that reflects the delay in the processing of incoming topics.

\(^{12}\) [https://github.com/yahoo/kafka-manager](https://github.com/yahoo/kafka-manager)
messages. The aforementioned publishers as well as the monitoring agents are running in the VNFs publishing messages in the 5G-MEDIA Kafka bus are considered as producers. Each consumer maps to a unique consumer group; the consumers that belong in the same consumer group share the messages while consumers in different consumer groups process all the messages.

![Figure 2 - Indicative list of topics in the Kafka Manager UI](image-url)
2.4. Monitoring dashboards

A set of monitoring dashboards have been developed in the 5G-MEDIA project that visualizes the monitoring metrics made available in the Kafka bus, in real time. More specifically, we provide:

- A dashboard for the OpenStack VDUs (VMs),
- A dashboard for the OpenNebula VDUs (VMs)
- A dashboard for the Kubernetes VDUs (containers)
- Dashboards for VNFs such as vTranscoder (UC1), vCompressionEngine (UC2) and vCache (UC3), and PNFs such as Plex media server (UC3)
- A dashboard for the listing of the applied optimization plans.

![Figure 3 - List of consumer groups in Kafka Manager UI](image)
Figure 4 - Visualization of a running VNF (single VDU) in Kubernetes

Figure 5 - Visualization of a running vTranscoder VNF, part of the UC1 service, deployed in Kubernetes NFVI

Figure 6 - Visualization of a running vCompressionEngine VNF, part of the UC2 service, deployed in OpenNebula NFVI
The aforementioned dashboards can easily be imported in a new MAPE deployment. These are also available in the 5G-MEDIA public repository – see Appendix A for details.
3. Cognitive Network Optimiser

3.1. Role of CNO optimisation algorithms in 5G-MEDIA

Figure 8 shows the overall role of the Cognitive Network Optimiser (CNO) in 5G-MEDIA. The CNO operates at multiple levels: firstly with the overall scope of all applications and services under the control of the 5G-MEDIA SVP (CNO_platform) and then at a lower level with the scope of an individual application/service type in each of the project’s use cases, which we refer to as the use case level (CNO_UC).

- CNO_platform is responsible for managing physical resources including computing and networking resources. The 5G-MEDIA platform purchases/rents resources from underlying network and computational resource providers. Given the purchase cost, CNO_platform needs to perform optimization algorithm to efficiently use its resources and sells its services to each use case (service providers) to get revenue. Based on the optimization results, CNO_platform needs to publish a QoE/cost list (Figure 10) so that each use case can choose their operating point within their budget/QoE constraints.

The overall platform-level optimisation algorithm considers a large set of resources that are made available from the underlying computational and network resource providers, consisting of multiple edges and regional DCs with many network path options between them. The result of the overall optimisation step defines the locations where the NSs should run and the range of resources (computational and network) that need to be made available to the use-case specific CNO algorithms. In other words,
it defines a constrained subset of resources that the use case algorithms can work within.

- CNO\textsubscript{UC} – CNO at use case level is responsible to adapt behaviour of the application to improve QoE within the resource constraints defined by the output of the optimisation algorithms at the CNO\textsubscript{platform} level. Note that the CNO\textsubscript{platform} allocates resources to each use case based on its budget/QoE constraints. For instance, in the demonstration scenarios we have defined for UC2 (see 5G-MEDIA deliverable D6.3), we assume that the CNO\textsubscript{platform} has previously identified that 150 Mbps of network capacity (shared with background traffic) are required for the remote production services, given the competing demands of all services on the available network resources. Given that constraint, the reinforcement learning algorithm in UC2 tries to adapt compression levels for each session given the maximum bandwidth of 150 Mbps and the amount of dynamically varying background traffic.

It should be noted than an alternative strategy could be for the CNO\textsubscript{platform} to reserve a dedicated amount of network bandwidth for UC2. In which case 150 Mbps (or whatever quantity of bandwidth had been determined by the service provider) would be allocated for the remote production service for the duration of the session. In this case there would be no need for a CNO\textsubscript{UC} algorithm to dynamically manage the compression levels as sufficient bandwidth would be reserved for the maximum bitrate of the video at the selected compression level. Although this would guarantee that network capacity is available for the session duration it is likely to result in an underutilisation of network resources as any unused capacity could not be used for any other purpose. Hard reservation of a fixed capacity would also mean a higher cost to the service provider and hence the end user.

In summary, CNOs need inputs from the platform and/or from the use case applications. Based on those inputs, CNOs apply optimization algorithm to find the best action to efficiently manage/use the underlying resources. In the 5G-MEDIA platform, CNOs get input information from the monitoring component which is responsible for gathering resources’ information from the platform and from the applications, then publishes them to Kafka bus so that the CNOs can easily get access. On the other hand, CNOs send actions to Kafka bus which are then implemented by suitable components to make change to the applications or the underlying resources.

The testbeds available to the 5G-MEDIA project are limited in size and are suitable for demonstrating the use case level CNO algorithms on the assumption that the platform-level optimisation algorithm has, in a prior step, defined the set of resources for the use-case algorithm to work within. Evaluation of the platform-level optimisation algorithms requires a much larger set of testbed resources (number of computational nodes, physically diverse edges, multiple options for routing paths, etc.) and/or a large set of service users spread over a wide geographical area. For this reason, the evaluation of the overall platform-level optimisation is mainly limited to simulation studies, as outlined in section 3.2.2. A small-scale evaluation of the overall optimisation algorithm making trade-offs between demands from different services in two of the project use cases will be described in the final project use case evaluation deliverables D6.2 and D6.3.
3.1.1. CNO for 5G-MEDIA platform (CNO\textsubscript{platform})

The CNO platform is responsible for managing physical resources (including networking and computing resources). Its role is to provide an overall optimization across multiple use cases (services) to guarantee fairness as well as maximize revenue for 5G-MEDIA platform. Therefore, CNO platform needs to perform optimization tasks considering the trade-off between QoE and cost. In other word, CNO platform needs to compute a Pareto front\textsuperscript{13} as shown in Figure 9.

![Figure 9 - Pareto front of cost and QoE](image.png)

In brief, the Pareto front shows an optimal relationship between cost and QoE. That is to say given a cost value, based on the Pareto front, we can identify the maximum QoE the system can achieve. Or the other way around, to guarantee a given QoE score, the Pareto front shows what is the minimum cost to the platform. Given the Pareto front as a reference, the 5G-MEDIA platform will choose a suitable operating point. For example, an operating point can be chosen as shown in Figure 9 as it provides a fairly high QoE with a reasonable low cost.

After choosing an operating point, CNO platform will publish a quality/cost list which can be a simple table as shown in Figure 10. The list identifies the range of QoE/costs that the platform is able to provide to the service providers operating the services/use cases, given the resources available from the underlying network and computational resource providers, the expected mix of services and their targeted quality levels. It may not be possible for all services to simultaneously request the highest quality levels. Due to resource constraints it may not be possible to purchase additional network or computational resources from the underlying infrastructure providers due to a real-world bound on available resources. This situation should be reflected in the price associated with the quality levels: when there is a high demand for physical resources, one strategy of the platform provider may be to increase the price for the higher quality levels to reduce the demand for the higher quality levels. If demand increases so that service providers are not able to request the QoE levels they would like at a sufficiently low cost then additional infrastructure resources would need to be provided: one possibility would be for the platform to seek relationships with additional infrastructure providers to increase network and/or computational capacity or to indicate to existing infrastructure providers that demand outstrips their capacity and that they should consider

\textsuperscript{13} https://www.igi-global.com/dictionary/pareto-front/21878
deploying additional resources. The latter interactions to increase the available infrastructure resources are out of scope of the current system under investigation but could be an aspect of future study.

Given the quality/cost options from the platform each use case can select the one that is suitable for their budget/QoE requirements.

<table>
<thead>
<tr>
<th>Quality level</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q₁</td>
<td>C₁</td>
</tr>
<tr>
<td>Q₂</td>
<td>C₂</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Qₙ</td>
<td>Cₙ</td>
</tr>
</tbody>
</table>

*Figure 10 - QoE/cost list*

After QoE/cost levels are selected, CNO_{platform} looks at current available resources and use cases’ demand, performing an optimization algorithm to make a set of related decisions: which computing nodes should host which VNFs? how VNFs are connected to form a VNFFG? and what are the underlying paths to route traffic between VNFs in a VNFFG? We present in Section 3.2 algorithms to answer those questions.

Conflicts between competing service providers may be handled on a first-come first-served basis. Currently deployed services under existing agreements between service providers and the platform should take precedence over new requests. The QoE/cost list published by the platform provide at any instant will reflect the available resources given the currently contracted services. If a service provider wishes to exceed the available resources with additional demand or for higher quality levels (implying more computation or network resources are required than the underlying infrastructure providers can supply) then the platform can investigate whether additional resources can be made available through new infrastructure providers or through existing ones deploying additional physical equipment. When physical resources have been exhausted supply and demand will be matched through the pricing strategy of the platform provider.

### 3.1.2. CNO for individual use cases

In this section, we briefly describe the objective of CNO for each use case. We have designed three CNO machine learning models for the three use cases. Depending on specific requirements of each use case, the CNO looks at current available resources (both computing and networking resources) to make decision on appropriate actions for the next step.

**Use case 1:** overall state of the system will be a combination of infrastructure, application-level and QoE metrics. Based on those states, CNO is trained to make decision on compression levels, number of encodings, CPU/GPU and FaaS/non-FaaS selection. (See Section 3.4 for more detail).

**Use case 2:** we have designed the CNO algorithm using the Asynchronous Advantage Actor-Critic (A3C) reinforcement learning framework. Inputs of the algorithm are network related
metrics such as available bandwidth and packet loss rate. Given QoE score feedback, CNO can select appropriate compression level and fairness policy for multiple sessions. (See Section 3.5 for more detail about the algorithm).

**Use case 3:** we use supervised learning for UC3 CNO algorithms. Based on history of the traffic load, CNO can forecast congestion on the network between vCaches and the users, which may cause performance degradation for the delivered video, reducing QoE for the users. Therefore, CNO needs to trigger scale-out operation to deploy additional vCaches available over non-congested network segments. (See Section 3.6 for more detail).

### 3.2. The overall optimization of CNO platform

In this section, we design optimisation algorithm for CNO platform. The goal is to answer the three questions aforementioned: (1) which computing nodes to host which VNFs? (2) how VNFs are connected to form a VNFFG? and (3) what are the underlying routing to route traffic between VNFs in a VNFFG?

Each use case requires different kinds of VNFs. On the other hand, those VNFs need to be connected to perform a general task which is called VNFFG. CNO platform needs to know the structure of VNFFGs so that it can efficiently allocate resources for each of the use case.

#### 3.2.1. VNF forwarding graph (VNFFG) structures

In this section, we introduce three basic structures of VNFFGs. These structures can be combined to form any complex VNFFG structures. We then formally define the problem and design CNO platform as an evolutionary algorithm to simultaneously optimise cost, latency and congestion for a general VNFFG structure.

**3.2.1.1. VNFFG structures**

![Figure 11 – VNFFG Linear structure](image)

One of the most common structures of VNFFG is the chain or linear structure. As an example, a user needs to watch a video on demand (VoD) content on his device (e.g. smart phone, tablet or smart TV). In Figure 11, the three virtual services vOriginServer, vTranscoder and vCache need to be connected as a chain to deliver a video stream to the user. The vOriginServer acts as the root server sending the stream to the vTranscoder and the vCache. The vTranscoder is responsible for transcoding the video stream into varying levels of quality to support adaptive streaming. The vCache is used to cache/store encoded video streams so that it can serve users quickly without connecting to the vOriginServer. For this linear structure, the end-to-end latency will be accumulated from each hop of the chain (including network latency and processing delay at each node).
Figure 12 – VNFFG Tree structure

Figure 12 shows an overview of the scenario covered by the remote production in broadcasting use case. The on-site reporter needs to send high quality content back to the media process engine before being broadcasted. The encoded video and the translated audio produced by the transcoder and the translation engine will be merged at the media process engine before sending out to the viewers. The translating step can be done in parallel with the encoding step and the end-to-end latency will be the longest of the two branches. This structure forms a directed acyclic graph and is referred to as a tree or a parallel structure.

Figure 13 – VNFFG Loop structure

The third structure we are presenting is a cycle or a loop structure. An example could be the closed loop system shown in Figure 13. It is similar to the chain structure except there is a loop to provide feedback for making decisions in subsequent rounds. The end-to-end latency is accumulated from each hop. Examples of this include services where components send any sort of application feedback to the source.

3.2.1.2. Problem definition

Figure 14 - Optimisation of routing and placement for VNFFGs
We use the linear structure use case (Figure 11) as an example of the optimisation problem addressed by CNO\textsubscript{platform}. In brief, high-quality communication paths between the \emph{vOriginServer} and the users are critical for VoD systems. Performance of the system is related to available bandwidth, latency of network links, computational capability and location of servers where service instances are deployed. To guarantee QoE, it may be necessary to modify the network path (e.g. via an SDN controller) or select service instances in different locations. In Figure 14, we show an example of two users watching the same VoD stream. In this example, there are several servers where we can deploy service instances (\emph{vOriginServer}, \emph{vTranscoder} and \emph{vCache}). We need to create a linear structure (or VNF chain) to connect between the \emph{vOriginServer}, the \emph{vTranscoder}, the \emph{vCache} and each of the users as a chain. To reduce operational costs, the \emph{vOriginServer} and the \emph{vTranscoder} instances can be shared between the two chains as shown in Figure 14. However, we assume that the link (\emph{R}_1 - \emph{R}_2) has limited bandwidth. Therefore, CNO\textsubscript{platform} decides to route traffic flows of the two chains from \emph{S}_1 to \emph{S}_2 via different paths to avoid making the link (\emph{R}_1 - \emph{R}_2) congested. On the other hand, to guarantee low latency, the \emph{vCache} needs to be located close to the end user, thus CNO\textsubscript{platform} deploys the two instances of the \emph{vCache} in \emph{S}_3 and \emph{S}_4 for the two users.

![Figure 15 - VNFFG placement with computational resource constraint](image)

Another example of VNFFG is shown in Figure 15. Assuming that the \emph{vTranscoder} in \emph{S}_2 can serve only one stream due computational constraint (e.g. CPU and/or memory), CNO\textsubscript{platform} then is able to find alternative solution as shown in Figure 15. The new solution needs to use \emph{S'}\emph{S}_2, which is worse in term of latency, however the computational constraint is satisfied. To sum up, the CNO\textsubscript{platform} algorithm needs to consider deployment cost, application delay, computational resource and network congestion in making decisions on service instance placement and the corresponding underlying network routing.

In this section, we will firstly define the data structures involved in the definition of the VNFFG structure. Then, we will introduce the concepts related to service placement and network routing optimisation and show how we connect them in CNO\textsubscript{platform}. 
### 3.2.1.3. General-purpose tree definition

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_e$</td>
<td>Network capacity of link $e$</td>
</tr>
<tr>
<td>$C_v$</td>
<td>Computing resource (abstraction of CPU/RAM) at node $v$</td>
</tr>
<tr>
<td>$D_i$</td>
<td>Demand of user $i$</td>
</tr>
<tr>
<td>$E$</td>
<td>Set of links in the network</td>
</tr>
<tr>
<td>$F_{kj}$</td>
<td>Fixed cost of service $j$ at server $k$</td>
</tr>
<tr>
<td>$F_{v_{kj}}$</td>
<td>Variable cost of service $j$ at server $k$</td>
</tr>
<tr>
<td>$G = (V, E)$</td>
<td>Network topology</td>
</tr>
<tr>
<td>$L_e$</td>
<td>Latency of link $e$</td>
</tr>
<tr>
<td>$L_{fh_{min}}^0$</td>
<td>Min first hop latency</td>
</tr>
<tr>
<td>$L_{fh_{max}}^0$</td>
<td>Max first hop latency</td>
</tr>
<tr>
<td>$L_{e2e_{min}}^0$</td>
<td>Min end-to-end latency</td>
</tr>
<tr>
<td>$L_{e2e_{max}}^0$</td>
<td>Max end-to-end latency</td>
</tr>
<tr>
<td>$P_{latency}$</td>
<td>Latency penalty</td>
</tr>
<tr>
<td>$P_{cost}$</td>
<td>Cost penalty</td>
</tr>
<tr>
<td>$P_{congestion}$</td>
<td>Congestion penalty</td>
</tr>
<tr>
<td>$P_{fh_i}$</td>
<td>First hop latency penalty of user $i$</td>
</tr>
<tr>
<td>$P_{e2e_i}$</td>
<td>End-to-end latency penalty of user $i$</td>
</tr>
<tr>
<td>$r$</td>
<td>Root of tree $T$</td>
</tr>
<tr>
<td>$s_T$</td>
<td>List of successors in tree $T$</td>
</tr>
<tr>
<td>$S_j$</td>
<td>Servers are available to deploy service $j$</td>
</tr>
<tr>
<td>$Serv_T$</td>
<td>List of distinct services in tree $T$</td>
</tr>
<tr>
<td>SPO</td>
<td>Service placement optimization only</td>
</tr>
<tr>
<td>$T = (r, s_T)$</td>
<td>VNFFG tree</td>
</tr>
<tr>
<td>$u_e$</td>
<td>Utilization of link $e$</td>
</tr>
<tr>
<td>$U = {i}$</td>
<td>Set of user $i$</td>
</tr>
<tr>
<td>$U_{min}$</td>
<td>Min link utilisation</td>
</tr>
<tr>
<td>$U_{max}$</td>
<td>Max link utilisation</td>
</tr>
<tr>
<td>$V$</td>
<td>Set of nodes in the topology</td>
</tr>
</tbody>
</table>

*Table 10 - Summary of key notations (alphabetical order)*
A given instance of the VNFFG structure will be represented by a tree where each node represents a service instance that needs to be deployed in a suitable server and the tree will define the dependencies between service instances. The branches on the tree indicate services that can be executed in parallel, while a linear chain of nodes indicates services that need to be completed sequentially. We use nodes with replicated identifiers to indicate the same service instance in the loop structure (see Figure 16c). In this way, we have a general representation scheme that allows to define different service structures.

We show in Figure 16 how the three VNFFGs be represented. In Figure 16b, the Transcoder and the Translation Engine need to connect to the root which is the OriginServer. Then they both connect to the Media Process Engine and the tree is terminated at the user. Note that in Figure 16c the Com & Adj (1) and Com & Adj (2) are the same Compare & Adjust instance (with replicated identifier) in the loop structure in Figure 13.

The tree is represented by a root node \( r \) and a function \( s_T \) that provides, for each node, the list of its successors in the tree: \( T = (r, s_T) \). Note that for a given service \( s \), the list \( s_T(n) \) will provide the services that need to be completed in order for \( s \) to be able to start. For a given service tree \( T \), we can also define \( \text{Serv}_T \), which is the list of services in the tree, i.e. the full set of services that appear at least once in the tree's nodes. \( |\text{Serv}_T| \) will therefore be used to denote the number of distinct services.

### 3.2.1.3.1. Service placement optimization

The optimisation algorithm will define which servers to deploy service instances for each user. This decision is made by trading off cost with performance of the network, such as latency, computational resources and congestion. Each server \( s_{kj} \in S_j \) (\( S_j \) represents the set of all servers able to deploy service \( j \)), will be defined by a tuple \( (F_{kj}^f, F_{kj}^v, R_{kj}) \) containing, respectively, the fixed deployment cost \( (F_{kj}^f) \) of the service \( j \) at server \( k \), its variable cost \( (F_{kj}^v) \) and maximum number of available session slots \( R_{kj} \) - an abstraction of computational resource constraint at node \( k \).

- **Fixed cost**: the cost of deploying the service instance for the first time at a server. For example, this can be thought as the cost of software installation in that server. The fixed cost is incurred only once and does not vary with the number/size of service instances at a certain location.
- **Variable cost**: this cost is proportional to the resources used by the service instances. The more service instances are deployed, the more resources are consumed and hence the cost increases with the number of instances.
- **We use session slot** as a unit of measurement representing how many user requests can be accommodated simultaneously due to computational resource constraints (e.g. CPU/memory). This is to guarantee that the available of session slots at node \( k \) is sufficient to serve user requests.
The service placement optimisation algorithm can be executed at various timescales, including initial service deployment and ongoing reconfiguration to migrate existing services, instantiate new services, undertake service scaling as demand patterns change. On the other hand, VNFFG optimisation is used to determine which instances of services should be interconnected to meet performance and cost objectives for specific user session requests. This can be undertaken at initial session establishment, as well as for the optimisation of already running sessions/VNF forwarding graph (VNFFG) instances.

3.2.1.3.2. Routing optimization

In order to jointly optimise for server placement and network level performance we add the network routing dimension to the problem. Thus, we will consider a network defined as a directed graph $G = (V, E)$, where $V$ is the set of nodes, while $E$ represents links connecting them. Each link $e \in E$ has a capacity $C_e$ defining the amount of traffic it can accommodate. In addition, each link $e \in E$ is associated with a propagation delay $L_e$. When demands are allocated to the links in the network, the total amount of traffic allocated over a link will be given by its load $l_e$. From the link capacity and load, a link utilization $u_e$ is calculated as: $u_e = \frac{l_e}{C_e}$.

We assume that each user $i \in U$ is connected to a node in the network topology (see Figure 14). The routing optimisation objective is to find an optimal end-to-end path connecting from the first to the subsequent service instances in the VNFFG structure and finishing at the end-user. The optimal path needs to take into account network latency and link loads. In addition, the routing optimisation also provides feedback to the service placement optimisation. For instance when the placement optimisation receives feedback saying that the link $(R_2, S_2)$ in Figure 14 is congested, it should instantiate new vTranscoder in another location to reduce the load on the link $(R_2, S_2)$. The CNO\textsubscript{platform} optimisation algorithm needs to take into account three dimensions in the formulation of the objective function: latency, cost and network congestion. We define, in the next subsection the multi-objective cost function used in the CNO\textsubscript{platform} algorithm.

3.2.1.4. Multi-objective cost function

3.2.1.4.1. Latency

![Figure 17 - Penalty function $p$ used to normalise the latency and cost metrics](image-url)
We consider latency which includes both network latency and processing time at the server where service instances are deployed. Also, some services require an extreme low latency between the users and the first hop service instance. For example, users should connect to a low-latency rendering component in an online game service to reduce lag as the player moves viewpoint, while the game simulation engine itself could be located more remotely, if the positions of other players do not change rapidly and so a longer latency would not impact game play. Therefore, along with the end-to-end latency, we also consider the first hop latency as a component of the cost function when deploying a VNFFG.

Inspired by the utility function from the work [Phan18a], we define a minimum and the maximum first hop and end-to-end latency thresholds. Hence, each user request, \( i \in U \), will be defined by a tuple \( (D_i, L_{fh}^{min}, L_{fh}^{max}, L_{e2e}^{min}, L_{e2e}^{max}) \), which define, respectively, the user demand, and the minimum and maximum required first hop and end-to-end latency. The minimum and maximum latency thresholds are used to capture the nature of the penalty imposed over the performance for services. For some services, if the latency is less than \( L_{min} \), the improvement is not perceived by the users of that service, thus the penalty is always zero. For instance, in high-quality voice over IP service, if latency is less or equal to \( L_{min} = 20 \) ms, the users get a feeling of real voice communication [Stone99]. Therefore, there is no need to consider any penalty \( (P = 0 \text{ - Figure } 17) \) when the latency is less or equal to \( L_{min} \). On the other hand, 150 ms is about the limit for keeping the user’s attention focused. Therefore, a huge penalty needs to set if the latency is more than \( L_{max} = 150 \) ms. We show in Figure 17 a graph of penalty vs. metric, where the metric \( X \) in this case refers to latency.

\[
P_{\text{latency}} = \frac{1}{2 \cdot |U|} \sum_{i \in U} (P_{e2e}^{i} + P_{fh}^{i})
\]

where \( P_{e2e}^{i} \) and \( P_{fh}^{i} \) represent the penalties for end-to-end and first-hop latencies for user request \( i \), being obtained by applying the penalty function \( p \) depicted in Figure 17 to the computed values of the respective latencies (\( Y \) was set to 10 in the evaluation work described in section 3.2.2), given an assignment of services to servers. Note that in the current model, we do not consider the impact of network congestion on the latency. For example, congestion on a link can increase queuing delay which in the end impacts on the overall latency. We plan a future work to incorporate queuing delay into the general latency model.

### 3.2.1.4.2. Deployment costs

We optimise both the fixed cost and the variable costs (see 3.2.1.3.1) in the CNO platform algorithm. The total cost \( TC \) of a solution (an assignment of services to specific servers) is calculated by \( TC = FC + VC \), summing the fixed costs (\( FC \)) incurred in the used servers and the variable costs (\( VC \)) for all user requests. The fixed cost is, thus, calculated as follows:

\[
FC = \sum_{j \in \text{Serv}} \sum_{k \in S_j} b_{jk} \times F_{kj}^{f}
\]

where \( b_{jk} \) is a binary variable, taking value 1 if the server \( s_k \) is used to deploy service \( j \), and 0 otherwise.

On the other hand, the variable costs are calculated as:
where \( b_{jki} \) is a binary variable, taking value 1 if the server \( s_k \) is used to deploy service \( j \) processing the user request \( i \), which has a total demand of \( D_i \).

To be able to create a unified cost function, we also normalise the cost penalty - \( P_{\text{cost}} = p(TC) \) - using the convex piecewise linear function as shown in Figure 17. The penalty cost is zero if the sum of fixed cost and variable cost are less or equal than a defined minimum budget \( \text{Cost}_{\text{min}} \), and we set a fast-growing penalty for total costs above a threshold \( \text{Cost}_{\text{max}} \). In our experiments, \( \text{Cost}_{\text{min}} \) was calculated as the minimum possible cost to satisfy all demands, while \( \text{Cost}_{\text{max}} \) was set to be equal to \( 2 \times \text{Cost}_{\text{min}} \).

### 3.2.1.4.3. Network congestion

We have adopted the congestion metrics proposed by Fortz and Thorup in their seminal paper [Fortz00] which is defined by a cost (or penalty) \( \Phi \) which is the sum of the cost function \( \Phi_e(u_e) \) for each link \( e \in E \). This is a convex piecewise linear penalty function, which has a low growth for small link utilisations, but increases more quickly with the link utilisation (see the piecewise convex penalty function in [Fortz00]). It has the advantage over metrics as maximum link utilisation of considering all links in the network, while also penalising heavily overloaded links.

As proposed in the original publication, this metric can be normalized over distinct network topologies, computing \( \Phi^* \), which divides the penalty \( \Phi \) over the minimum load in the network, obtained by routing over shortest paths with link weights set to 1. In this case, the optimal theoretical value of the objective function is 1.

Note that the definition of this objective function follows the same principles of the two previous ones. Since the optimal value in this case is 1, we set \( P_{\text{congestion}} = 1 - \Phi^* \). In this way, network congestion can be combined with latency and cost in the multi-objective formulation.

### 3.2.1.4.4. Overall cost function

Here, we consider two scenarios: in the first, which we will denote by service placement optimisation only (SPO), we address the optimisation of the assignment of the required services for all user requests seeking to minimize \( P_{\text{latency}} \) and \( P_{\text{cost}} \), while in the latter (our CNO platform algorithm) we will consider also routing in the network, and therefore will add a third objective \( P_{\text{congestion}} \) to the optimisation framework.

In this work, we consider an aggregation of the different cost functions through a sum of their values, weighted by considering individual parameters for each component. Thus, our objective will be to minimize \( P \), defined as:

\[
P = \alpha P_{\text{latency}} + \beta P_{\text{cost}} + \gamma P_{\text{congestion}}
\]

As a way to further normalize the results, we will consider only cases where \( \alpha + \beta + \gamma = 1 \). In the case of SPO, we will consider \( \gamma \) to be set to 0.

Notice that as the latency, cost and congestion values are normalized, the optimal value of each of the components of the cost function is 0, and therefore this will also be the theoretical...
optimal value of \( P \), although this might not be achievable given the specific instances. In addition, when we set \( \alpha = \beta = \gamma = 1/3 \), the three components are considered in the optimization process with the same priority.

### 3.2.1.5. Evolutionary algorithm

#### 3.2.1.5.1. Overall structure of the EA

![Evolutionary algorithm flow chart](image)

*Figure 18 - Evolutionary algorithm flow chart*

We address the optimisation problem formulated in the previous section through the use of Evolutionary Algorithms (EAs). These are justified in this case by the complexity of the underlying optimisation problem and by the numerous cases where they have shown success in related tasks [Phan18b] [Rocha07]. We present in Figure 18 a general flowchart showing the main steps of the EA. We implemented the EA based on the *inspyred* package\(^{14}\) in Python, following the default structure of the provided EA.

Next, we explain in detail the steps of the EA:

- **Initialisation**: The initial population is a set of potential solutions to the problem. Each solution contains a list of integers, each being randomly generated within the allowed range for each position.

- **New population**: after each generation, a new population is created by the following steps until reaching the stopping criterion:
  - **Selection**: select parents from a population according to their fitness (the lower the fitness, the higher chance to be selected).
  - **Crossover**: using the selected crossover operator, create new individuals (offspring) from the selected parents.
  - **Mutation**: over the generation, use the selected mutation operators to change part of their genes.

\(^{14}\) [https://pypi.org/project/inspyred/]
- Reinsertion: the newly generated offspring will be combined with some individuals selected from the previous population to create the next population.

Note that each time of computing a solution for a user request, we only consider nodes which have available session slots to guarantee computational resource constraint.

- Stop: if the termination criterion is reached (maximum number of solution evaluations), the algorithm stops, and returns the best solution in the current population.

3.2.1.5.2. Solution representation and evaluation

A solution for the considered optimisation problem is represented as an individual in the EA, and is encoded as a list of integer values. This list encompasses two parts, the first represents the assignment of the services to specific servers, for each user request, while the latter represents the way routing is done in the network, through the definition of weights for each network link, used to route the traffic using an intra-domain routing protocol as OSPF. Notice that in the SPO case, only the first part of the solutions exists.

Regarding the first part, the service-server assignment, the length is given by $|U| \times |ServT|$. Thus, for each user request, all services defined in the structure have an assigned server. The integer value at each position will specify the index of the server assigned to deploy that specific service for that user request. This part of the solution is decoded into a finite function indexed by user identifier. For each user, another finite function from service to server identifier keeps the assignment for that specific user request.

Regarding the second part, encoding routing weights, the length is given by the number of links in the network. Each value specifies the weight of a link, to be used in the shortest path calculation for each demand, using the defined intra-domain routing protocol (e.g. OSPF).

The range of the values in this part of the individual is the set $[1, MaxW]$. In the experiments, $MaxW$ was considered to be 20.

Each individual is evaluated by first decoding its genome (list of integer values) into a service-server assignment for each user, and if routing is optimised into the OSPF weights, as described. The service-server assignment is used to compute the penalties for latency and costs. The routing weights are used to compute the full paths for each user request, from those the respective link loads and utilisations, and finally compute the congestion penalty. The fitness of each individual is given by the overall penalty $P$.

3.2.1.5.3. Genetic operators

The EA implemented in this work, makes use of both crossover and mutation operators. Regarding the crossover, we use a single operator provided by the inspyred package, namely the one-point crossover. This takes two parents, cuts in a random position and combines the genes alternating from each parent, generating two new solutions (individuals).

Regarding mutation, we make use of three different operators: random mutation, an operator from the inspyred package which randomly changes the value of a given gene for another in
the allowed range for that position (given a probability per position); *increment mutation*, which changes a randomly selected gene adding or subtracting one; and, an *intelligent mutation*, which works as random mutation but only accepts the change if it leads to a higher fitness value, being the process repeated $N_{mut}$ times (in the experiments, we considered its value to be 20).

### 3.2.1.5.4. Other parameters

In the EA, the selection of the parents to undergo recombination is based on ranking selection, where individuals are ranked and selection is made based on values calculated from the ranking position. The recombination scheme is generational, being replaced in each generation 80% of the individuals in the population, with an elitism value of 2 (i.e. the two best individuals are always kept). The population size was set to 100, and the termination criterion is based on a maximum number of solutions evaluated (set to 20000 in the experiments).

In the objective function, the default values for the weights were the following: $\alpha = 0.4$, $\beta = 0.3$, $\gamma = 0.3$. For SPO, both $\alpha$ and $\beta$ were set to 0.5.

### 3.2.2. Evaluation results

#### 3.2.2.1. Simulation setup

We evaluated the algorithm with three VNFFGs: the linear (Figure 11), the tree (Figure 12), and the loop (Figure 13).

![Figure 19 - Geant network topology](image)

We use the Geant network as the core topology, which contains 22 nodes and 36 links, as shown in Figure 19 [Orlo10]¹⁵.

Over this network core, we simulated a set of problem instances, by generating realistic values of user demands, server costs and latency requirements (end-to-end and first-hop). We assume that all nodes having enough resources to serve all user requests. We considered 2 distinct levels of latency requirements and of overall demands (named A and B) to create

---

¹⁵ Please note that a smaller scale testbed-based evaluation of the platform-level optimisation in the 5G-MEDIA testbeds is also planned in addition to the simulation-based evaluation described here. This will be reported in the use-case evaluation deliverables D6.2 and D6.3.
distinct levels of difficulty (where A is the easier and B the more difficult scenario). For instance, latency thresholds $L_{\text{min}}$ and $L_{\text{max}}$ in A are bigger than those in B. As a result, there is more chance for finding good solutions in A than in B.

Users and servers are randomly placed in network nodes, although the proportion of nodes that can deploy servers was considered to be 30% of the whole set. All generated instances have an average of 5 user requests per network node.

3.2.2.2. Algorithm convergence

We show in Figure 20 how the EAs converge over 250 generations, corresponding to 20000 solution evaluations in a selected set of cases which are representative of the whole set of simulations. We show the convergence of SPO and CNO\textit{platform} algorithms for the linear, the tree and the loop structures. As SPO is simpler than CNO\textit{platform} in terms of algorithm complexity, SPO has better fitness values in the first generations. However, the two algorithms quickly converge to stable fitness values in less than 100 generations. The other observation is that computational complexity increases from the linear to the tree and the loop structures. As a result, the linear structure converges in less than 50 generations while it takes around 100 generations for the loop structure to reach its point of convergence.

To benchmark CNO\textit{platform}, one would argue that we need to design an optimal formulation (e.g. using integer linear program) and compare CNO\textit{platform} result with the optimal one. Recall that CNO\textit{platform} tries to minimize the fitness function $P$ where $P \geq 0$. Therefore, we can derive the optimal fitness value which is $P_{\text{optimal}} \geq 0$. As shown in Figure 20, the fitness values in all the three structures converge to a value which is close to zero. In other word, we can see that CNO\textit{platform} algorithm performs well and is able to find near-optimal solutions.

Figure 20 - Convergence of SPO and CNO\textit{platform}: (a) linear (b) tree and (c) loop structures
3.2.2.3. CNO\textsubscript{platform} vs. service placement optimisation only

<table>
<thead>
<tr>
<th>T</th>
<th>D</th>
<th>L</th>
<th>\text{SPO}</th>
<th>\text{CNO_{platform}}</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>(P_{\text{lat}})</td>
<td>(P_{\text{cost}})</td>
</tr>
<tr>
<td>L</td>
<td>A</td>
<td>A</td>
<td>2.09</td>
<td>0.30</td>
</tr>
<tr>
<td>L</td>
<td>A</td>
<td>B</td>
<td>4.13</td>
<td>0.53</td>
</tr>
<tr>
<td>L</td>
<td>B</td>
<td>A</td>
<td>2.08</td>
<td>0.30</td>
</tr>
<tr>
<td>L</td>
<td>B</td>
<td>B</td>
<td>4.11</td>
<td>0.49</td>
</tr>
<tr>
<td>T</td>
<td>A</td>
<td>A</td>
<td>0.86</td>
<td>0.26</td>
</tr>
<tr>
<td>T</td>
<td>A</td>
<td>B</td>
<td>2.27</td>
<td>0.32</td>
</tr>
<tr>
<td>T</td>
<td>B</td>
<td>A</td>
<td>0.44</td>
<td>0.30</td>
</tr>
<tr>
<td>T</td>
<td>B</td>
<td>B</td>
<td>1.92</td>
<td>0.30</td>
</tr>
<tr>
<td>P</td>
<td>A</td>
<td>A</td>
<td>4.88</td>
<td>0.12</td>
</tr>
<tr>
<td>P</td>
<td>A</td>
<td>B</td>
<td>7.89</td>
<td>0.17</td>
</tr>
<tr>
<td>P</td>
<td>B</td>
<td>A</td>
<td>4.88</td>
<td>0.10</td>
</tr>
<tr>
<td>P</td>
<td>B</td>
<td>B</td>
<td>7.87</td>
<td>0.15</td>
</tr>
</tbody>
</table>

In this section, we show a comparison between CNO\textsubscript{platform} and the service placement optimisation only (SPO). The full results are shown in Figure 21. Column \(T\) stands for the topology - L (linear), T (tree) and P (loop). Columns \(D\) and \(L\) represent the two levels of demands and latency requirements, respectively. The other columns represent the results of the EA for the three components of the objective function. Each EA was run 5 times and the mean of the results for the best solution are shown in the table. Notice that for SPO the penalty regarding the congestion is not optimised and is shown to illustrate the values of congestion obtained when this is not taken into account.

The first conclusion we may take from the analysis of these results is that the SPO provides good results for latency and costs in the tested instances, but by not taking into account the routing in the network it causes unacceptable levels of congestion in the network, with values for \(P_{\text{cong}}\) around 2000, which means that there are several highly overloaded links in the network.

On the other hand, the CNO\textsubscript{platform} algorithm provides acceptable results for the three objectives in all instances. Of course, with more constraints on the solution space, CNO\textsubscript{platform} has, in most cases, slightly higher penalties on cost and latency, as compared to SPO. Still, these are normally not very significant, being still on a range of quite acceptable values, are largely compensated by major improvements on network congestion.

Indeed, CNO\textsubscript{platform} significantly improves on the congestion path. Note that as SPO does not consider routing, we assume that shortest path is used between two service instances. For example, in Figure 14, if using SPO the routing between \(S_1\) and \(S_2\) for the two flows will always be \([S_1 - R_1 - R_2 - S_2]\) which causes serious congestion on the link \(R_1 - R_2\). Therefore, CNO\textsubscript{platform} is far better in term of routing flexibility to overcome congested links.
3.2.2.4. \textit{CNO}_{\text{platform}} results for instances with different latency requirements

One of the observations from the results in the previous section is that the levels of latency requirements seem to affect heavily the results. We show in this section how \textit{CNO}_{\text{platform}} reacts with different latency levels. We consider one of the instances for each topology and vary the latency requirements in four levels (L1 - L4), from the hardest to the easiest. As shown in Figure 22, the cost, latency and congestion penalties reduce from the L1 to L4 scenarios. This observation is true in all the VNFFG structures. This shows some consistency in the results of the algorithm, and demonstrates it can be used over a large range of possible scenarios.

3.2.2.5. \textit{CNO}_{\text{platform}} results with different weighted parameters

One of the main advantages of \textit{CNO}_{\text{platform}} is that the weights defined for each component of the cost function provides flexibility to network operators to choose different operating points with distinct trade-offs of the objectives.

To illustrate this, we show in Figure 23 how \textit{CNO}_{\text{platform}} reacts with different weights in the objective function, for a selected instance (in this case, using the linear structure). We vary \(\alpha\), \(\beta\), and \(\gamma\) which respectively control the importance of latency (including first-hop and end-to-end latency), cost (including fixed and variable costs) and congestion in the objective function. Recall that \(\alpha + \beta + \gamma = 1\), and thus we take the default configuration and try variants where one (or two) of these weights is increased or reduced, being this compensated by the other one(s).
If we set $\alpha > \beta$, the CNO$_{\text{platform}}$ algorithm tends to reduce latency, but potentially increases the cost. For instance, when we increase $\alpha$ in the configurations $w1 - w12$ and as a result, the latency penalties tend to be reduced. A similar observation can be found for $\beta$ and $\gamma$.

However, as we are dealing with the three weights ($\alpha$, $\beta$, and $\gamma$) at the same time, changing one of them also has impact on the others. For example, the weight settings $w10$, $w11$ and $w12$ have the same $\alpha$ value, but the latency penalties are different as we change $\beta$ and $\gamma$. Finding the correlation between the three weights is important as it helps to decide suitable operating points. For example, based on these results, if one needs to minimise the cost while having congestion penalty to be less than 1, the weight setting $w12$ should be used. In summary, by changing the weights, CNO$_{\text{platform}}$ can provide an approximation of the Pareto front of the trade-off between the three objectives (latency, cost and congestion).

### 3.2.3. Conclusion

The output of the platform-level optimisation algorithm identifies how the VNFFG instances of each of the services should be deployed on the underlying computational and network resources. It selects the set of NFVI instances where VNFs should be deployed and the network resources required between the NFVI instances. These decisions are conveyed to the MANO functions to guide the instantiation of the VNFs forming each of the network services. In other words, the platform-level optimisation algorithm defines the constraints within which the use-case level optimisation algorithms operate. The use-case level algorithms manage at a finer level the application behaviour, VNF instantiation and scaling actions without violating the decisions of the overseeing platform-level system optimisation decisions. The service level optimisation algorithms per use case are described in the following subsections.

### 3.3. Media service level optimisation

Several different reinforcement learning (RL) algorithms could be used to train the CNO learning agent in 5G-MEDIA’s first two use cases. For instance, the following have previously been used in the context of streamed media content, including Asynchronous Advantage Actor-Critic (A3C) [Mnih16], Deep Q-Network (DQN) [Mnih15], REINFORCE [Sutton19], tabular Q-Learning [Wat89].

With tabular Q-learning, all possible actions (e.g. choosing a video chunk bitrate, switching between CPU/GPU) and states (i.e. observations from environments like available network capacity, loss rate, latency) are stored in a single table called a “Q-table”. Each row of the Q-table represents a separate state and each column maps to a distinct action. The key drawback of table-based algorithms is that they do not scale well when the state-action space is very large.

A typical approach to solve the scalability issue of tabular RL solutions is to decrease the state space by making some simplifications that often translate to unrealistic conditions. For example, the state of the art tabular RL algorithm for choosing the correct bitrate of video

---

\[16\] The interactions between the MAPE CNO algorithms and the MANO functions within the overall 5G-MEDIA SVP will be included as part of the final architecture as described in deliverable D2.4 [D2.4].
chunks assumes that network throughput follows a Markovian model [Chiar16]. This means that the agent does not need to hold throughput information of the past video chunks, but knowing the throughput of the last chunk could be used to choose a bit rate for the next chunk. However, this kind of assumption can make inaccurate estimations of future network conditions, and, in turn, result in the incorrect selection of chunk bit rates, especially in networks with highly dynamic conditions. This is because simple network models such as Markovian dynamics [Chiar16] may fail to capture the intricacies of real dynamic networks. For this reason, we consider an alternative approach based on a deep reinforcement learning algorithm utilising a neural network rather than explicit state-action tables [Mnih16]. A3C incorporates a large amount of history information into its state space to overcome the limitations of tabular-based methods. It has been successfully applied to several learning problems [Mao17][Jader17][Wu17]. This is our key motivation to explore options for using a deep neural network approach and to compare its performance with tabular Q-learning.

Unlike conventional approaches, such as like Deep Q-Networks (DQN), where a single agent implemented by a single neural network interacts with a single environment, A3C can be trained via parallel agents asynchronously and thus learns more efficiently and quickly. Each learning agent sees a different set of input parameters (i.e. different environments as shown in Figure 24). The agents then continually send their local information (e.g. state, action, and reward) to the central agent which aggregates all the information and updates a single learning model. This updated model will then be sent back to the agents and the learning process continues in this way. Note that the central agent populates the updated model to each agent asynchronously. The parallelism of A3C reduces the training time by several order of magnitudes compared to conventional techniques like DQN, mainly because it allows each agent to be exposed to a completely new environment (see Figure 24). Because the experience of each agent is independent of another agent, the overall learning (by central agent) becomes extremely diverse. This is also known to generate better results.
A3C runs two separate neural networks simultaneously: actor and critic networks. The main responsibility of the critic is to give feedback to the actor in order to refine the policy function of the actor network so that a better action can be selected given a particular state. Note that the critic network is only used during the training phase to improve the actor network policy (i.e. in the post-training phase only the actor network is required to execute the desired behaviour in the real environment). Figure 25 demonstrates the high-level interaction between the actor and critic networks. Both actor and critic networks receive the same set of states from the environment. The actor network generates an action ($a_t$) based on a policy function $\pi(a_t, s_t)$ which is the probability that an action ($a_t$) is taken at a particular state ($s_t$). The critic network produces the value function $V(s_t)$ which estimates (from empirically observed rewards) the expected total reward starting at state ($s_t$) for a particular policy. In this way the critic network helps the actor network to have better understanding of consequence of choosing an action at a particular state.

Note should also be taken that A3C is designed to run on a machine with multiple CPUs (i.e. one can train a A3C model reasonably quickly without having expensive GPUs).

3.3.1. Training methodology:

The most crucial part of any machine learning algorithm (both supervised or unsupervised learning techniques) is the training phase. If a machine learning algorithm is not sufficiently trained it may react to unseen conditions very poorly, making inappropriate decisions.

Unlike supervised learning algorithm that requires diverse labelled datasets, a reinforcement learning algorithm requires to be exposed to diverse environments in order to effectively learn from its mistakes before it can be usable in real environments. However, in practice, exposing an RL algorithm to a large number of conditions can be a time-consuming process, although it could be considered more accurate compared to simulated/emulated approaches. For example, in UC2, the CNO (i.e. RL agent) should explore a live video streaming environment, ideally using an actual live video streaming client to learn what compression level to employ for video chunks under a wide range of network conditions in order to improve the quality of experience (QoE) of the video receiver. However, creating such conditions requires the receiver (i.e. the broadcaster site at RTVE in this case) to continuously watch live videos, streaming from journalists under a wide variety of network conditions that introduce different
levels of congestion, packet latency and loss. Alternatively, this can be achieved more simply if the receiver watches pre-recorded videos rather than live streaming from journalists. Unfortunately, both approaches require a large number of training sessions to be set up and executed with QoE being assessed from each case - the QoE being a function of the original video, the selected compression parameters and the distortion introduced by the network as the network throughput, latency and loss will be affected by the background traffic over the path from the remote venue to the broadcaster site.

To make this process faster, it may be reasonable to train the RL agent in a simulated environment where the RL agent can be exposed to a wide range of conditions. For example, in this way, the RL agent can explore various video profiles, network conditions and compression schemes that in practice may require several weeks/months of training.

Choosing an appropriate simulating platform is also crucial. A packet level simulator like Network Simulator-3 (NS3) is an ideal choice because it can model various network environments more accurately compared to flow-level or chunk-level [Mao17] simulators. However, NS3 is slow compared to a simple chunk-level simulation used in [Mao17] that is particularly designed to model video streaming contents. It has been argued that although no simulator can capture all artefacts of real environment, if a RL algorithm is sufficiently trained, with even a simple simulator, it can operate well in real systems [Mao17]. We therefore plan to exploit a simple chunk-based simulator to train the CNO in both UCs.

3.3.2. Offline vs online learning

There are several ways to deploy an RL algorithm in real systems. The first approach is referred to as “offline learning” where the algorithm is trained in simulated/emulated environments, and then it is deployed in the real environment with no more learning. In this way, if a RL algorithm is well-trained it can perform well when deployed in the real system. A drawback of this approach is that a simulation is not identical to the real-world environment. On the one hand, a simulation will not model all aspects of a real system behaviour 100% accurately resulting in the algorithm being trained against a less realistic model. On the other hand, a simulation environment is more controlled and could result in some metrics being readily available that could be difficult to extract in a real-world system.

The alternative approach is an online one where the RL algorithm explores actions in a real-world setting and uses the reward function obtained from its actions to update its policy.

In machine learning literature this is referred to as “online learning”. A drawback of online learning is that it can be time-consuming to set up and execute a wide range of training scenarios. As mentioned in [Mao17], offline simulation-based training can experience 100 hours of online training in only 10 minutes. It also has the disadvantage that the exploration of the action-space of the RL algorithm results in real-world, rather than simulated, behaviour.

Another variation of online learning is a hybrid of the above two techniques whereby the algorithm is first trained in a simulated environment the algorithm continues to learn in the operational environment.

To evaluate these trade-offs, we decide to develop and compare the online, offline and hybrid learning methods.
3.4. Reinforcement learning in UC1: Immersive Media

Reinforcement learning (RL) is a process of learning in which a situation is mapped to an action in order to maximize either a specific numerical or abstract reward. No advice is given to the entity following the learning procedure, the learner, regarding which actions to take. Instead of this, the learner should discover the rewards that are yielded from the available actions [Sutton18]. The procedures followed by a RL algorithm have similarities to a young person growing up. Without prior knowledge, the young person, being curious, puts his/her hands on the stove and gets an aching burn. The young person realizes that a negative reward was received, hence it is highly unlikely that he/she will put his/her hands on a burning stove again.

In recent years, RL is gaining increasing popularity as partial information on the environment is enough for an algorithm to begin its training and converge to an optimal policy. Greedy or even supervised learning algorithms are limited by network peculiarities deriving from geographic and other factors. The RL approach seems more promising in the sense that the model is continuously updated according to the collected rewards. These reasons have led RL to already be applied in various works [Lolos17] [Padala14] [Barrett13], aiming for dynamic resource management, network management and elasticity in the cloud.

The framework behind most RL algorithms is a Markov Decision Process (MDP). A Markov Decision Process is composed by:

- a finite number of states $S$;
- a finite number of actions $A$;
- a transition function $T: S \times A \times S \rightarrow [0, 1]$ assigning a probability distribution over states for each state and action;
- a reward function $R: S \times A \times S \rightarrow \mathbb{R}$ emitting the immediate reward received after each transition.

In general, the goal of RL algorithm is to learn a mapping from states to actions, or policy, $\pi$. The optimal value of an action $a$ taken from the state $s$, denoted by $V^*(s)$, expresses the expected sum of discounted rewards that an agent would receive when, starting from state $s$, the agent performs an action $a$ and follows the optimal policy. The aforementioned values are connected through the following equations:

\[
Q^*(s, a) = \sum_{s' \in S} T(s, a, s') [R(s, a, s') + \gamma V^*(s')] \\
V^* = \max_{a \in A(s)} Q^*(s, a)
\]

The optimal values of the states are, therefore, the solutions to the set of equations:

\[
V^*(s) = \max_{a \in A(s)} \left( \sum_{s' \in S} T(s, a, s') [R(s, a, s') + \gamma V^*(s')] \right)
\]

The optimal policy, given the optimal values of the states is:
\[
\pi^*(s) = \arg\max_{a} \left( \sum_{\forall s', s \in S} T(s, a, s')[R(s, a, s') + \gamma V^*(s')] \right)
\]

Q-learning is employed when the above computations are not feasible or desirable. Q-learning performs a local incremental computation upon a decision. In Q-learning, estimations of the value of the state-action pairs are maintained and updated when new experience is acquired, according to the following equation:

\[
Q(s, a) \leftarrow (1 - \alpha) Q(s, a) + \alpha(r + \gamma \max_{a' \in A(s')} Q(s', a'))
\]

where the parameter \(\alpha\) is the well-known learning rate, determining how fast is the algorithm learning from new experiences, is the new state and is the reward of the transition.

### 3.4.1. State-space modelling and reward function

**State-Space.** The aforementioned optimality is achieved assuming that the overall system presents approximately the same behaviour every time it finds itself in a specific state, at least for relatively long periods of time. In order to capture the complexity of the system, the required RL model could be unrealistic in terms of needed resources and training time. On this ground, methods for the adaptive state space partitioning can be employed, aiming to the minimization of the total number of states. In this iteration of our implementations the number of states was relatively small, hence even though an algorithm which includes adaptive state-space partitioning was implemented, the choice was to define static states, leveraging statistical characteristics of the selected parameters (metrics).

**Modelling Approach.** Concerning the algorithm’s modelling, the optimal policy can be calculated following either a model-based or model-free approach. In the model-based approach, the RL agent tries to model the exact behaviour of the environment, while in the model-free approach the RL agent evaluates the effectiveness of the actions without modelling the exact behaviour of the environment.

In the model-based approach the optimal policy is calculated exploiting dynamic programming algorithms, or alternatives such as Prioritized Sweeping. Prioritized Sweeping uses careful bookkeeping to concentrate the computational effort on the most interesting parts of the system [Moore93]. A fixed amount of computation is allowed between each real-world observation, i.e., each transition from a state to another. After a specific transition, the transition probability estimate is updated along with the transition probabilities of previously observed successors. On the other hand, the most common representative of the model-free approach is the Q-learning algorithm. Q-learning is efficient from the aspect of the necessary computational and memory resources [Wat92]. Nevertheless, Q-learning performs only local updates to the values, and at each step only the value of the performed action can be updated, and as a consequence, only the policy of the initial state. In order to converge to an optimal policy, Q-learning would require a larger amount of experiences. For these reasons, the model-based approach was selected and Prioritized Sweeping was exploited as the algorithm for the calculation of the optimal policy.
**Reward Function.** The reward function is one of the most vital elements of a reinforcement learning algorithm. It should be designed carefully as it is able to assist the fast convergence of the algorithm or lead to false optimal policies for each state [Mati06]. In our CNO system the reward function is composed by five reward components aiming for optimizations on different parts of the system\(^\text{17}\). The final reward value acquired after the execution of an action is the weighted sum of the values of these reward components, which are defined as follows:

- **GPU Usage:** This reward component heavily depends on the usage of a GPU-node. If the vTranscoder VNPs are deployed on a CPU-only node, the reward will be positive as the cost of such a node is substantially lower compared to the cost of a GPU-node. In case a GPU-node is utilized and exploited by the majority of live spectators, this reward component will be positive as well. If the employed GPU-node is underutilized the reward value will be negative. Assuming \(u\) is the total GPU usage, this reward component is defined as:

\[
g(u) = \begin{cases} 
1, & u = 0 \\
1 - u, & u > 0.5 \\
1 - u, & u \\in [0, 1] 
\end{cases}
\]

- **QoE of Single Spectator:** This reward component is defined as the percentage of increment or decrement on a single spectator’s QoE after the execution of a certain action. Hence, this component’s value will be positive if a spectator’s QoE is improved or negative otherwise. Assuming \(q_{\text{init}}, q_{\text{after}}\) is a spectator’s QoE before and after an action’s execution respectively, this reward component is defined as:

\[
s(q_{\text{init}}, q_{\text{after}}) = \frac{q_{\text{after}} - q_{\text{init}}}{q_{\text{init}}}, q \in (0, 5]
\]

- **Combination of QoE & Transcoding Cost:** This reward component takes into account the sum of the spectators’ QoE and the total transcoding cost, which depends on the number and type of produced profiles, i.e., CPU-produced or GPU-produced. The value of this reward component will be positive in case the QoE sum is higher than the transcoding cost, or negative otherwise. Let \(q_s, t\) be the spectators’ QoE sum and total transcoding cost respectively. Then the reward components are defined as:

\[
c(q_s, t) = \begin{cases} 
\frac{q_s}{t}, & q_s \geq t \\
-(1 + \frac{q_s}{t}), & q_s < t > 0.5 
\end{cases}
\]

\(^{17}\) Note that the specific reward function we have proposed here is a reference example for validation and evaluation purposes, aiming at modelling the desired optimisation strategy of the reinforcement learning algorithms. Other metrics and combinations of metrics may also be considered, depending on the optimisation targets of the specific media application. These should be determined by the service provider depending on how they wish cost and performance aspects to be traded off. We have implemented a way for the reward function to be defined interactively by the service provider through the 5G-MEDIA SDK – see deliverable D5.2 [D5.2].
- **Monitoring Parameters**: This reward component depends on the percentage of increment or decrement of certain selected parameters, namely the bit rate and frame rate perceived by a single spectator, following the execution of a certain action. The reward is computed as weighted mean of the increment or decrement of both types of measurements. Let \( b_{\text{init}}, b_{\text{after}}, f_{\text{init}}, f_{\text{after}} \) be the initial and after values of bit rate and frame rate respectively. This component is then defined as:

\[
m(b_{\text{init}}, b_{\text{after}}, f_{\text{init}}, f_{\text{after}}) = \frac{1}{2} \left( \frac{b_{\text{after}} - b_{\text{init}}}{b_{\text{init}}} + \frac{f_{\text{after}} - f_{\text{init}}}{f_{\text{init}}} \right), b,f > 0
\]

- **Number of produced profiles**: This reward component depends on the number of profiles that are produced at a given moment. A positive reward is emitted in case the number of produced profiles is reduced or a negative reward in case it increases. Let \( n_{\text{init}}, n_{\text{after}} \) be the number of produced profiles before and after an action’s execution, this reward component is then defined as:

\[
p(n_{\text{init}}, n_{\text{after}}) = n_{\text{init}} - n_{\text{after}}
\]

Among the aforementioned reward components, two are focused on the performance of a single spectator (QoE of Single Spectator, Monitoring Parameters), while three are focused on the performance and cost of the overall system (GPU Usage, Combination of QoE & Transcoding Cost, Number of Produced Profiles). Adjustments of the reward component’s weights will lead the CNO to focus on specific aspects of the system while performing the necessary optimizations.

### 3.4.2. Employed monitoring parameters

The Cognitive Network Optimizer should be able to determine the overall state of the system as precisely as possible in order to be capable of suggesting and/or executing the essential corrective actions. On this ground, a combination of infrastructure, application-level and quality of experience metrics have been utilized to depict an accurate image of the system’s operation. For the metrics related to containers running on Kubernetes cluster, Prometheus has been employed and two derived metrics expressing the packet loss of the transmitted and received network packets are exported. Besides the metrics exported through Prometheus, application-level metrics are exported from both the spectators and the vTranscoder VNFs. Moreover, the Mean Opinion Score (MOS) value is computed based on various monitoring parameters depending on the selected QoE model, as explained in the relevant section. Table 11 summarizes the metrics that have been employed.
<table>
<thead>
<tr>
<th>Origin of Metric</th>
<th>Metric Name</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prometheus</td>
<td>container_network_receive_packet_loss_percentage</td>
<td>The loss of the transmitted network packets as a percentage.</td>
</tr>
<tr>
<td>(derived metrics)</td>
<td>container_network_transmit_packet_loss_percentage</td>
<td>The loss of the received network packets as a percentage.</td>
</tr>
<tr>
<td>Spectator</td>
<td>bitrate_on</td>
<td>The bit rate perceived by the spectator.</td>
</tr>
<tr>
<td></td>
<td>bitrate_aggr</td>
<td>The mean value of the bit rate perceived by the spectator.</td>
</tr>
<tr>
<td></td>
<td>framerate_on</td>
<td>The frame rate perceived by the spectator.</td>
</tr>
<tr>
<td></td>
<td>framerate_aggr</td>
<td>The mean value of the frame rate perceived by the spectator.</td>
</tr>
<tr>
<td></td>
<td>Profile</td>
<td>The profile currently consumed by the spectator.</td>
</tr>
<tr>
<td>vTranscoder</td>
<td>no_of_profiles_produced</td>
<td>The number of profiles currently produced by the vTranscoder.</td>
</tr>
<tr>
<td></td>
<td>output_data_bytes</td>
<td>The output bytes of the vTranscoder.</td>
</tr>
<tr>
<td></td>
<td>working_fps</td>
<td>The working frames per second of the vTranscoder.</td>
</tr>
<tr>
<td></td>
<td>theoretic_load_percentage</td>
<td>The theoretic load of the vTranscoder as a percentage.</td>
</tr>
<tr>
<td>QoE</td>
<td>mean_opinion_score</td>
<td>The Mean Opinion Score (MOS) value, denoting the overall quality of the transcoded content.</td>
</tr>
</tbody>
</table>

Table 11 - The monitoring parameters employed by the CNO in UC1

The monitoring parameters are collected in order to form a single feature vector, and this vector is mapped to a state according to the combination of the metrics’ values. In our initial trials the limits for each state of the metrics are statically defined, based on statistical characteristics of these metrics.
3.4.3. Supported optimization actions

The role of the CNO in Use Case 1 is to make decisions about essential optimizations and corrective actions regarding both vTranscoder VNFs and their spectators. The available optimization actions that are selected and executed by the implemented RL algorithm are summarized in Table 12. These actions can set:

a) a spectator to start the consumption of a specific transcoder profile 
   (*set_vtranscoder_client_profile*)

b) a vTranscoder VNF to start or end the production of specific profiles
   (*set_vtranscoder_profile*)

c) a vTranscoder VNF to migrate from a CPU-only to a GPU-node or vice-versa
   (*set_vtranscoder_processing_unit*)

Additionally, it is possible that the CNO will decide that there is no need for optimization (*no_operation*). It should be clear that the CNO makes decisions for an action that must be executed on a spectator; however, it is responsible for executing all the prerequisite actions as well.

<table>
<thead>
<tr>
<th>Action</th>
<th>Target</th>
<th>Supported Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>set_vtranscoder_client_profile</td>
<td>Spectator</td>
<td>[0, 1, 2, 3, 4, 5]</td>
</tr>
<tr>
<td>set_vtranscoder_profile</td>
<td>vTranscoder VNF</td>
<td>[1, 2, 3, 4, 5]</td>
</tr>
<tr>
<td>set_vtranscoder_processing_unit</td>
<td>vTranscoder VNF</td>
<td>[“cpu”, “gpu”]</td>
</tr>
<tr>
<td>no_operation</td>
<td></td>
<td>None</td>
</tr>
</tbody>
</table>

*Table 12 - Supported Optimization Actions in UC1*

3.4.4. Quality-of-Experience

Quality of Experience (QoE) is a measure of satisfaction of the user of an application or service. QoE is a holistic concept, similar to the User Experience (UX), with its roots lying in telecommunications. The definition adopted by the International Telecommunication Union (ITU) defines QoE as:

“The degree of delight or annoyance of the user of an application or service. It results from the fulfilment of his / her expectations with respect to the utility and / or enjoyment of the application or service in the light of the user’s personality and current state.”

5G-MEDIA is directly targeted to the media industry, ergo, the inclusion of a subjective or objective QoE model aiming at the Mean Opinion Score (MOS) computation, is considered essential for the enhancement of the CNO’s operation, in order to ensure it succeeds in making informed and trustworthy decisions about the corrective actions required for the improvement of the user’s quality of experience.
Mean Opinion Score (MOS) is a measure used in the QoE domain and telecommunications engineering, and it represents the overall quality of a stimulus or system. MOS is a commonly used measure for audio, video or audio-visual quality evaluation. For this iteration of the implementation, two objective QoE models were examined, i.e., a model suggested by the International Telecommunications Union (ITU) [ITU-P1070] and a model targeted to measuring gaming QoE [Zad18].

3.4.4.1. Opinion model for video-telephony applications (ITU)

According to the model developed by the ITU for video-telephony applications, perceived video quality is expressed, on a high-level, as:

$$V_q = 1 + I_{coding} \exp\left(-\frac{P_{piv}}{D_{piv}}\right)$$

where $I_{coding}$ represents the basic video quality affected by the coding distortion for a specific combination of video bit rate $Br_v$ (kbit/s) and video frame rate $Fr_v$ (fps), $D_{piv}$ is the degree of video quality robustness due to packet loss, and represents the packet loss rate.

The basic video quality affected by coding distortion is defined as:

$$I_{coding} = I_{0fr} \exp\left(-\frac{(\ln(Fr_v) - \ln(O_{fr}))^2}{2D_{Fr_v}^2}\right)$$

where $O_{fr}$, the optimal frame rate that maximizes the video quality at each video bit rate is expressed as:

$$O_{fr} = v_1 + v_2 Br_v, 1 \leq O_{fr} \leq 30, v_1, v_2 : constants$$

$I_{0fr}$ represents the maximum quality of a video for a constant bit rate and is defined as:

$$I_{0fr} = v_3 - \frac{v_5}{1 + \left(\frac{Br_v}{v_4}\right)^v_5}, 0 \leq I_{0fr} \leq 4, v_3, v_4, v_5 : constants$$

$D_{Fr_v}$ represents the degree of video quality robustness due to frame rate and is expressed as:

$$D_{Fr_v} = v_6 + v_7 Br_v, 0 < D_{Fr_v}, v_6, v_7 : constants$$

As for the packet loss robustness factor, it is expressed as:

$$D_{piv} = v_{10} + v_{11} exp\left(-\frac{Fr_v}{v_8}\right) + v_{12} exp\left(-\frac{Br_v}{v_9}\right), 0 < D_{piv}, v_8, v_9, \ldots, v_{12} : constants$$

In the relations defined above, coefficients $v_1, v_2, \ldots, v_{12}$ depend on codec type, video format, key frame interval and video display size.
Indicatively, the following figure presents the computed MOS value for a variable packet loss rate, a constant bit rate of 6.106 Mbps, a constant frame rate of 11 fps with H.264 codec type, VGA video format and a display size of 9.2 inches. Observation of the figure shows the dominance of the packet loss on the MOS value, as with a packet loss rate as minor as 0.5%, the computed MOS value falls below 2.5.

![MOS vs Packet Loss Rate](image)

**Figure 26 - Mean Opinion Score with respect to packet loss rate (ITU-T Recommendation P.1070)**

### 3.4.4.2. Gaming QoE model

The model presented in [Zad18] targets the computation of gaming QoE, which depends on the Peak-Signal-Noise-Ratio (PSNR) and frame rate (FR) of the video. It defines three components that play a role in the computation of the overall MOS, i.e., the Video Quality Model, the Reactiveness Model and the Positive Affect Model. The overall MOS value is computed according to the following equation:

\[
MOS = 1.102 + 0.59 \cdot PositiveAffect + 0.24 \cdot Reactiveness + 0.25 \cdot VideoQuality
\]

Video quality is affected by video compression in both temporal and spatial dimensions. A non-linear model has been extracted, leveraging PSNR and FR, for the Video Quality component’s computation:

\[
VideoQuality = C_1 + C_2 \cdot FR + C_3 \cdot FR + C_4 \cdot PSNR^2 + C_5 \cdot FR^2 + C_6 \cdot PSNR \cdot FR
\]

The reactiveness is affected only by the perceived frame rate of the game, has a sigmoid relation with frame rate and is defined as follows:

\[
Reactiveness = e^{0.844 + \frac{4.43}{FR}}
\]
The positive affect expresses the positive user’s experience in terms of emotions, sentiments and enjoyment and depends strongly on user factors. In this modelling, only the effect of video compression on positive affect is taken into account. The impairment from coding artefact on positive affect is estimated as follows:

\[ I_{coding} = C_1 + C_2 \cdot FR + C_3 \cdot FR + C_4 \cdot PSNR^2 + C_5 \cdot FR^2 + C_6 \cdot PSNR \cdot FR \]

This is a non-linear model, as in case of video quality, with obviously different constant coefficients. Given the \( I_{coding} \), the positive affect is computed according to the following equation:

\[ MOS_{PositiveAffect} = MOS_0 - I_{coding} \]

where \( MOS_0 \) is the MOS value in case of zero coding degradation (30 Mbps at 60 fps).

### 3.4.5. Initial training

In RL the model theoretically converges to the optimal actions for each state by receiving the rewards acquired through each action and by updating the model accordingly. In theory, when the algorithm is executed with no initial experience it will finally converge to the optimal model, yet this procedure will be time-consuming.

In order to accelerate the learning process an initial dataset has been formed. For the generation of this dataset random actions were executed for eight hours, while the necessary selected measurements were recorded before and after the execution of these actions. By the term experience, we refer to a vector of the following form:

\[ (pre\_action\_measurements, action, post\_action\_measurements) \]

From the collected experiences 4000 were randomly selected to form the initial training set. Having acquired this set of experiences, the algorithm is initially trained in a more efficient way. It should be noted that throughout the algorithm’s execution the measurements and actions are recorded in the aforementioned format so that the acquired experience is not lost in case the algorithm needs to be redeployed and the converged model can be recovered.

### 3.4.6. Algorithmic flow

To begin with, a set of monitoring metrics is collected. Metrics exported through Prometheus API (hosted in Kubernetes/FaaS VIM) and application-level metrics (Spectators, vTranscoder VNFs) are collected and the MOS value is computed. These metrics altogether form a feature vector and are fed to the algorithm. Secondly, the collected measurements are mapped to a state. A combination of metric values is unequivocally mapped to a state. In the simple version the limits representing the different levels of a specific metric are defined statically, while in the version with adaptive state-space partitioning the levels of a specific metric are defined dynamically. Then, the optimal action to be executed from this state is established. Before the suggested optimal action is indeed executed, prerequisite actions might be required. For a better grasp of the algorithmic flow, the following cases have been defined:
Case 1. The collection of metrics is mapped to an optimal state, i.e. a state where the overall system’s performance is satisfactory, along with the computed QoE value. No corrective action needs to be executed in this case, hence the algorithm determines that the optimal action is “no_operation”. The action is recorded and nothing needs to be altered.

Case 2. The system reaches a state where a spectator should consume from a different profile and the vTranscoder VNFs already produce this profile (because another spectator is already consuming). The only necessity is to command the spectator to start consuming from this profile.

Example: if the optimal action is for the spectator to switch to profile no. 2, the action “set_vtranscoder_client_profile: 2” will be executed.

Case 3. The system reaches a state where a spectator should consume from a different profile and the vTranscoder VNFs do not currently produce this profile. The vTranscoders run on CPU-only node. The requested profile has no special requirements and can be produced on a CPU-only node. The initial step in this scenario is to command the vTranscoders to start producing the requested profile. The next step is to command the spectator to start the consumption from this profile. After the execution of the aforementioned commands, the algorithm will check if there are any unconsumed profiles that are produced by the vTranscoder VNFs in order to stop the consumption.

Example: A spectator is consuming from profile no. 1 and the optimal action is to switch to profile no. 2. The vTranscoder VNFs currently produce profiles no. 0 and no. 1. The action “set_vtranscoder_profile: [0, 1, 2]” is executed and the vTranscoder VNFs continue to produce profiles no. 0 and no. 1, and they now produce profile no. 2 as well. The next command that is executed is “set_vtranscoder_client_profile: 2” and the spectator starts consuming from profile no. 2. Moreover, the algorithm finds that profile no. 1 is no longer needed and the production can stop. Hence the command “set_vtranscoder_profile: [0, 2]” is executed and the production of profile no. 1 is stopped.

Case 4. The system reaches a state where a spectator should consume from a different profile and the vTranscoder VNFs do not currently produce this profile. The vTranscoders run on CPU-only node. The requested profile has special requirements and can be produced only on a GPU-node. The first step in this scenario includes the migrations of vTranscoders to a GPU-node. Afterwards, a command is issued to the vTranscoder VNFs to start producing the requested profile X. The next step is to set the spectator to start the consumption from the profile X. After the execution of the aforementioned commands, the algorithm will check if there are any unconsumed profiles that are produced by the vTranscoder VNFs in order to stop the consumption.

Example: A spectator is consuming from profile no. 1 and the optimal action is to switch to profile no. 4. The vTranscoder VNFs currently produce profiles no. 1 and no. 2. Profile no. 4 has special requirements and can only be produced on a GPU-node, hence the vTranscoders have to be migrated. The command “set_vtranscoder_processing_unit: gpu” is executed and the vTranscoder VNFs are migrated from the CPU-only to a GPU-node. The action
“set_vtranscoder_profile: [1, 2, 4]” is executed and the vTranscoder VNFs continue to produce profiles no. 1 and no. 2, and they now produce profile no. 4 as well. The next command that is executed is “set_vtranscoder_client_profile: 4” and the spectator starts consuming from profile no. 4. Moreover, the algorithm finds that profile no. 1 is no longer necessary and the production can stop. Hence the command “set_vtranscoder_profile: [2, 4]” is executed and the production of profile no. 1 is stopped.

A little while after the optimal action’s execution, a new set of monitoring metrics will be collected, a new MOS value will be computed and the newly formed feature vector will once again be mapped to a state. Consequently, the reward is computed according to the pre-action and post-action measurements. The latest experience in the form 
(pre_action_measurements, action, post_action_measurements) is recorded and the model values are updated using Prioritized Sweeping.

3.4.7. Experimenting with reward weights

In this paragraph, a comparison of various combinations of reward components’ weights is presented, in order to demonstrate the overall operation of the implemented algorithm. In the following diagrams, the names of the scenarios are of the form

\[ w_1 \cdot w_2 \cdot w_3 \cdot w_4 \cdot w_5 \]

where the \( w_1 \) reflects the weight of the GPU usage reward component, the \( w_2 \) reflects the weight of the reward component related to a single spectator’s QoE, the \( w_3 \) defines the weight of the reward component that takes into account the overall of the spectators’ QoE and the total transcoding cost, the \( w_4 \) reflects the weight of the component that takes into account the motoring metrics and finally \( w_5 \) reflects the weight of the reward component that consider the number of profiles that are produced at the same time per vTran- scoder VNF. Each weight \( w_i \) (where \( i=1,...,5 \)) fluctuates in the range \([0.0, 1.0]\) considering that the sum of these weights must be equal to 1.

As expected, the CNO aims for optimizations of different aspects of the system according to the weights of the reward components. Figure 27 shows the statistical characteristics of the Mean Opinion Score for each of the reward weights combinations, while Figure 28 and Figure 29 demonstrate the transcoding cost and QoE-to-Rate respectively.

The setup with weights \(00_1_00_00_00\) focuses only on the maximization of the single spectator’s QoE without taking cost or other factors into account. Indeed, this version appears to emit the maximum QoE compared to all versions with a steady, high transcoding cost of 15.5, denoting that a GPU node is utilized. Due to the constantly high transcoding cost, the QoE-to-Cost rate is constantly in low levels.

On the other hand, the setup with weights \(035_01_01_01_035\) mostly focuses on the optimization of transcoding cost, in terms of GPU usage, and number of profiles only, hence it is the worst in terms of a single spectator’s QoE while achieving the lowest transcoding cost when compared to all other versions. Ergo, it achieves one of the best QoE-to-Cost ratios.
All the other scenarios lie between the optimization of QoE and Cost, hence their performances are overall more balanced between QoE and Cost.

*Figure 27 - Box plot of Mean Opinion Score for combinations of reward weights*
Figure 28 - Box plot of Transcoding Cost for combinations of reward weights

Figure 29 - Box plot of QoE-to-Cost Rate for combinations of reward weights
3.5. Reinforcement learning in UC2: Mobile Contribution, Remote and Smart Production in Broadcasting

3.5.1. Scenario 1: Optimise vCompression levels

The vCompression engine is mainly responsible for compressing and encoding the signal to a lower bitrate stream that is well-suited for transferring and distributing over the Internet. The compression levels at a vCompression VNF should be configured dynamically by CNO based on the perceived broadcaster’s quality of experience (QoE), and available computational resources and network condition at the vCompression VNF.

Adjusting an appropriate compression scheme would significantly improve the overall QoE perceived by a user (in this case a broadcaster). For example: (1). If the vCompression VNF faces some limitations in computational resources to do its job, in a limited allocated time budget it has, then it may be appropriate to allocate more vCPUs, memory and storage to the VNF dynamically (if possible); (2). If the network conditions between the vCompression VNF and the broadcaster site is deteriorated due to network congestion for instance, it may be appropriate to adopt a different compression scheme which is more suited for the present network condition.

Generally, a highly compressed video requires less network bandwidth to be streamed compared to a low compressed video. Thus, if the network bandwidth is suddenly become limited, it might be reasonable to switch to a high compression scheme if the vCompression VNF has enough computational resources. Switching to a compression scheme that does highly compress the video content may also negatively affect the quality of video, and, in turn, the QoE perceived by users. The core responsibility of CNO here is to achieve a right trade-off.

Overall, CNO aims to make a right decision to achieve the best QoE for a broadcaster based on current network condition and available computational resources that could be allocated to the vCompression VNF.

Algorithm outline

- CNO inputs
  - QoE perceived by the broadcaster. The QoE metrics could be retrieved from a special VNF located at the network edge and/or directly from a broadcaster site at RTVE.
  - Available/used computational resources of the vCompression VNF such as available CPUs, memory and storage).
  - Network conditions between the vCompression VNF and broadcaster site. This could be obtained by monitoring packet statistic in/out of virtual interfaces of the vCompression VNF, including the number of packets/bytes send, receive, drop.

- CNO actions
  - Change compression levels. This action could be done in several ways:
    - Switch between multiple vCompression VNFs. Note that each vCompression VNF has one instance of FFmpeg which supports only a particular compression level (it could be either low, high, medium or premium level). Figure 30
demonstrates this action. The switching between the vCompression VNFs is done by a load-balancer which follows the CNO commands.

- Switch between multiple instances of FFmpeg within a single VNF via an internal switcher/load-balancer. Note that each FFmpeg instance is a separate process in the vCompression VNF. Figure 31 depicts this scenario. The switching between the FFmpeg instances is done by an internal load-balancer/mechanism which follows the CNO commands.
- Instantiate a more powerful/resourceful VNF on the fly. This could be done via a FaaS technique or a scale-out operation supported by OSM. In case of the latter, OSM can dynamically add a new VNF to an already instantiated network service. When this new VNF becomes operational (i.e. ready to be used), which may take some time, CNO needs to reroute traffic from the old VNF to the new one. Similar to the other actions presented above, this could be done via a load-balancer VNF (see Figure 32).

![Figure 30](image)

*Figure 30 - CNO dynamically changes the compression level of the vCompression VNF by switching between different vCompression VNFs. Each VNF has FFmpeg running with specific compression level. We consider four different compression levels: low, medium, high and premium.*

![Figure 31](image)

*Figure 31 - CNO dynamically changes the compression level of the vCompression VNF by choosing an appropriate FFmpeg instance via an internal load-balancer.*
CNO/MAPE instantiates a new vCompression VNF on the fly (see the box with dotted line) either via a FaaS technique or a scale-out operation supported by OSM. In case of the latter, a new vCompression VNF equipped with high compression scheme is instantiated on the fly. CNO/MAPE executes this operation via OSM. Once a new VNF becomes operational, CNO reroutes the MPE traffic to it via a load-balancer VNF (see the dotted arrows).

Figure 33 shows the high-level interaction between components in the environment (right box) and the RL agent (left box) for UC2. CNO (i.e., the RL agent in this case) receives a set of inputs from the environment where some of which will be used as state inputs into the RL algorithm (see the dotted boxes at the bottom left of the RL agent). Currently, in UC2, CNO considers the following as state inputs to the neural network: (1) history of available capacity
(e.g. past eight samples); (2) history of loss rate (e.g. past eight samples) and (3) last bitrate of a video stream.

The output of the RL algorithm is an action (e.g. switching between CPU/GPU for the Speech-To-Text VNF or a compression level for a live video streaming passing through the vCompression engine VNF). The impact of the selected action is then returned to CNO from the environment through a reward function which may also consider the QoE score parameter. The QoE score could be an integer valued between 0 and 10 and is calculated at a QoE probe by a set of metrics that are mainly tailored for examining user’s quality of experience. Combining QoE metrics with other metrics, such as cost of computational resources and network related, is formed a reward function, and in turn a reward. The aim of an RL agent is to adjust the weight of neurons in its neural network in order to maximize this cumulative discounted reward.

In UC2, the RL agent is trained its neural network model offline (by our custom written simulator in Python) in which the reward function is emulated. This way once a trained offline model is deployed in a real system it is not essential for CNO to receive live feedback from the environment (i.e. CNO can operate without a QoE probe).

The reward function we have defined for UC2 has the following formula:

\[
\text{Reward} = \text{Bitrate} - \alpha \times (\text{Lossrate}) - \beta \times (\text{Smoothness})
\]

Bitrate in the above formula corresponds to the last bitrate that has been selected by CNO/RL agent; Alpha (\(\alpha\)) is a weight factor for the loss rate parameter. The higher alpha (\(\alpha\)) is, the less tolerable CNO becomes against packet losses. Currently we set the value of this parameter empirically to 500. Smoothness (\(\beta\)) ensures that CNO does not fluctuate rapidly and largely from one bitrate to another. In other words, it is considered to be negative aspect in the reward function if CNO rapidly changes the compression level of a video from a very high level to a very low level as this significantly damage user’s perceived quality of experience. If CNO predicts that there would be lack of network resources in near future it begins (in advance) decreasing the compression level at vCE smoothly. Overall, the reward function defines the objectives of the RL agent, which is maximizing the value of positive elements/parameters and minimizing the value of negative elements/parameters in the reward function.

As illustrated in Figure 33, there are several different entities in the environment which may potentially send information to CNO via a Kafka bus: vCompression engine VNF and/or Speech-To-Text VNF, network link, QoE probe VNF, and video client at RTVE (broadcast site).

The vCompression engine VNF sends information related to CPU/GPU usages, and video profile that includes a set of compression levels. The video profile basically defines a range of compression levels with the fixed upper and lower bound. CNO then ought to select a compression level within this predefined range. This range could be different for each video, and it mainly depends on the nature of the video and customer quality preferences. For example, a high-resolution high-motion video contents might be a better match with less
compressed options (i.e. high compression levels) compared to a low-resolution low-motion video content. This is due to the fact it is generally difficult to compress high-resolution high-motion videos given that they typically produce less redundant frames.

The monitoring system (e.g. Ceilometer) collects metrics such as available computational resource from the machine hosting the vCompression engine VNF (e.g. available CPU/GPU and memory usage). In reality, network related measurements such as available capacity, latency, and packet loss rate can also be collected by monitoring systems from virtual and/or real network links between the vCompression engine VNF and the QoE probe.

The QoE probe is mainly responsible for calculating the QoE score as it has been just briefly discussed. It calculates the QoE score by considering video related metrics such as bitrate, smoothness, rebuffering, compression artefact (e.g. image blockiness and blur). CNO can also consider the QoE score in its reward function to calculate a reward alongside of other parameters such as cost of computational resources. The QoE probe can be located at RTVE (broadcast site), so it can easily receive some video related measurements from the video client. It is also possible to place this VNF at the network edge. This way a 5G-MEDIA service provider ensures that it is optimising resources to achieve an expected QoE for each user between servers/compute nodes and the network edge. Alternatively, CNO can consider multiple QoE probes located at different locations. For example, one at the network edge (e.g. at a Telefónica edge), and another one at the receiver site (e.g. at RTVE). Obviously, the former approach provides better outcome compared to others as it allows CNO to have full measurement end-to-end.

Finally, the monitoring systems periodically sends relevant measurements to CNO. The interval for sending measurements dictates how fast CNO can react to changes in underlying network condition. The lower this interval is the CNO has better reaction time.

3.5.1.1. Bandwidth reservation, fairness, and billing schemes for cognitive network optimiser

There are several ways CNO can allocate bandwidth to its services/users. In this document we will explore two different schemes both of which may potentially be deployed (emulated) for UC2 in our small testbed at the Telefonica network in Madrid.

3.5.1.1.1. Scheme 1 (soft bandwidth allocation)

In this scheme CNO tries to allocate bandwidth to each service/user according to an agreed service level agreement (SLA). If an SLA of a particular service/user dictates that CNO should provide a particular set of resources (notably network bandwidth) then CNO will try to meet such requirements/constraints. With this scheme, the CNO is not required to reserve bandwidth for services/users by coordinating with underlying network devices such as existing switches/routers (both virtual and real network devices).

CNO is aware of network condition, active services/sessions and their requirements/constraints, therefore it should be able to split the available resources across all of them with objectives of providing the minimum quality of experience (QoE) of each user/service while maximising the total number of users sharing the same resources.
With this model, CNO may allocate more resources than the minimum QoE of a service/user is required when there are spare resources. An upper limit to this allocation may be followed by CNO according to a policy devised by the service provider or requested by users.

To get a better sense of this bandwidth allocation scheme, let’s explore CNO behaviour in UC2. Each video stream of a broadcaster is bundled with a quality profile which includes a set of compression levels. This implies that each quality profiles (e.g. low, standard, premium) comprises a bandwidth range with a lower and an upper bound. For example, with standard quality a user should be allocated a bandwidth range with lower bound of 15Mbps (compression level 4) and upper bound of 30Mbps (compression level 7). CNO tries to allocate bandwidth within the specific range of this profile according to underlying network conditions. Note that in this particular scenario CNO may not allocate more bandwidth than 30Mbps. Furthermore, if CNO finds itself in a difficult position to maintain the minimum QoE of its users due to lack of network (and possibly computation) resources it may follow a particular policy to reduce bandwidth of each user in a way that the QoE of users are not exceedingly damaged and the stream continuity is preserved (without any artefact). Such policies may also consider scenarios that CNO should reduce bandwidth from standard users before premium ones. We will examine various policies.

CNO may also consider a particular policy for distributing spare network bandwidth across its users based on their priorities. Note that by spare we mean a condition in which CNO have already satisfied the minimum QoE of all users/services, and yet there are spare resources (e.g. network bandwidth) to be distributed across them. In such conditions, CNO may consider allocating more resources to users with premium profile compared to users with standard and low profiles. This approach may result in a higher revenue if we assume that the premium users are more expensive than other users (more discussions about different billing schemes follows next).

A detailed discussion about billing can be found in the section 5.2 of D4.2[D4.2]. But for the sake of completeness of this section we briefly discuss possible billing approaches that can be easily designed on top of the soft bandwidth allocation scheme described earlier in this section. In short, we can categorise billing approaches in two groups: (1) Variable Billing Scheme (VBS); and (2) Fixed Billing Scheme (FBS). Variable billing scheme can be fairly easily designed based on the above bandwidth allocation model. Users will pay for a service according to the minimum QoE of that service and if the service provider did manage to provide more QoE than the minimum (up to a certain limit that can be defined by the service provider and/or user) then users will be charged accordingly. On the other hand, if the service provider fails to satisfy the minimum QoE of a service then affected users should be refunded according to the refund policy. In this way providing higher QoE to users or providing users with their minimum QoE while accommodating a greater number of users results in a high revenue for a 5G-MEDIA service provider.

With Fixed Billing Scheme (FBS), one can assume users will pay a fixed amount for each service. This way billing might be completely decoupled from the resource consumption and CNO splits available resources across users with objective of meeting their minimum requirements and providing a higher QoE (more than their minimum) whenever possible.
3.5.1.1.2. Scheme 2 (hard bandwidth allocation)

This is similar to the Scheme 1. But CNO reserves bandwidth by help of network element (e.g. underlying network switches and routers). In this way, a set of resources (both computational and network) will be allocated to a user regardless of whether that user is capable of using it or not. This approach may only be appealing for services with highly strict QoE requirements.

Given that the bandwidth is reserved by the network in this model it is less likely that the service provider fails to deliver the minimum QoE of a service/user. But yet penalty policy can still be considered in case the network itself actually fails to do so, similar to scheme 1.

Billing schemes are also similar to scheme 1, but as just discussed above the penalty due to failure of delivering the minimum QoE may less likely occur with this scheme. More detailed discussion about billing is available in the section 5.2 of the D.4.2 documents.

Note that another variant of this scheme could be thought of as a model in which CNO reserves bandwidth for the maximum QoE of a service/user. Unfortunately, with this scheme we do not need CNO, and thus we do not consider such scenarios.

<table>
<thead>
<tr>
<th>Scheme 1 (soft BW allocation)</th>
<th>Minimum required QoE</th>
<th>Maximum required QoE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Not guaranteed</td>
<td>Not guaranteed</td>
</tr>
<tr>
<td></td>
<td>(best effort to satisfy min required QoE)</td>
<td>(only when there is spare resource)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Scheme 2 (hard BW allocation)</th>
<th>Minimum required QoE</th>
<th>Maximum required QoE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Guaranteed</td>
<td>Not guaranteed</td>
</tr>
</tbody>
</table>

Table 13 – Mapping of QoE requirements to bandwidth allocation schemes

<table>
<thead>
<tr>
<th>Variable Billing Scheme (VBS)</th>
<th>Fixed Billing Scheme (FBS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scheme 1 (soft BW allocation)</td>
<td>Supported</td>
</tr>
<tr>
<td>Scheme 2 (hard BW allocation)</td>
<td>Supported</td>
</tr>
</tbody>
</table>

Table 14 – Mapping of billing type to bandwidth allocation schemes

3.5.2. Scenario 2: Mobile contribution

In this scenario journalists use a smartphone to connect to the network for live streaming video and/or audio to the broadcasting centre. Unlike other scenarios the signal is compressed and encoded on the smartphone before transmission to the network. In other words, the compression engine is located at the sender’s smartphone. Then the stream passes through a Cognitive Service function, which is possibly triggered using FaaS, where it is combined with...
additional information. The cognitive service here could be a face and/or image recognition where it tags and identifies people in the video and provides the broadcaster with supplemental information and metadata from a database for further value-added services to enhance the viewing experience.

**Scenario 2a:** in this scenario a mobile journalist connects to a network edge of the TID network to send its live content to a broadcaster site at RTVE. At first, a load-balancer at a network edge of TID forwards the journalist’s traffic to a face/image detection VNF that is located close by the load-balancer VNF (See Figure 34). Unfortunately, during live video streaming, the QoE perceived by the broadcaster site (at RTVE) is suddenly degraded due to poor network conditions between this VNF and the broadcaster site, or due to a lack of computational resources for the Face/image recognition VNF. CNO/MAPE detects this problem and instructs the load-balancer to redirect the journalist traffic to another face/image detection VNF that has good network condition, or adequate computational resources, to broadcaster’s site.

![Figure 34 - CNO instructs the LB to move the journalist traffic to another already instantiated face/image detection VNF that may be located anywhere in the TID network. The digits on the arrows depicts the order of which the actions should be taken. Note that we can think of this scenario in a Multi-NFVI use case.](image)

**Scenario 2b:** this is similar to the above scenario except that a FaaS technique is used to instantiate a new temporary face/image detection VNF. After CNO realises that rerouting the journalist traffic to a new face/image detection VNF would be essential and improve overall broadcaster’s QoE, it instructs the FaaS controller to instantiate a temporary image/face detection VNF somewhere in the TID network (see Figure 35, arrows with #1 & #2). Once this VNF becomes operational, CNO instructs the load-balancer to reroute the journalist’s traffic to this VNF (see Figure 35, arrows with #3 & #4).
Figure 35 - First CNO instructs a FaaS controller to instantiate a new temporary face/image detection VNF. It then instructs the LB to reroute the journalist traffic to the new VNF. The digits on the arrows depicts the order of which the actions should be taken.

Algorithm outline

- **CNO inputs**
  - QoE perceived by a broadcaster. This could also be retrieved by MAPE via a special QoE probe VNF located at the network edge.
  - Network conditions between the Face/Image detection VNF and broadcaster’s site.
  - Available/used computational resources of the Face/image detection VNF (e.g. vCPU, RAM, storage)

- **CNO actions**
  - Switch to another VNF with good network condition or computation resources. In this scenario, CNO should collaborate with a load-balancer to detour traffic to another VNF (See Figure 34). In case of FaaS, CNO should first instruct to the FaaS controller to get a new VNF up and running and then instructs the load-balancer to reroute the journalist traffic to this FaaS-based VNF (See Figure 35).
  - Modify computation resources (e.g. increasing the number of vCPUs)

- **CNO algorithm**
  - The main objective of the algorithm is to maximise QoE within a total cost budget (or to minimise cost without violating a minimum QoE boundary), given the constraints of the available network capacity and computational resources. Note that the cost of using/reserving computational resources may vary between the different edges of the TID network. This will be studied through:
    - A heuristic based algorithm that maps input conditions to new configuration options.
    - A reinforcement learning based approach.
3.5.3. **Scenario 3: Optimise computational resources allocated to cognitive services**

This section identifies future work to extend the current algorithms to manage the quantity of computational resources allocated to Cognitive Services VNFs. A similar approach to the reinforcement learning algorithms employed for UC1 as described in section 3.4 is envisioned.

The compressed output stream of the vCompression Engine is routed to the Speech-to-Text Engine where the audio gets analysed and a text is extracted. This text is then added as subtitles to the video stream. The final signal with subtitles can then be accessed either via a browser from the broadcaster for further in-house use or it can be offered to the public as an online service.

There are several reasons that this VNF may perform poorly, but a common one might be related to insufficient computation resource available at this VNF. To prevent such a scenario, CNO plans to monitor the performance of this VNF by considering the broadcaster observed QoE and consumption of computational resources at this VNF on real-time. CNO therefore would allocate more resources to this VNF if it sees deterioration to the QoE perceived by broadcaster (e.g. text lag, noise, etc.).

**Algorithm outline**

- **CNO Inputs**
  - QoE from users directly and/or from a specialised QoE probe VNF that can be located at the network edge
  - Computational resources of the speech-to-text VNF (e.g. vCPU, RAM, storage)
- **CNO actions**
  - Switch to a more powerful/resourceful speech-to-text VNF. In this way, CNO should collaborate with a load-balancer to detour traffic to a more powerful VNF (See Figure 36).
  - Increase computational resources of a speech-to-text VNF. (see Figure 37)
  - Instantiate a more powerful/resourceful VNF dynamically. This could be done via a FaaS technique or a scale-out operation supported by OSM. This is quite similar to the action iii of the previous scenario (vCompression).
3.6. Optimisation in Use Case 3: Ultra-high Definition over Content Distribution Networks

3.6.1. Scenario 1: Triggering of scale-out and scale-in of vCaches

The CDN NS depicted in Figure 38 is used as the basis for the evaluation of the CNO’s ML algorithms for anomaly detection and traffic forecasting. The scenario is one where anomalous traffic, a flash-crowd event, for example, causes congestion on the network between vCaches and the users, which may cause performance degradation for the delivered video, reducing QoE for the users. The solution to which is the triggering of a scale-out operation by the MAPE component to deploy additional vCaches available over non-congested network segments.
A supervised learning algorithm was developed and described in deliverable D3.3 [D3.3] to identify anomalous traffic events that are likely to cause congestion between the currently instantiated vCaches and the users. We used a fully connected neural network implemented with TensorFlow\textsuperscript{18} and Keras\textsuperscript{19}. Each training input consisted of the moving window of load values plus a label of whether there was a congestion event within a defined look-ahead window immediately following the current time. The idea is to train the ML algorithm to identify early characteristics of the traffic anomalies and to identify whether the link would be congestion within the look-ahead period. More details of the implementation and evaluations are in the deliverable D3.3 [D3.3].

One of the main challenges that may face a machine learning developer while working on anomaly detection topics is the rareness of anomaly labelled data. In this context, finding a way to train detection models using only raw data or more precisely negative data (normal cases) could be a really great solution of the above described problem. One potential approach for future work is based on the use of autoencoder neural networks [Jordan19].

In a nutshell, an autoencoder is an unsupervised machine learning algorithm that applies backpropagation, setting the target values to be equal to the inputs (i.e. no labelled dataset is required). Autoencoders are typically used to reduce the size of the inputs into a smaller representation. If one needs the original data, they can reconstruct it from the compressed data. A well-trained and well-structured autoencoder neural network model should then be able to produce its output data the same as its input data with minimal reconstruction error. Note that an autoencoder typically performs well when it receives input data for what it has been particularly trained with. For example, inputting a dog image to an autoencoder model that has been trained with cat images may result in a poor output image with a high reconstruction error.

An idea of utilizing an autoencoder in UC3 is to train a neural network with normal network traffic. Once the trained algorithm is deployed in a real system, the algorithm can detect anomalous conditions (where the model has not been trained with) by merely looking at the reconstruction error. If the reconstruction error is constantly larger than a predefined threshold (e.g. larger than 2 standard deviation from the mean), an anomalous condition is

\textsuperscript{18} https://www.tensorflow.org/

\textsuperscript{19} https://keras.io/
suspected, and then an alarm is raised. It is obvious that choosing an appropriate threshold is not trivial. If the threshold value is too large, anomalous condition may appear as normal traffic; if it is too small, normal traffic may appear as anomalous.

To make sure the raised alarm is valid and it is related to the actual network congestion a regression analysis could be conducted within the CNO by considering other available metrics, such as available capacity, and traffic demand on a vCache VNF, before a scale-out operation is executed by CNO/MAPE.

The simplest architecture of an autoencoder is a feedforward, non-recurrent neural network with an input layer, an output layer and one or more hidden layer connecting them. Figure 39 shows a simple fully connected feedforward neural network. There are 4 neurons in each layer (input, hidden and output). Using lower number of neurons at the hidden layer causes higher compression of the input data which may result in higher reconstruction error. Therefore, in order to minimize reconstruction error due to compression at the hidden layer, it is desirable to use the same number of neurons in this layer as the input layer. Finally, because an autoencoder should reconstruct the input data, it should have the same number of neurons in its output layer as in its input layer.

At the time of writing the use of autoencoders in the project’s prototypes remains an aspect of future study.
3.6.1.1. Deep-learning algorithm for triggering scale-out actions

In this subsection we describe how the supervised learning algorithms for anomaly detection on network traffic levels we described in D3.3 were expanded to include computational metrics on the performance of the vCache VNFs.

Deep Learning is performed utilizing different variation of gradient descent. Initially the first order derivative of the loss function is calculated and then the weight is updated accordingly. This process is called back-propagation and is one of the basic features of various deep learning frameworks, as back-propagation is a method of calculating derivatives accurately and efficiently in large systems composed by elementary subsystems or calculations, represented by known, differentiable functions [Werb90]. Combined with other optimization algorithms, deep learning transforms the weights of the neural networks during training producing a model of knowledge learned from training data.

For the purposes of Use Case 3 a Deep Learning library in Scala, DeepLearning.scala\(^{20}\) was employed. DeepLearning.scala is a deep learning toolkit for Scala, combining object-oriented and functional programming constructs in order to create statically typed dynamic neural networks from map/reduce and other higher order functions [DLScala]. Utilizing this library, we trained a simple neural network in order to label the CDN’s overall operation as regular or irregular and determine whether the CDN is underutilized. Regular implies that the streamed content is satisfactory without visual artefacts, while irregular implies the existence of visual artefacts, possibly subject to network congestion.

This library implements Automatic Differentiation [Bayd18] with attributes that do not appear in other frameworks. Until this framework made its appearance, the state-of-the-art deep learning frameworks followed two fundamental approaches, Define-and-Run and Define-by-Run:

- Define-and-Run: allows users to create computational graphs (immutable Abstract Syntax Trees) of object languages that can be evaluated by the framework runtime. Computational graphs can be scheduled to multiple devices, yet the object languages have bad interoperability with the meta language.
- Define-by-Run: is able to execute forward pass calculation in user written code and generate the internal states for running a backward pass. Define-By-Run frameworks have good interoperability with the hosting language, therefore, control flows and native function calls can be easily used during the execution of neural networks.

The DeepLearning.scala framework introduced the mechanism of Monadic Deep Learning. The Neural Networks in this framework are immutable as in Define-And-Run frameworks, while being interoperable with Scala as in Define-By-Run frameworks. With the assistance of DeepLearning.scala complex neural networks can be created from simple code. The code written in this framework is differentiable, containing trainable variables that learn the knowledge.

3.6.1.1.1. Employed metrics

The Cognitive Network Optimizer should be able to determine the state of the system, specifically, the regularity or irregularity of the vCDN’s operation and suggest/execute the necessary corrective actions targeting to the improvement of the overall system’s performance. Therefore, a combination of infrastructure and application-level metrics have been employed.

In UC3 the employed infrastructure is OpenStack NFVI, hence the selected metrics with origins in the infrastructure are exported through Ceilometer and include resources consumption and a couple of network-related metrics. Moreover, the vCache VNFs export application-level metrics such as the hit rate of the caches, the number of connected clients and the rates of received and sent bytes. Furthermore, the media server (Plex Server21) exports metrics related to the total number of clients streaming content, the number of active streams, the total bandwidth et cetera. The selected metrics that form a feature vector in our implementations as well as their origins are presented in the following table.

<table>
<thead>
<tr>
<th>Origin of Metric</th>
<th>Metric Name</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenStack NFVI</td>
<td>cpu_util</td>
<td>The CPU’s utilization as a percentage.</td>
</tr>
<tr>
<td></td>
<td>memory_util</td>
<td>The memory’s utilization as a percentage.</td>
</tr>
<tr>
<td></td>
<td>network.incoming.packets.rate</td>
<td>The rate of incoming network packets.</td>
</tr>
<tr>
<td></td>
<td>network.outgoing.packets.rate</td>
<td>The rate of outgoing network packets.</td>
</tr>
<tr>
<td>vCache VNF</td>
<td>hostdb.cache.total_hits_rate</td>
<td>The rate of cache hits (derived from successive measurements of cache hits).</td>
</tr>
<tr>
<td></td>
<td>http.current_client_connections</td>
<td>The number of clients connected on the vCache.</td>
</tr>
<tr>
<td></td>
<td>bytes_sent_rate</td>
<td>The rate of sent bytes (derived from successive measurements of sent bytes).</td>
</tr>
<tr>
<td>Metric</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>bytes_recv_rate</td>
<td>The rate of received bytes (derived from successive measurements of received bytes).</td>
<td></td>
</tr>
<tr>
<td>packets_sent_rate</td>
<td>The rate of sent packets (derived from successive measurements of sent packets)</td>
<td></td>
</tr>
<tr>
<td>packets_recv_rate</td>
<td>The rate of received packets (derived from successive measurements of received packets).</td>
<td></td>
</tr>
<tr>
<td>response_data_stream_count_direct_play</td>
<td>The number of direct play streams.</td>
<td></td>
</tr>
<tr>
<td>response_data_stream_count_direct_stream</td>
<td>The number of direct streams.</td>
<td></td>
</tr>
<tr>
<td>response_data_stream_count_transcode</td>
<td>The number of transcoding streams.</td>
<td></td>
</tr>
<tr>
<td>response_data_stream_count</td>
<td>The total number of streams.</td>
<td></td>
</tr>
<tr>
<td>response_data_lan_bandwidth</td>
<td>The LAN bandwidth.</td>
<td></td>
</tr>
<tr>
<td>response_data_wan_bandwidth</td>
<td>The WAN bandwidth.</td>
<td></td>
</tr>
<tr>
<td>response_data_total_bandwidth</td>
<td>The total bandwidth.</td>
<td></td>
</tr>
</tbody>
</table>

It should be noted that from the aforementioned metrics, the ones expressing rates (e.g. bytes_sent_rate, packets_sent_rate) were not available out-of-the-box. Instead they were computed manually based on their successive values.

### 3.6.1.1.2. Available actions and execution flows

In this iteration of the CNO implementation for UC3 the supported actions include the scaling-out and scaling-in of an edge vCache VNF of the vCDN service.
<table>
<thead>
<tr>
<th>Action</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>vnf_scale_out</td>
<td>The command suggests that the Edge vCache should be scaled out and another Virtual Deployment Unit (VDU) should be added to the edge cache VNF.</td>
</tr>
<tr>
<td>vnf_scale_in</td>
<td>The command suggests that the Edge vCache should be scaled in, hence a VDU will be removed from the edge cache VNF.</td>
</tr>
<tr>
<td>no_operation</td>
<td>No action needs to be executed.</td>
</tr>
</tbody>
</table>

**Table 16 - Available CNO actions**

Regarding the operation of the implemented algorithm, the following five cases have been identified:

- **Case 1:** The vCDN’s operation is labelled as “regular” or “underutilized” and the edge vCache has not been scaled out. In this case, no corrective action is required and the “no_operation” command is issued.

- **Case 2:** The vCDN’s operation is labelled as “regular” and the edge vCache has been scaled out. Once again, no corrective action is required and a “no_operation” command is issued.

- **Case 3:** The vCDN’s operation is labelled as “irregular” and the edge vCache has not been scaled out. In this scenario, the action that should be executed is “vnf_scale_out” and a VDU will be added to the existing edge cache VNF. The scaling-out operation should have been completed.

- **Case 4:** The vCDN’s operation is labelled as “underutilized” and the edge vCache has been scaled out. The action that should be executed in this scenario is “vnf_scale_in” and a VDU will be removed from the existing edge cache VNF.

- **Case 5:** The vCDN’s operation is labelled as “irregular” and the edge vCache has been scaled out. In this scenario the action that should be executed is “vnf_scale_out”. If the current number of running VDUs in the edge vCache is lower than the allowed total number of VDUs as defined in the vCDN Network Service Descriptor (NSD) then a VDU is added to the edge cache VNF.

These cases cover all the possible situations that may occur during the execution of the CNO application.

3.6.1.1.3. Construction of the training dataset and training of the neural network

No existing dataset could be found for the training of the utilized neural network. Ergo, a need for the creation of a custom dataset emerged. The idea behind the creation of this dataset was to recreate the “regular”, “irregular” and “underutilized” states of the system and perform manual classifications by observing the quality of the streamed content while recording the collection of the aforementioned selected measurements.

For the introduction of background traffic iPerf [Werbos90] was utilized. The tests that were executed for the collection of measurements included a variable number of connected clients and variable values of background traffic. During the conducted tests, a tester visually
classified the delivered content as regular or irregular, recording the timestamps of the transitions so that the recorded feature vectors could be labelled and fed to the neural network for training purposes. As for the “underutilized” state of the vCDN, a small number of clients was connected with low background traffic and a variable number of VDUs in the edge cache VNF. After the collection of the aforementioned dataset, a normalization of the collected data followed, a procedure considered essential for the convergence and correct operation of the neural network.

The aforementioned procedures led to the construction of a dataset of manually labelled feature vectors. The acquisition of this dataset was followed by the training of the neural network. We experimented with a variety of learning rates, initial weight and number of iterations (epochs). When the neural network’s accuracy was considered satisfying the neural network was frozen, i.e. the optimal weights were recorded and the algorithm was considered ready for deployment.

3.6.1.1.4. Application architecture and interactions

CNO, in this iteration of the implementations, consists of two components, the Metric Collector and the Classifier.

The role of the Metric Collector component (Pre-processing service) is to collect, synchronize and pre-process the collected measurements. The Metric Collector interacts with the Kafka bus, subscribes and receives measurements from two topics; from ns.instances.trans, the Metric Collector receives application-level measurements, originated in the vCache VNFs, as well as infrastructure-level measurements exported by Ceilometer, while in the app.originserver.metrics topic, the metrics exported from the Plex server are published. When a complete set of measurements is collected the normalization procedure follows and a formatted feature vector is published to the ns.instances.prep topic (in which the pre-processed measurements are published in a format proper for the Classifier).

The role of the Classifier is to classify the overall system’s operation as regular, irregular or underutilized. DeepLearning.scala is employed, for the implementation of a neural network capable of making the aforementioned classifications. The Classifier component subscribes to the ns.instances.prep topic of the Kafka bus from which it receives feature vectors in a format proper for feeding them to the neural network. When a decision is reached the outcome is published to the ns.instances.plan topic for further processing.
3.6.1.1.5. Algorithmic flow

To begin with, a set of monitoring metrics is collected, including metrics exported from the OpenStack infrastructure through Ceilometer, metrics exported from the media server regarding the number of clients, number of active streams, total bandwidth etc., and finally application-level metrics from the mid and edge vCache VNFs. The collected metrics compose a feature vector and are given as input to the trained neural network, after a normalization of the data is performed.

The neural network will then classify the feature vector as regular, irregular or underutilized, in the sense that the streamed content might be regular or irregular, or the deployed vCDN service is underutilized. If the neural network suggests that the vCDN service is underutilized, then either no action or a scaling-in action will be executed in case the edge vCache had previously been scaled-out. If the neural network classifies the system’s operation as regular then no action is performed. Finally, if the neural network classifies the system’s operation as irregular, then a scaling-out action will be triggered and executed, but only in case the number of currently running VDUs in the edge vCache is lower than the maximum allowed VDUs, as defined in the NS descriptor of the vCDN service. If a scaling-out operation is triggered the algorithm will wait until the additional VDU is instantiated before allowing any further scaling-out operations.

3.6.1.1.6. Future extensions for load balancer optimisation

This subsection identifies future extensions of the CNO in UC3 to optimise the load balancer functionality. The details of the algorithm extensions will be reported in deliverable D6.4 following further evaluation in the project testbeds.

Assuming that we have a set of available vCaches, our goal is to configure the load balancer to decide which users to be served by which vCaches. State-of-the-art load balancing algorithms such as round robin, weighted round robin, least traffic, least latency, etc. can be used. In this work, we are investigating an advance machine learning based approach for the load balancer. The idea is to predict future traffic load on a link which has impact on LB decision. We show a simple example in Figure 41.

![Figure 41: Load balancer with ML](image)

The three users are served by vCache1 and vCache2 as shown in Figure 41. Because of serving two users, vCache1 has heavier load than vCache2. Assume that a new user - user 4 would like to join the system, then the LB needs to determine which vCache should serve the user 4. If
using least traffic policy, the vCache2 will serve user 4. However, our load balancing algorithm uses advance ML to predict future traffic load on the outgoing link of vCache1 and vCache2. Let’s assume that the ML estimates the traffic load on vCache2 will be surged soon (possibly because of the background traffic), thus the LB will decide vCache1 to serve the user 4.

3.6.2. Scenario 2: Dynamic instantiation of serverless vTranscoders

Figure 42 depicts a vCDN service composed by a hierarchy of vCaches plus load balancing and probing functions. The OriginServer is deployed outside the NS scope and it’s coupled with a vTranscoder for offering a set of pre-encoded high-quality media contents. The end-users are streaming the media contents via the vCaches hierarchy and according to the current configuration of the vLoadBalancer, while the vProbe function [D4.2] is evaluating the QoE for each streaming session, providing a score.

When anomalous traffic (flash-crowd event) causes congestion on the users’ network, the QoE decreases and the vProbe produces a bad score for the current ongoing streaming sessions (Figure 43).

As a solution for re-establishing a proper QoE score for the end users, the MAPE is triggering the spawning of a FaaS vTranscoder (Figure 44). The scope of the new function is to produce a tailored encoding of the media content for one user, producing a new encoded version, in order to obtain also a better consumption of the available bandwidth in the congested link. The FaaS vTranscoder will live the time needed for completing the transcoding of the media content related to a specific streaming session, then will be automatically removed according to FaaS Life Cycle Management (LCM). In this scenario the vTranscoder will retrieve the
“original” content from the OriginServer and the new transcoded content will be made available at the edge vCache (Figure 45).

- CNO inputs:
  - Current and historical link load. The link load includes video, background and anomaly traffic.
  - vProbe score per streaming session.

- CNO outputs:
  - Based on the load, ML will predict if anomalous traffic will be in the near future or not. If it is the case, MAPE will trigger the spawning of a FaaS vTranscoder.
  - The vTranscoder will be configured for encoding a requested media content for a target streaming session

This scenario can be potentially extended in the following ways:

1) A new edge vCache can be scaled out (as described in section 3.6.1) and coupled with the vTranscoder.
2) We can think about optimizing two different vCDN services (attached to the same OriginServer and to the same users’ link) running in parallel and serving two different group of users. In this case, the vProbe is producing an overall score for the ongoing streaming sessions per vCDN service. When the users’ link is congested, both groups of users are experiencing a bad QoE, then the MAPE decides to spawn a FaaS...
vTranscoder for encoding differently the media content for one of the two groups. This way the overall bandwidth consumption in the network were the vCDN services are deployed will be optimized and the result will be a better overall QoE for both groups of users.

3.6.3. Scenario 3: joint optimisation of load balancing and video encodings at vCaches

This scenario is possible future expansion of the algorithms in scenario 2 considering the positioning of content encodings throughout the vCache hierarchy. We assume that there is a predefined set of content encodings (e.g. UHD, Full HD, HD, etc.) and depending on computational resources (e.g. memory, CPU), each vCache is capable of serving a subset of those encoded videos. Each user will specify its QoS requirements, e.g. at least Full HD video, latency should be less than 100ms, etc.

An integer linear programming (ILP) model can be defined to find the optimal solution. In particular, the solution needs to answer:

- Which encoded videos will be stored in each vCache (to check if the vCache allows to cache the encoded video we want to).
- Which users will be served by each vCache to satisfy user QoS requirement.

Figure 46 shows a sample solution produced by the ILP model.

![Figure 46 - Configuration of load balancer and vCaches](image)

Algorithm outline

- CNO inputs:
  - Current and historical link load.
  - Available computational resources at vCaches.
  - QoS requirements at users, e.g. minimum encoding resolution, maximum latency.
- CNO outputs:
  - Based on the predicted future load, the available computational resources (CPU and memory) and user QoS requirements, the algorithm will find a suitable solution. One example output of the algorithm is shown in Figure 46:
There are three predefined encoding resolutions: HD, Full HD and UHD.

There are three vCaches and four users in the system. Next to each user in Figure 46 is the minimum required encoding resolution. We may have latency requirement if we have multi-NFVI testbed environment where vCaches are in different locations.

Based on available computational resource at vCaches and QoS user requirement, the output of the algorithm decides:

- vCache1 and vCache3 only store Full HD and UHD respectively.
- vCache2 stores two encoded video: HD and full HD.
- Which vCaches to serve which users are shown in Figure 46.

Traffic prediction in scenario 2 can be used. In addition, we will use integer linear programming (ILP) to find optimal placement of the encoded videos in vCaches and the optimal assignment of users to the suitable vCaches (an example of the output of the ILP is shown Figure 47).

**Figure 47 - Integer linear program (ILP) model**

\[
\text{obj: min}\left[\sum_{i \in Z} \sum_{v \in V} M^i_v x^i_v\right] \quad (1)
\]

\[
\sum_{v \in V} \sum_{i \in Z} c^i_{uv} = 1 \quad \forall u \in U \quad (2)
\]

\[
\sum_{u \in U} \sum_{i \in Z} D^i_u c^i_{uv} \leq C_v \quad \forall v \in V \quad (3)
\]

\[
x^i_v \geq c^i_{uv} \quad \forall u \in U, i \in Z, v \in V \quad (4)
\]

\[
c^i_{uv} \in \{0, 1\}, x^i_v \in \{0, 1\} \quad \forall u \in U, i \in Z, v \in V \quad (5)
\]

We define \( i \in Z \) as a predefined set of encoded video, for example \( Z = \{\text{HD}, \text{FullHD}, \text{UHD}\} \). We use binary variable \( x^i_v \) to indicate if the encoded video \( i \) is stored on the vCache \( v \) or not and \( M^i_v \) is the associated cost of placing the video \( i \) in the vCache \( v \). On the other hand, we use binary variable \( c^i_{uv} \) to decide if user \( u \) connect to the vCache \( v \) to get the video \( i \) or not. The ILP can be explained as follows:

- The objective function (1) minimises the cost of placing the encoded video on vCaches. Note that we can change the objective function to add other metrics we are interested in to minimise or to maximise.
- Constraint (2) is used to ensure one user can connect to only one vCache to get the required encoded video.
- Given resource constraints at the vCache (3), we can place certain encoded videos at a vCache (we use \( D^i_v \) as resource required to have the video \( i \) for the user \( u \)).
- Constraint (4) is used to decide whether or not encoded video \( i \) is stored at vCache \( v \).
4. Serverless framework within the 5G-MEDIA SVP

4.1. Serverless-based VNFs and ETSI MANO

In D3.2, we presented a specification and implementation of a serverless Virtual Infrastructure Manager (VIM) aligned with the ETSI MANO approach. We implemented the proposed Serverless VIM as a plugin to OSM R5 using Apache OpenWhisk and Kubernetes (K8s) as serverless NFVI.

Initial use cases (basic UC1) tackled via our serverless VIM approach have been characterized by instantiating a set of serverless VNFs as a self-contained session meaning that all VNFs of the session are instantiated and de-instantiated together.

As our integration experience demonstrated, for this type of the session-based workloads, Serverless VIM was an adequate and cost-efficient solution.

As the 5G-MEDIA project progressed, and our serverless framework matured, we started considering more complicated scenarios (e.g. full UC1 implementation, UC2.b scenario and UC3). In these scenarios, serverless VNFs introduce a new set of problems vis-à-vis ETSI MANO and OSM.

While in the initial implementation of UC1, the Game Server (i.e., the game portal part of the application) configured vTranscoders to communicate with each other for gem bouts (i.e., served as a service orchestrator for Day 1 and Day 2 configuration), in UC2.b, making a mobile journalist application responsible for service orchestration is not a reasonable option. A service comprising up to three cognitive functions at the edge need to configure itself transparently to the mobile journalist application and present a point of presence where the mobile application can stream the captured media. This should happen dynamically in response to the mobile journalist request which is not anticipated in advance and therefore is treated as an event.

An interesting requirement in the UC2.b scenario is that (depending on real time features selection by a mobile journalist when she connects to the application to initiate a mobile contribution session for a new mobile contribution) either no serverless cognitive services in the edge will be required or just speech to text serverless VNF will be applied to the raw video stream, or just face detection serverless VNF will be applied, or both. In addition, based on the journalist selection at run time, each new mobile production can be streamed to one of the three environments: an environment local to the edge, a remote live broadcaster environment at the headquarters or a remote safe environment of the broadcaster. We discuss this Use Case in section 3.5.2 and the final design will be documented in deliverable D6.3. This scenario requires non-trivial network service instantiation workflow and Day 1 and Day 2 configuration which are not supported out of the box by OSM\(^\text{22}\).

\(^{22}\) It should be noted that one potential solution to UC2.b project would be to have every cognitive service as a separate on-boarded service and then chain services together. However, first, dynamic service chaining is not supported in OSM and this would preclude changing feature selection (i.e., cognitive functions selection) in the application throughout the session life time exactly as inability to start individual VNFs would preclude that.
While UC2.b, UC1 (full scenario), and UC3 (full scenario) appear different, what is common in all these use cases w.r.t. the serverless approach is that VNFs must be instantiated on event-driven basis after the service has been already instantiated (from the OSM point of view) and configured with Day 0, Day 1, and Day 2 configuration operations. To exemplify, consider Use Case 1, which is schematically presented in a vReplay VNF of UC1 that is instantiated on-demand when in-application event, such as one player scoring points against another one is recorded. The vReplay function will be instantiated upon this event use the buffered media to produce a clip for the spectators. This simple sub-scenario requires instantiating individual VNF and connecting it on-demand to the rest of the service.

While this sub-scenario can still be considered relatively easy (albeit not being supported by OSM), because it only requires instantiating an individual VNFs (vReplay) with an appropriate Day 0 configuration and Day 1 configuration (e.g. pointing the vReplay VNF to a vBuffer VNF that buffers media in memory), more complex sub-scenario exist.

For example, a spectator joins in an edge, where she is a first spectator of a game. This requires instantiating a VNF called vBroker at that edge. This VNF must be configured with the endpoint of the main vBroker that is used by the two gamers and if that vBroker does not support QoE requested by this new spectator, a new vTranscoder should be started for that QoE and a new topic in the game session vBroker should be created, into which the media frames with this QoE will be published. In other, words, not only individual VNF instantiation with Day 0 and Day 1 is required, but also Day 2 configuration to reconfigure the running service is a must.

In the standard ETSI MANO orchestration flows, network services are instantiated as a VNF package and then configured using charms\textsuperscript{23}. In OSM the VNFs packaged into NSD are

Second, requiring a network service developer always to partition serverless part of the service into elementary services and on-board and manage their lifecycle separately as the only means of service development, management, and orchestration would be too restrictive. The problem of dynamicity poised by the serverless approach has to be solved in general terms.

\textsuperscript{23} We evaluated the Juju charms mechanism in D3.2 and found it poorly suited for serverless container configuration.
instantiated once and in a fixed order of VNFD appearance in the NSD. This implementation complicates individual VNF instantiation required in our UCs. Moreover, such flows are not supported in a standard way in ETSI MANO. We term these orchestration flows *post-instantiation flows* in the rest of this section.

While there exist fields in VNF and VDU descriptors allowing to specify 0 replicas for a minimum size of an autoscaling group for a specific VNF/VDU, the implementation of an autoscaling group assumed by the standard is via introducing a Load Balancer VDU as part of the VNF, so that this Load Balancer would perform as a front-end of the VDUs whose number can vary between the minimal and maximum number specified in the VDU descriptor.

Initially, we envisioned to represent serverless VNFs as autoscaling group of VDUs with cardinality ranging from 0 to N. However, after a thorough examination, we found that this would be incompatible with the spirit of the serverless approach, difficult to integrate with OSM and confuse concepts clearly defined by ETSI.

First, serverless VNFs rely on the built-in Load Balancer of the serverless framework that serves as NFVI. Adding another level of Load Balancing at the VNF level would be cost-inefficient, because the Load Balancer will have to be executed even in absence of an actual workload (which is exactly the problem that the serverless approach aims at remedying). Second, representing serverless VNF as an auto-scaling group comprised of 0 to N VDUs, where each VDU is, in fact, a serverless VNF, would abuse the ETSI MANO concepts and notation. Third, OSM does not support instantiation of a VNF with zero number of VDUs. Fourth, OSM does not support instantiation of a single individual VNF or a single VDU belonging to a VNF.

**Considering these factors, we decided to develop a bolder approach to serverless integration and extend the concept of serverless VIM to serverless Virtual Network Function Manager (VNFM), and serverless Service Orchestration (SO) – resulting in a full serverless management and orchestration stack in 5G-MEDIA SVP as depicted in Figure 49.** This stack is interoperable with OSM and complements its functionality allowing to unleash the full potential of a serverless computing paradigm for media applications.
We reckon that the problems that we are facing in our use cases are of general nature and require a more flexible approach to management and orchestration workflows than the one that is currently allowed by ETSI MANO and OSM. More specifically, we need to instantiate VNFs not necessarily in the order of their appearance in NSD (i.e., in the VNFD package), but at arbitrary discrete points in time (e.g. event driven) and reconfigure the already instantiated service to incorporate these new VNF instances (as well as accommodate for their termination at arbitrary points in time throughout the service instance lifetime).

We explore utilizing K8s native workflow manager Argo\textsuperscript{24} to execute the post-instantiation orchestration flows involving serverless VNFs. Argo's expressiveness is very high. It takes away all the boilerplate code involved with executing the orchestration flows allowing us to concentrate on the business logic of service orchestration. We apply this mechanism consistently across all three use cases. More details on our approach are provided in Subsection 4.3.

4.2. Generalization of the Problem

It is easy to observe that the problem described in the previous section generalizes to other scenarios involving dynamic changing of Virtual Network Forwarding Graph (VNFG). Indeed, dynamically instantiating a serverless VNF is tantamount to adding a Virtual Link to the network service instance topology. Similarly, in multi-NFVI scenarios (e.g. UC3), some VNFs of comprising a network service can be VM based, yet some other can be serverless.

For example, in UC3, virtual transcoders employed at the edge in the form of VMs. When demand fluctuates significantly, some transcoders can be instantiated as serverless VNFs to

\textsuperscript{24} Argo, Open Source Kubernetes Native Workflows, Events, CI and CD, \url{https://argoproj.github.io/}
mitigate these transient demand bursts cost-efficiently (see section 3.6.2). Post-instantiation orchestration flows should be supported as described in the previous section to facilitate this mode of operation.

Note that similar post-instantiation flows would be required if an individual VNF is added to the network instance even if this VNF is VM based. The standard ETSI MANO flows only support post-instantiation flows related to scaling. However, other use cases arise in 5G-MEDIA SVP. For example, a VNF can be instantiated in multiple NFVIs (e.g. edges) because of the CNO decision to embed a network instance topology into the physical network topology in a certain way to improve QoE for the maximum number of services.

Therefore, we believe that the proposed mechanism is extensible beyond the serverless framework per se. We will discuss this in the final deliverable D2.4.

4.3. From Serverless VIM to Serverless VNFM and SO

4.3.1. Argo Project

Argo project comprises Argo Workflows\textsuperscript{25}, Argo Events\textsuperscript{26}, and Argo CD\textsuperscript{27}. Argo is a CNCF project. It is an open source container-native workflow engine for orchestrating jobs on K8s. Argo Workflows is implemented as a K8s CRD (Custom Resource Definition). Argo Workflows allows to define workflows where each step in the workflow is a container.

Argo Workflows models multi-step workflows as a sequence of tasks and allows capturing dependencies between tasks using a directed acyclic graph (DAG). Argo Workflows is designed from the ground up for containers in K8s. Its primary goal is to orchestrate highly parallel jobs on K8s. Argo is cloud-agnostic and can run on any K8s cluster making the workflows defined with Argo workflows portable from Day 1, an important feature for 5G-MEDIA SVP.

Although not available as an out of the box feature, Argo has been demonstrated recently to run efficiently across multiple K8s clusters\textsuperscript{28}. This would facilitate complex multi-NFVI scenarios, in which VNFs comprising a network service are deployed in a physically distributed NFVIs. Argo Workflows support a rich set of features: DAG or Steps based declaration of workflows, Artifact support (S3, Artifactory, HTTP, Git, raw), Step level input and outputs (artefacts/parameters), Loops, Conditional Branching, Parameterization, Timeouts (step and workflow level), Retry (step and workflow level), Resubmit from the last memorized state, Suspend and Resume, Cancellation, K8s resource orchestration, Exit hooks (on notifications with clean-up), Garbage Collection, Scheduling (affinity, node selectors, etc.), Volumes attachment (ephemeral or persistent), Daemoned steps (e.g. starting a service that will run throughout the whole workflow duration), Script steps, Sidecars, CI/CD, Parallelism limits enforcement, and Docker in Docker (DinD).

\textsuperscript{25} https://argoproj.github.io/argo
\textsuperscript{26} https://argoproj.github.io/projects/argo-events
\textsuperscript{27} https://argoproj.github.io/projects/argo-cd
\textsuperscript{28} https://admiralty.io/blog/running-argo-workflows-across-multiple-kubernetes-clusters/
Table 17 shows a simple example of an Argo Workflow. Argo defines a new K8s resource, *Workflow*, and the definition is at the YAML level, aligned with the K8s resource definition style. This YAML definition is validated by Argo CLI and applied against the K8s API Gateway (as any resource definition in K8s). Argo workflow controller reacts on appearance of the new Workflow object in the K8s etcd database, as well as on every event pertaining to the Workflow object. Argo Workflow object reflects the state of the workflow. Argo workflow controller can be deployed in a variety of configurations. For example, several controllers can be deployed simultaneously. In that case they will automatically partition the work (workflows) among themselves. In case an Argo workflow controller fails, another Argo controller will pick up the workflow from the consistent state in etcd. Argo workflow controller ensures eventual consistency.

In the example of Table 17, the workflow does little more than printing a message. However, any executable *template*, which is eventually a container can be specified as Argo *step*. Also, script steps are possible. In this case, Argo would automatically create a container that would run the script.

Argo Events defines two new custom resources (CRDs): *Gateway* and *Sensor*. Sensor is a resource, that can trigger any K8s resource either native or CRD in response to an event passed to it by its corresponding Gateway. This includes an ability to trigger Workflows conditionally, e.g. based on the content of a payload of an event passed to it by the Gateway.

The Sensor and Gateway are defined via YAML files (as always in K8s). The Gateway and Sensor are defined as a pair and their corresponding YAML files are applied to K8s API Gateway and controlled by the Argo Sensor Controller and Gateway Controller respectively.

Gateways might be of different types: e.g. Kafka, WebHook, etc. In our prototypical implementation we currently use a WebHook Gateway type. In the future we plan to add a Kafka Gateway to seamlessly interact with the Kafka bus of the 5G-MEDIA SVP.

In our architecture, the Sensor serves as Service Orchestrator (SO) that handles events requiring to execute specific orchestration and management flows, such as instantiation, which are specific to this service and might involve different combinations of VNFs comprising the service (as a whole) as well as the individual VNFs (e.g. individually instantiated serverless VNFs or collections thereof) in arbitrary order and at any point in time. These flows interact with specific serverless VNFs for instantiation, Day 1, Day 2 configuration, etc. These individual steps effectively act as serverless VNFMs. This way, we are no longer limited in the 5G-MEDIA SVP to the rigid orchestration flows of OSM and can cost-efficiently orchestrate scenarios involving serverless VNFs and (in the future) any dynamic VNF instantiation scenarios (also potentially spanning multiple NFVIs).
The following workflow executes a diamond workflow:

```
apiVersion: argoproj.io/v1alpha1
class: Workflow
metadata:
  generateName: dag-diamond-
spec:
  entrypoint: diamond
  templates:
    name: diamond
dag:
  tasks:
    - name: A
      template: echo
      arguments:
        parameters: [{name: message, value: A}]
    - name: B
      dependencies: [A]
      template: echo
      arguments:
        parameters: [{name: message, value: B}]
    - name: C
      dependencies: [A]
      template: echo
      arguments:
        parameters: [{name: message, value: C}]
    - name: D
      dependencies: [B, C]
      template: echo
      arguments:
        parameters: [{name: message, value: D}]
```

Table 17 - A simple example of an Argo flow

<table>
<thead>
<tr>
<th>command</th>
<th>echo</th>
<th>&quot;((inputs.parameters.message))&quot;</th>
</tr>
</thead>
</table>

Page 92 / 99
4.3.2. *Serverless VNFM and SO*

Figure 50 presents the extended serverless framework in more detail. The part shown inside the dashed rectangle on the right side represents the new extensions since previously reported Serverless Framework Specification in D3.2.

The key concept in the new approach is usage of a Kubernetes native workflow management, Argo. Argo Events offer two new Custom Resource Definitions (CRD): Gateway and Sensor. Both Gateway and Sensor instances are K8s pods created out of their respective yaml definitions. A Gateway instance is configured to receive events from the environment and pass it to a Sensor configured as a sink for this Gateway instance. In our current implementation, a Gateway is a Webhook one and an event can be an HTTP request. When a request is issued against the Gateway instance, it passes its payload to a Sensor that triggers Argo Workflows conditionally on the payload’s content.

We utilize this mechanism to individualize serverless SO. As shown in Figure 51, there is a new [mandatory] serverless VNF called Bootstrapper. This serverless VNF is created using our regular methodology OpenWhisk Docker actions. In the current implementation, at the time of a container creation, a yaml files, the definition for a Sensor, which will serve as a “private SO” for the service instance and private VNFM for the VNFs comprising the service is being added to the image (the files are picked from a local file system, designated by the user in a dockerfile. In addition, a Gateway yaml definition is provided in the Bootstrapper VNF image. The Gateway definition is standard and should not be touched by a developer. The Sensor definition is more intricate it involves using Argo dialect of yaml to specify orchestration workflows (such as individual VNF instantiation) that Argo Workflow will execute in response to an event (HTTP request that will be passed to the gateway).
5G-MEDIA serverless SO extension would supply a standard template that a developer needs to fill in to specify “instantiation”, “deinstantiation”, “day 1 configuration”, day 2 configuration” workflows.

In Step A, either an operator or an application/automation requests a network service instantiation from the OSM.

OSM instantiates each VNFD in the package in turn, including the Bootstrapper, which is, in fact, an OpenWhisk action that sets up a serverless orchestrator for this service, which is an Argo Sensor and Gateway pair. The Gateway URL is exposed publicly as an entry point to the serverless orchestration.

In the example given by the figure, VNF1 and VNF2 are started on the first path and VNF3 at some other point in time.

Each serverless VNF that should not be started on the first pass (i.e., when OSM instantiates the service initially) is marked using OpenWhisk annotation “start”:“fiveg-no-start-on-instantiation”. This annotation is not interpreted by OSM at all. However, Serverless VIM interprets this annotation as an OpenWhisk action annotation (which is part of the serverless “image” metadata) and instead of starting a VNF marked by it, stores a placeholder value in the VNFR for this VNF instance.

At some arbitrary point in time now, when a serverless SO (i.e., the Gateway) is contacted to start VNF3 (Step 6), the Sensor will run an instantiation and configuration flows for VNFD3, instantiating it and configuring it to be part of the rest of the service instance. At this point in time, VNF3 will have actual IP:PORT values that will be revealed by serverless VIM in a VNFR for that VNF replacing the placeholder value by an actual value.
5. Conclusions

This document accompanies the software prototype release of the 5G-MEDIA Operations and Configuration Platform. It focuses on the design of the platform and the associated algorithms for Quality of Service Control and Management tools in the 5G-MEDIA project.

The document presented: the overall MAPE architecture with an emphasis on the system for monitoring the infrastructure resources and the VNFs forming the deployed media applications; the design of the algorithms implementing the CNO; the design of serverless framework of the 5G-MEDIA SVP and the proposed enhancements to the ETSI NFV architecture to include serverless VNFs.

The software components forming the 5G-MEDIA SVP are available through the project’s public repositories as linked in Appendix A.
6. References


[Jordan19] https://www.jeremyjordan.me/autoencoders/ (Online, retrieved 20/11/19)


Appendix A: Software releases of the 5G-MEDIA Operations and Configuration Platform

The software, API specifications and user guides of the 5G-MEDIA Operations and Configuration Platform are available at [https://github.com/5g-media](https://github.com/5g-media).

Further details of the specific packages are available at [http://www.5gmedia.eu/outcomes/software](http://www.5gmedia.eu/outcomes/software).